Luc Brun
Mario Vento (Eds.)

LNCS 3434

Graph-Based Representations in Pattern Recognition

5th IAPR International Workshop, GbRPR 2005
Poitiers, France, April 2005
Proceedings

Springer

Lecture Notes in Computer Science 3434

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Luc Brun Mario Vento (Eds.)

Graph-Based Representations in Pattern Recognition

5th IAPR International Workshop, GbRPR 2005
Poitiers, France, April 11-13, 2005
Proceedings

Springer

Volume Editors

Luc Brun
GREYC, ENSICAEN
6 Boulevard du Maréchal Juin, 14050 Caen, France
E-mail: luc.brun@greyc.ensicaen.fr

Mario Vento
Università di Salerno, MIVIA Lab
Dipartimento di Ingegneria dell'Informazione ed Ingegneria Elettrica
Via Ponte Don Melillo, 1, 84084 Fisciano (SA), Italy
E-mail: mvento@unisa.it

Library of Congress Control Number: 2005922822

CR Subject Classification (1998): I.5, I.4, I.3, G.2.2, E.1

ISSN 0302-9743
ISBN-10 3-540-25270-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-25270-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11405405 06/3142 5 4 3 2 1 0

Preface

Many vision problems have to deal with different entities (regions, lines, line junctions, etc.) and their relationships. These entities together with their relationships may be encoded using graphs or hypergraphs. The structural information encoded by graphs allows computer vision algorithms to address both the features of the different entities and the structural or topological relationships between them. Moreover, turning a computer vision problem into a graph problem allows one to access the full arsenal of graph algorithms developed in computer science.

The Technical Committee (TC15, http://www.iapr.org/tcs.html) of the IAPR (International Association for Pattern Recognition) has been funded in order to federate and to encourage research work in these fields. Among its activities, TC15 encourages the organization of special graph sessions at many computer vision conferences and organizes the biennial workshop GbR. While being designed within a specific framework, the graph algorithms developed for computer vision and pattern recognition tasks often share constraints and goals with those developed in other research fields such as data mining, robotics and discrete geometry. The TC15 community is thus not closed in its research fields but on the contrary is open to interchanges with other groups/communities. Within this framework, the TC15 community decided to organize the fifth edition of its workshop jointly with the international conference Discrete Geometry for Computer Imagery (DGCI) organized by TC18 of the IAPR. Indeed, within the pattern recognition field, many graph-based algorithms are used to analyze the structures of the underlying objects. On the other hand, many algorithms of discrete geometry aim at finding the structures of unstructured sets of pixels or voxels. From this point of view, both communities aim at studying the structures of discrete objects. Both conferences were held in Poitiers, during the same week, with a common session on Wednesday 13th of April.

This volume contains the papers presented at the 5th Workshop on Graph-Based Representations in Pattern Recognition (GbR) organized by the IAPR TC15. The workshop was held at the University of Poitiers, France during April 11–13, 2005. The previous workshops in the series were held in Lyon, France (1997), Haindorf, Austria (1999) [3], Ischia, Italy (2001) [2], and York, UK (2003) [1].

The papers presented during this workshop, while all based on graphs, cover a wide range of research fields related to image processing and understanding. Indeed, one paper presented by Alain Bretto and Luc Gillibert uses graphs for low image processing such as noise attenuation and edge detection. Then several papers present several segmentation methods based on graphs together with improved graph data structures to encode fine properties of the partitions. Graphs or hierarchical graph data structures may thus be used to encode fine proper-

ties of the image's content. However, graphs may also be used to encode shape information. Many papers presented during this workshop encode a shape using either its skeleton or a set of points characterizing it. Given a graph describing an object (a shape, an image, a graphic, etc.) the next step consists of determining a measure of similarity between these graphs in order to derive a similarity measure between the underlying objects. Several papers devoted to graph matching attack this difficult problem using either exact or inexact algorithms. Algorithms based on graph kernels and the heat kernel equations provide alternative and interesting approaches to graph matching. Graph-matching algorithms may be pushed one step further by studying not only the matching between two graphs but also the classification of a set of graphs or the analysis of a sequence of graphs. Several papers presented during the workshop present novel and interesting ideas on these topics.

The papers presented here have all been reviewed by two reviewers and revised by their authors. The 50 papers submitted to the GbR were written by authors coming from 20 different countries located on five different continents. Based on these 50 submitted papers the Program Committee selected 18 of them as full papers and 17 of them as posters. We would therefore like to thank the members of the Program Committee and the additional reviewers for their help in ensuring that the papers were given a thorough and critical evaluation. We would also like to thank our sponsors who provided the material and financial help for the organization of this workshop.

April 2005

Luc Brun
Mario Vento

References

1. Edwin Hancock and Mario Vento (Eds.) *Graph-Based Representations in Pattern Recognition, GbRPR 2003*, volume 2726 of *LNCS*, York, UK, June/July 2003. IAPR-TC15, Springer. ISBN 3-540-40452-X
2. Jean-Michel Jolion, Walter G. Kropatsch, and Mario Vento, editors. *Graph-Based Representations in Pattern Recognition, GbR 2001*, Ischia, Italy, May 2001. IAPR-TC15, CUEN. ISBN 88-7146-579-2.
3. Walter G. Kropatsch and Jean-Michel Jolion, editors. *Graph-Based Representations in Pattern Recognition, GbR 1999*, Haindorf, Austria, May 1999. IAPR-TC15, OCG. ISBN 3-85403-126-2.

Organization

Program Co-chairs

Luc Brun
GREYC
ENSICAEN
6 Boulevard du Maréchal Juin
14050 Caen
France

Mario Vento
MIVIA Lab - Università di Salerno
Dipart. di Ing. Dell'Informazione ed Ingegneria Elettrica
Università di Salerno
Via Ponte Don Melillo,
1 Fisciano (SA) Italy

Program Committee

Horst Bunke	University of Bern, Switzerland
Terry Caelli	University of Alberta, Edmonton, Canada
Guillaume Damiand	Université de Poitiers, France
Luigi Pietro Cordella	Università di Napoli, Naples, Italy
Jean Michel Jolion	INSA Lyon, Lyon, France
Walter Kropatsch	Vienna Univ. of Technology, Vienna, Austria
Marcello Pellilo	Università Ca' Foscari di Venezia, Venice, Italy

Referees

I. Bloch (France)
A. Braquelaire (France)
L. Brun (France)
H. Bunke (Switzerland)
T. Caelli (USA)
L.P. Cordella (Italy)
G. Damiand (France)
P. Foggia (Italy)
F. Fontanella (Italy)
E.R. Hancock (UK)
C. Irniger (Switzerland)
J.-M. Jolion (France)
W.G. Kropatsch (Austria)
O. Lezoray (France)
M. Neuhaus (Switzerland)
M. Pellilo (Italy)
M. Revenu (France)
C. Sansone (Italy)
C. De Stefano (Italy)
M. Vento (Italy)

Sponsoring Institutions

ENSICAEN - 6 Bd. Maréchal Juin, 14050 Caen, France
GREYC - 6 Bd. Maréchal Juin, 14050 Caen, France
Université de Poitiers, France

Table of Contents

Graph Representations

Graphs and Linear Representations

Combinatorial Maps

Matching

Hierarchical Graph Abstraction and Matching

Inexact Graph Matching

Learning

Graph Sequences

Graph Kernels

Graphs and Heat Kernels

Hypergraph-Based Image Representation

Alain Bretto and Luc Gillibert

Université de Caen, GREYC CNRS UMR-6072, Campus II, Bd Marechal Juin BP, 5186, 14032 Caen cedex, France
{alain.bretto, lgillibe}@info.unicaen.fr

Abstract. An appropriate image representation induces some good image treatment algorithms. Hypergraph theory is a theory of finite combinatorial sets, modeling a lot of problems of operational research and combinatorial optimization. Hypergraphs are now used in many domains such as chemistry, engineering and image processing. We present an overview of a hypergraph-based picture representation giving much application in picture manipulation, analysis and restoration: the Image Adaptive Neighborhood Hypergraph (IANH). With the IANH it is possible to build powerful noise detection an elimination algorithm, but also to make some edges detection or some image segmentation. IANH has various applications and this paper presents a survey of them.

Keywords: Image Processing, Image Model, Segmentation, Edge Detection, Noise Cancellation, Hypergraph, Graph, Neighborhood Hypergraph.

1 Introduction

Graphs are very powerful tools for describing many problems and structures in computer sciences but also in physic and mathematics. But graphs only describe some binary relations and are not always sufficient for modeling some complex problems or data. Hypergraph theory, originally developed by C. Berge [8] in 1960, is a generalization of graph theory. The idea consists in considering sets as generalized edges and then calling a hypergraph the family of these edges. This concept models more general types of relations than graph theory do. In the last decades, the theory of hypergraphs has proved to be of a major interest in applications to real-world problems. These mathematical frameworks can be used to model networks, data structures, process scheduling, computations, and a variety of other systems where complex relations between the objects in the system play a dominant role.

To any digital image, a hypergraph, the Image Adaptive Neighborhood Hypergraph (IANH), can be associated and used for image processing. Many publications were written about this hypergraph model and many applications were found [1, 2, 4, 6]. This paper is a survey of different methods about image analysis and treatment based on this adaptive neighborhood hypergraph model.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 1–11, 2005.

First, we give basic definitions about hypergraphs and the definition of the IANH. Then we present an algorithm building the IANH and we study its properties and its complexity. Finally we illustrate some applications of the IANH to the image segmentation, the edge detection and the noise cancellation, three of the most important low level image processings. We give some powerful algorithms always based on the adaptive neighborhood hypergraph associated to an image. A set of examples is shown to illustrate the effectiveness of the algorithms.

2 Definitions

The general terminology concerning graphs and hypergraphs is similar to [8, 7]. All graphs in this paper are, finite, undirected, connected with no isolated vertex and simple, *i. e.* graphs with no loops or multiple edges. We denote a graph $G = (V; E)$. Given a graph G, we denote by $\Gamma(x)$ the *neighborhood* of a vertex x, *i. e.* the set consisting of all vertices adjacent to x which is defined by $\Gamma(x) = \{y \in V, \{x, y\} \in E\}$.

A *hypergraph* H on a set $\mathbf{X}$ is a family $(E_i)_{i\in I}$ of non-empty subsets of $\mathbf{X}$ called *hyperedges* with;

$$\bigcup_{i\in I} E_i = \mathbf{X}, \quad I = \{1, 2, \ldots, n\}, \quad n \in \mathbf{N}$$

Let us note $H = (\mathbf{S}; (E_i)_{i\in I})$. For $x \in \mathbf{S}$, a *star* of H (with center x) is the set of hyperedges which contains x, and is called $H(x)$. The *degree of* x is the cardinality of the star $H(x)$ denoted by $dx = Card(H(x))$.

Let $H = (\mathbf{S}; E = (E_i)_{i\in I})$ be a hypergraph, the *dual hypergraph* H^* is the hypergraph such that the set of vertices is the set of hyperedges, and the set of hyperedges is the set of stars of H. We can represent a hypergraph as in figure 1-(a).

A hyperedge E_i is *isolated* if and only if:

$$\forall j \in I,\ j \neq i,\ if\ E_i \cap E_j \neq \emptyset\ then\ E_j \subseteq E_i$$

An important structure from a hypergraph is the notion of *intersecting family*. A family of hyperedges is an intersecting family if the hyperedges from this family intersect two by two. We can distinguish two types of intersecting families:

- Intersecting families with an empty intersection.
- Intersecting families with an non empty intersection.

A hypergraph has the Helly *property* if each family of hyperedges intersecting two by two (intersecting family) has a non empty intersection (belongs to a star). As example in figure 1-(a) the hypergraph has the Helly property. Figure 1-(b) shows these two types of intersecting hyperedges. To each graph one can associate a hypergraph. Indeed, let $G = (X, E)$ be a graph, the hypergraph having the vertices of G as vertices and the neighborhood of these vertices as hyperedges (including these vertices) is called the *neighborhood hypergraph* of G and is denoted by:

$$H_G = (X, (E_x = \{x\} \cup \Gamma(x)))$$

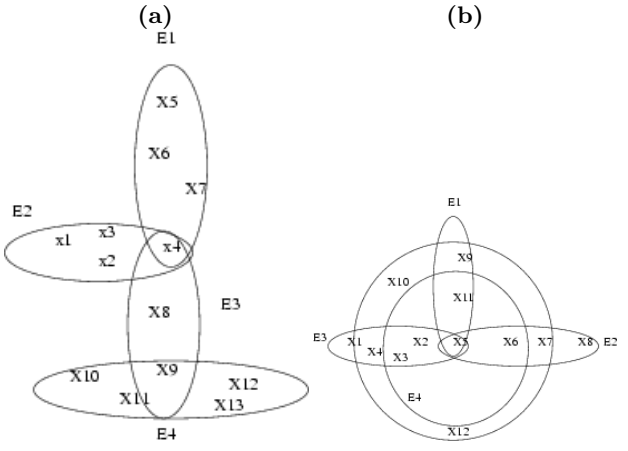


Fig. 1. (a) Example of hypergraph, the set of vertices is $\{x_1, x_2, \ldots, x_13\}$ and the set of hyperedges is $\{E_1, E_2, E_3, E_4\}$. (b) We have two types of intersecting families the first is the star the second has an empty intersection

3 Image Adaptive Hypergraph Model

First we recall some definitions about digital images. A distance d' on X defines a *grid* (a graph connected, regular, without both loop and multi-edge). A *digital image* (on a grid) is a two-dimensional discrete function that has been digitized both in spatial coordinates and in magnitude feature value. Throughout this paper a digital image will be represented by the application $I : X \subseteq Z^2 \to \mathcal{C} \subseteq Z^n$ with $n \geq 1$, where $\mathcal{C}$ identifies the *feature intensity level* and X identifies a set of points called *image points*. The couple $(x, I(x))$ is called a *pixel*. Let d be a distance on $\mathcal{C}$, we have a neighborhood relation on an image defined by:

$$\forall x \in X,\ \Gamma_{\alpha,\beta}(x) = \{x' \in X, x' \neq x \mid d(\mathcal{I}(x), \mathcal{I}(x')) < \alpha \text{ and } d'(x, x') \leq \beta\} \quad (1)$$

The neighborhood of x on the grid will be denoted by $\Gamma_\beta(x)$. So to each image we can associate a hypergraph called *Image Adaptive Neighborhood Hypergraph* (IANH): $H_{\alpha,\beta} = \big(X, (\{x\} \cup \Gamma_{\alpha,\beta}(x))_{x \in X}\big)$. The attribute α can be computed in an adaptive way depending on local properties of the image. If α is constant the hypergraph is called the *Image Neighborhood Hypergraph* (INH). Throughout this paper α will be estimated by the standard deviation of the pixels $\{x\} \cup \Gamma_\beta(x)$.

Algorithm: Image Adaptive Neighborhood Hypergraph
Construction of the hypergraph $H_{\alpha,\beta}$.

Data: Image I of size $m_x \times m_y$, and neighborhood order β
$X = \emptyset$;
For each pixel x **of** I, **do** ;
$\alpha =$ the standard deviation of the pixels $\{x\} \cup \Gamma_\beta(x)$;

$\Gamma_{\alpha,\beta}(x) = \emptyset$;
For each pixel y **of** $\Gamma_\beta(x)$, **do**
if $d(I(x), I(y)) \leq \alpha$ **then**
$\Gamma_{\alpha,\beta}(x) = \Gamma_{\alpha,\beta}(x) \cup \{y\}$;
end if
end for
$X = X \cup \{x\}$; $E_{\alpha,\beta}(x) = \{\Gamma_{\alpha,\beta}(x) \cup \{x\}\}$;
end for
$H_{\alpha,\beta} = (X, (E_{\alpha,\beta}(x))_{x \in X})$;
End

Data Structures Used: For each x, $\Gamma_{\alpha,\beta}(x)$ is a table of booleans, so $E_{\alpha,\beta}$ is a $m_x \times m_y$ table of tables. The set X is a $m_x \times m_y$ table of booleans.

Proposition 1. *Given β, the algorithm converges to a unique solution. Its complexity is in $O(n)$ (n standing for the pixel number of the image). (For the proof report to [2]).*

4 Detection of Impulsive Noise

A common type of corruption that occurs in image data is corruption by an impulsive noise process. Attenuation of noise and preservation of details are usually two contradictory aspects of image. Various noise reduction algorithms make various assumptions, depending on the type of imagery and the goals of the restoration [9],[10].

In this section, we present a noise cancellation algorithm that exploits a lack of homogeneity criterion. We consider that the global homogeneity characterizes regions, local homogeneity characterizes edges, no homogeneity characterizes a noise. A noise reduction algorithm is based on the following criterion: binary classification of hyperedge of image (H_0 noisy hyperedge and H_1 no noisy hyperedge) and filtering the noisy parts.

Noise Definition - We will call disjoined chain a ordered succession of hyperedges disconnected two by two and build on some adjacent pixels. A disjoined chain is *thin* if the cardinality of each hyperedge is equal to 1. To model a noise we propose the following definition:

We say that $E_{\alpha,\beta}(x)$ is a *noise hyperedge* if it verifies one of the two conditions:

- The cardinality of $E_{\alpha,\beta}(x)$ is equal to 1 and $E_{\alpha,\beta}(x)$ is not contained in disjoined thin chain having five elements at least.
- $E_{\alpha,\beta}(x)$ is an isolated hyperedge and there exists an element y belonging to the open neighborhood of $E_{\alpha,\beta}(x)$ on the grid, such that $E_{\alpha,\beta}(y)$ is isolated. (i.e. $E_{\alpha,\beta}$ is isolated and it has an isolated hyperedge in its neighborhood on the grid).

This definition allows a good discrimination between edge pixels and noisy pixels. The lemma below shows that a noisy hyperedge must be isolated.

Lemma 1. *If the cardinality of a hyperedge is equal to one, then this hyperedge is isolated. (For the proof report to [6]).*

With the noise definition above, a noise detection algorithm is simple. The IANH is built and all the hyperedges satisfying the conditions of the noise definition are selected. This selection is separated in two step, first the detection of the isolated hyperedges and then, in the set of the isolated hyperedges, the detection of the noisy hyperedges.

Algorithm: Noise Detection
Data: Image I of size $m_x \times m_y$, IANH $H_{\alpha,\beta}$.
Determination of isolated hyperedges of $H_{\alpha,\beta}$
  For each vertex x **of** $H_{\alpha,\beta}$, **do** ;
    $E_x^{'} = \bigcup_{y \in E_{\alpha,\beta}(x)} E_y$;
      If $E_x^{'} == E_x$, (E_x is an isolated hyperedge) **then**
        If Cardinality of E_x is equal to one **then**
          $ISO[x] = E_x$;
        Else
          $IS[x] = E_x$;
        end if
      end if
  end for
Detection of noise hyperedges of $H_{\alpha,\beta}$
  For each E_x **of** $ISO[]$, **do**
    For each E_y **of** $ISO[]$, **and** $x \neq y$ **do**
      For each E_z **of** $ISO[]$, **and** $x \neq z$ **and** $y \neq z$ **do**
        If $y, z \notin \Gamma_\beta(x)$ **then**
          $NH[x] = E_x$;
        end if
      end for
    end for
  end for
  For each E_x **of** $IS[]$, **do**
    If there is $y \in \Gamma^o(E_x)$ such as $E_{\alpha,\beta}(y) \in ISO[] \cup IS[]$, **then**
      $NH[x] = E_x$;
    end if
  end for
End

Data Structures Used: The data structures used for the IANH and its hyperedges are the same that in the IANH construction algorithm. The sets ISO, ES and NH are some $m_x \times m_y$ tables of hyperedges.

The complexity of this algorithm is in $O(n^3)$. This algorithm has been tested on several images in order to show how effective our method is. This method has a great advantage over the class of linear filters; it preserves the edges, so the additional complexity provides some additional results. Some experimental results can be found in [6]. Some visual examples are shown in figure 2.

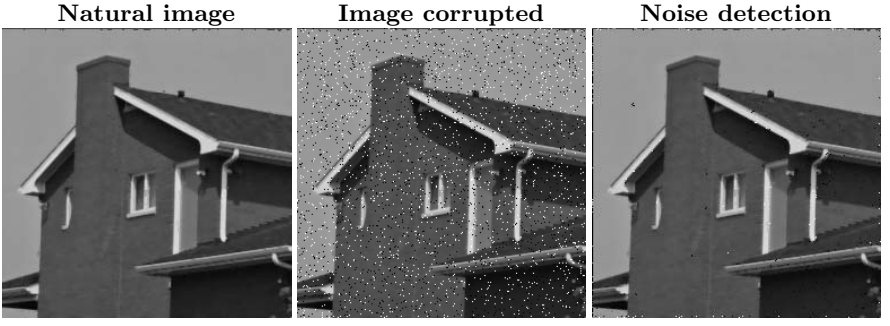


Fig. 2. Example of IANH-based noise detection and cancellation

5 Segmentation

One of the first major step of low level vision is segmentation. Segmentation is the process which consists in partitioning an image into some non-intersecting regions such that each region is homogeneous and that the union of two adjacent regions is never homogeneous. The algorithm below will give us the segmentation of an image. This algorithm is based on the detection of stars in the hypergraph model.

The algorithm process can be divided into in two main parts. In the first part, a covering of the image by a minimal set of stars is computed. In the second part, selected stars are aggregated to obtain the regions. This regions are the segments of the image.

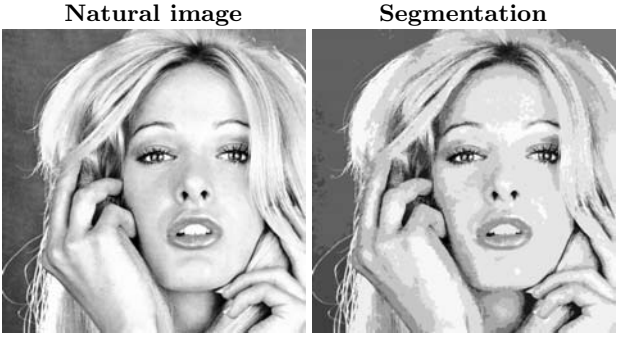


Fig. 3. Example of IANH-based image segmentation

Algorithm: Covering and Selection for Segmentation

Data: Image I of size $m_x \times m_y$, IANH $H_{\alpha,\beta}$.

Choosing a cover of the image by a minimal set of stars.

Chose a minimal cover of the image, $E = \{H(x_1), H(x_2), \dots H(x_n)\}$, such that any pixel of the image belongs to at most one hyperedge of at least one star of the set E.

Building aggregate areas.
For each $H(x_i)$ in E, **do**
$I(x_i)$ = grey level of the center of the star $H(x_i)$;
$Agg_i = \emptyset$ (initialization on a new aggregation area)
For each $H(x_j)$ intersecting with the star $H(x_i)$, **do**
If $I(x_j)$ in $[I(x_i) - \alpha, \quad I(x_i) + \alpha]$, **then**
$Agg_i = Agg_i \cup$ (vertices of $H(x_j) \cup H(x_i)$);
end if
end for
end for
Reducing the number of areas.
For each aggregate area Agg_i, **do**
g_i = center of gravity of Agg_i;
min_i = minimum grey level of the centers of the stars of Agg_i;
max_i = maximum grey level of the centers of the stars of Agg_i;
med_i = medium grey level of the centers of the stars of Agg_i;
end for
For each aggregate area Agg_i, **do**
For each area Agg_j intersecting with Agg_i, **do**
If g_i **or** g_j is in $Agg_i \cup Agg_j$,
and min_i in $[min_j - \alpha,\ min_j + \alpha]$,
and max_i in $[max_j - \alpha,\ max_j + \alpha]$,
and med_i in $[med_j - \alpha,\ med_j + \alpha]$, **then**
Aggregate Agg_i and Agg_j;
end if
end for
end for
Assigning each star to an aggregation area to obtain a partition.
Chose the area Agg_i containing the greatest number of stars
For each star H in Agg_i, **do**
Assign the star H to Agg_i;
For each aggregate area $Agg_j \neq Agg_i$, **do**
Remove H from Agg_j;
end for
end for
Repeat chose until all the stars have been assigned.
Assigning the pixels generating edges
For each pixel x in I, **do**
If x in several stars, **then**
assign x to the area of the star center whose grey level is the closest
end if
end for

Finally, the pixels lefts are assigned to the area of the neighboring pixel which has already been assigned, and whose grey level is the closest.

Data Structures Used: The sets E is a $m_x \times m_y$ tables of hyperedges structured as the IANH in the other algorithms. Each aggregate area Agg_i is a list. The set of all the aggregate areas is also a list.

Visual example can be see in figure 3. Experimental results can be found in [1].

6 Edge Detection

Another important aspect of image analysis is the edge detection. In a grey level image containing homogeneous objects, an edge is a boundary between two regions of different constant grey levels. If the difference is clear-cut between regions we have ideal edges. But there are several factors that degrade edges, such as noise or irregularities of the surface. An effective method for detecting edges must be based on an adequate definition of these edges.

Let H be a hypergraph, we will call *triangle* an intersecting family of three hyperedges without common vertex. A *triangle intersecting family (TIF)* is an intersecting family which is not contained in a star. This means that the set of hyperedges family of a TIF do not share a common vertex.

Theorem 1. *Let H be a hypergraph and let $\mathcal{IF} = \{E_1,\ E_2,\ \ldots E_k\}$ be an intersecting family. This family is a TIF if and only if $\mathcal{IF}$ contains a triangle. (For the proof report to [2]).*

Proposition 2. *Let I be an image and $H_{\alpha,\beta} = \big(X, E_x = (\{x\} \cup \Gamma_{\alpha,\beta}(x))_{x\in X}\big)$ be its INAH. $H_{\alpha,\beta}$ is isomorphic to $H^*_{\alpha,\beta}$. (For the proof report to [2]).*

We will said that a pixel $(x, I(x))$ belongs to edges if it belongs to a triangle of $\mathcal{H}_{\alpha,\beta}$. So we can define an algorithm for detecting edges in an image. This one is based on three steps: (1) Construction of the line-graph of $H_{\alpha,\beta}$ with $\beta = 1$ (*Algorithm: Line-graph*). (2) 3-cliques detection in the line-graph. This step gives us the intersecting family of hyperedges in the image hypergraph (*Algorithm: Detection of* 3*-cliques*). (3) Test if the 3-cliques stand for a triangle in $H_{\alpha,\beta}$ (*Algorithm: Hypergraph triangle test*).

All the pixels that belong to triangles in the third step of the algorithm correspond to the image edges. The following algorithm constructs the line-graph of the Image Adaptive Neighborhood Hypergraph. It easy to see that its complexity is $O(n^3)$ where n is the pixel's number.

Algorithm: Line-Graph.
Data: Image I, Image Adaptive Neighborhood Hypergraph $H_{\alpha,\beta}$.
Construction of the vertice's set of $L(H_{\alpha,\beta})$
  For each hyperedge E_x **of** $H_{\alpha,\beta}$, **do**
    $V(x) = e_x$;
  end for
Construction of the edge's set of $L(H_{\alpha,\beta})$.
  For each hyperedge E_x of $H_{\alpha,\beta}$, **do**
    For each pixel y **of** E_x, **do**

```
            For each pixel z ≠ y of E_x, do
                If BE[yz] = false, then
                    E[xy] = {e_x; e_y};
                    BE[xy] = true;
                end if
            end for
        end for
    end for
End
```

Then we must detect the 3-cliques. The edge number of $L(H_{\alpha,\beta})$ is in $O(n)$ and there are n vertices, so the complexity of this algorithm is in $O(n^2)$.

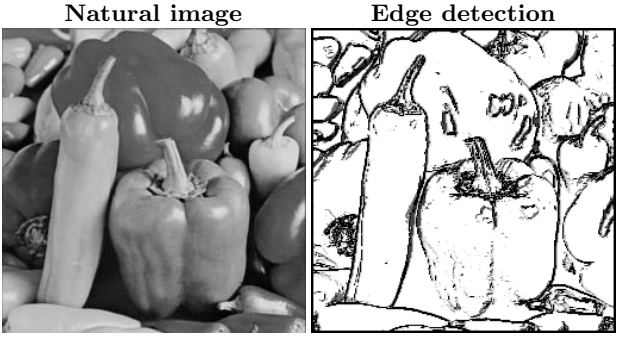


Fig. 4. Example of IANH-based edge detection

```
Algorithm: Detection of 3-Cliques.
Data: Line-graph L(H_{α,β}).
For each edge E[xy] of L(H_{α,β}), do
    For each vertex z of L(H_{α,β}), and z ≠ x, y do
        If BE[xz] = true and BE[yz] = true, then
            T[xyz] = {e_x, e_y, e_z};
        end if
    end for
end for
End
```

Finally we have to test if the 3-cliques stand for a triangle in $H_{\alpha,\beta}$. The triangle number of $L(H_{\alpha,\beta})$ is in $O(n^3)$, so the complexity of this algorithm is in $O(n^3)$.

```
Algorithm: Hypergraph Triangle Test.
Data: Image Adaptive Neighborhood Hypergraph H_{α,β} and set of triangle T[].
```

For each triangle $T[xyz]$, **do**
If $E_x \cap E_y \cap E_z = \emptyset$ **then**
$edges[xyz] = \{x\} \cup \Gamma_\alpha(x) \cup \{y\} \cup \Gamma_\alpha(y) \cup \{z\} \cup \Gamma_\alpha(z)$;
end if
end for
End

The method has been tested on a set of images. The results of our algorithm have beeen compared with the classical Canny-Deriche filter. Our method gives more meaningful edge maps and looks promising and more robust. Some visual examples are shown in figure 4.

7 Conclusion

Hypergraph-based image representation, as shown in this paper and in a lot of publications [1, 2, 4, 6], can be used for low-level treatment and gives some good results, only counterbalanced by the high complexity of the corresponding algorithms. Such representation is not limited to segmentation, edge detection and noise cancellation. Very wide perspectives are opened by the use of hypergraphs-based models. By example, loseless and lossy picture compression can be performed with an appropriate hypergraph-based picture representation, as shown in [3]. In fact hypergraphs are very promising tools and can be applied in many other images process. Further works will be focused on:

- Conception of algorithms in high level image processing using hypergraphs (by example for shape recognition).
- Hypergraph-based compression for 3D images.
- Hypergraph-based entropy on 2D and 3D images.
- Generalisations of the IANH for 3D images, generalisation inducing some segmentation, edge detection and noise cancellation algorithms on 3D images.

References

1. A. Bretto, J. Azema, H. Cherifi, and B. Laget. *Combinatoric and image processing*. Computer Vision, Graphic and Image Processing (Graphical Model and Image Processing), 59(5):128–132, September 1997.
2. A. Bretto. *Introduction to Hypergraph Theory and its Applications to Image Processing*. Mongraphy in: Advances in Imaging and Electron Physics. Academic Press Elsevier, vol. 131, march 2004.
3. A. Bretto and L. Gillibert. *Hypergraph Lossless Image Compression*, preprint, submited to ITCC.
4. S. Chastel, P. Colantoni and A. Bretto. *Displaying Image Neighborhood Hypergraph*. 10th International Conference on Discrete Geometry for Computer Imagery (DGCI'2002).

5. A. Bretto and B. Laget. Helly *property associated with the discrete planes*. Southwest journal of pure and applied mathematics, 1:56–63, 1998.
6. S. Rital, A. Bretto, H. Cherifi and D. Aboutajdine. *Application of Adaptive Hypergraph Model to Impulsive Noise Detection*. CAIP'2001, Springer-Verlag, 2001, Warsaw, Poland, September 5-7, Lecture Note in Computer Sciences, 34–42.
7. C. Berge. *Graphe*. Dunod, 1983.
8. C. Berge. *Hypergraphs*. North-Holland Mathematical Library, 1989.
9. T. Chen and H. R. Wu. *Adaptive Impulse Detection Using Center Weighted Median Filters*. IEEE Signal Processing Letters, **series 8**, 2001, 1–3.
10. T. Chen and H. R. Wu. *Application of Partition-Based Median Type Filters for Suppressing Noise in Images*. IEEE Transactions on Image Processing, 2001.

Vectorized Image Segmentation via Trixel Agglomeration

Lakshman Prasad and Alexei N. Skourikhine

Los Alamos National Laboratory,
P.O. Box 1663, MS D436, Los Alamos, NM 87545, U.S.A
{prasad, alexei}@lanl.gov

Abstract. We present a broad algorithmic framework for transforming an image comprised of pixels into a vectorized image segmented into polygons that can be subsequently used in image processing and understanding. A digital image is processed to extract edge pixel chains and a constrained Delaunay triangulation of the edge contour set is performed to yield triangles that cover the pixelated image without crossing edge contours. Each triangle is attributed a color by a Monte Carlo sampling of pixels within it. A combination of rules, each of which models an elementary perceptual grouping criterion, determines which adjacent triangles should be merged. A grouping graph is formed with vertices representing triangles and edges between vertices that correspond to adjacent triangles to be merged according to the combination of grouping rules. A connected component analysis on the grouping graph then yields collections of triangles that form polygons segmenting and vectorizing the image.

1 Introduction

1.1 Background

Image segmentation is a critical processing step in the automation of image understanding by computers. Broadly speaking, image segmentation is analogous to the process of perception in human vision which does not assume *a priori* knowledge of the objects in an image; rather, it uses spectral and structural cues in an image to parse the image into visually distinct parts. Segmentation consists of decomposing an image into regions corresponding to features and objects that define the semantic content of the image. Image segmentation sets the stage for object detection and recognition by providing a higher-level representation of an image in terms of regions of uniform color/intensity and structural relevance. Objects in images are typically contiguous subsets of such regions. Thus, image segmentation transforms an image from a low-level pixel-based representation to an intermediate-level region-based representation that enables the piecing together of the *jigsaw puzzle* of regions into individual objects of the image. Several segmentation methods have been developed over the decades, most of which can be classified into one of two categories, namely methods that start with *pixels* as feature primitives (e.g., [1]) and group them based on their spectral and textural cohesiveness into regions, and methods that start with

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 12–22, 2005.

edgels (edge elements) detected in images as feature primitives (e.g., [2]) and seek good continuations and closures of groups of these edgels to obtain contours of form. While these two approaches have yielded some of the best results to-date, they are far from perfect. In particular, they can lead to erroneous groupings. The first approach, by focusing on regional uniformities, can result in false edges, while the second approach, by not taking into account the regional correspondences of edges, can result in false continuations and hence false regions.

1.2 Motivation

Cognizant of the inextricable duality of image regions and their contours, and therefore the impossibility of extracting complete information of either aspect of images independent of the other, we adopt a hybrid region-contour approach. In particular, we seek an image feature primitive that is well adapted to the region-contour duality of the segmentation process.

One of the ultimate goals of image segmentation is the delineation of shapes of objects in images for purposes of object recognition and, eventually, image understanding. Two important descriptors of a shape are its contour and skeleton. A shape's skeleton provides regional information about the shape. A well-known characterization of a shape's skeleton is its medial axis transform (MAT) [3]. The MAT of a shape with uniformly sampled contour is closely related to the generalized Voronoi diagram [4] of the contour points, which is the geometric dual of the constrained Delaunay triangulation (CDT) [4] of the planar straight line graph defined by the shape contour. In earlier works [5, 6] we derived the chordal axis transform (CAT) as a useful variant of the MAT for discrete shapes solely from the CDTs of discretized shape contours. The non-boundary edges of triangles of a discrete shape's CDT are transverse to the shape's skeleton. Such edges relate contour points that are across from each other with respect to the skeleton, providing a regional binding of contour elements. The vertices of a triangle in a discrete shape's CDT define an elemental region interior to the shape, providing a contour binding of regional elements. Thus the triangles of a shape's CDT are well suited as feature primitives that capture the region-contour correspondence of shapes.

In this paper we extend this notion to images by obtaining the CDT of the planar straight line graph consisting of chains of edge pixels resulting from an edge detection process. The triangles so obtained are additionally attributed the average color of the pixels in their interior. We refer to these attributed triangles as TRIangular eXcision ELements or *trixels*, so called because the image will be *excised* or cut along certain edges of the trixels to obtain polygonal segments. The trixels of an image are then region-contour feature primitives that form a complete and irregular tiling of the image that is adapted to the edges of the image. Trixels combine the properties of both pixels and edgels. They behave like pixels insofar as they are the regional units of uniform color. They behave like edgels insofar as their edges are either a priori image edge units obtained from edge detection, or are candidate edge units that later may serve to connect two edge chains in the process of segmentation. In what follows, we will use this trixel representation of images to achieve polygonal image segmentation and vectorization. We note that constrained triangulations of contour sets have been used by others [8, 9] to obtain regional decompositions of images.

However, we explicitly exploit the region-contour duality property of constrained Delaunay triangulations for modeling perceptual grouping to perform region-contour inferences that yield visually faithful regional segmentations and contour completions in images [7].

2 The Framework

2.1 Outline

Briefly, in our approach, contours comprised of pixel chains are extracted from a digital image (Fig. 1) via an edge detection algorithm such as the Canny edge detector [7] and used as an initial incomplete set of feature constraints (Fig. 2). A constrained Delaunay triangulation [11] is applied to the planar straight line graph defined by these contours (Fig. 3), with the resulting triangle vertices anchored at contour pixels and with no triangle edge crossing a contour. The image region under each triangle is sampled using a Monte Carlo technique to estimate the color of the triangle (Fig. 4).

Fig. 1. Input image

Fig. 2. Canny edge contours

Remark 2.1.1. It is reasonable to expect a nearly uniform coloration of each triangle's underlying pixels, as a significant variation would have created edges within the region leading to a different contour set, and hence triangulation, than the one at hand.

Triangles are attributed their estimated color and termed trixels. The trixels form a complete irregular tessellation of the image that conforms to the image's detected edge structure. They are the new hybrid region-contour feature primitives comprising the image.

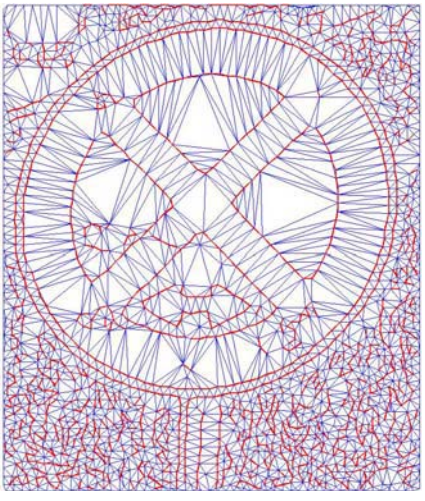


Fig. 3. Delaunay triangulation of contours

Fig. 4. Monte Carlo sampling of triangle colors

The edges of trixels act as region separators as well as contour connectors. Specifically, whenever a trixel edge is deleted, two region elements are connected into a single region element. Likewise, whenever a trixel edge is retained, two contours are connected into a single contour with the retained trixel edge serving as the interpolating contour element. The trixel edges thus act as switches that explicitly model and control the region-contour duality in the segmentation process. We draw upon certain elementary notions in perceptual organization, such as spatial proximity, continuity, spectral proximity, etc., as evidenced in human vision [12] to develop trixel grouping criteria that determine whether a trixel edge should be retained or not. While each of these grouping criteria weighs in on the fate of an edge, it is a combination of these criteria that finally determines the edge's retention or deletion. We note that the proximity and region-contour duality properties embodied by Delaunay triangulations allow perceptually meaningful continuations and completions of contours. This is *not* generally true of other triangulations. A trixel grouping graph with vertices representing the image's trixels is created, with edges connecting vertices corresponding to trixels whose common edge has been deleted by the combination of grouping criteria (Fig. 5).

Remark 2.1.2. This graph is, in essence, the skeleton of the inter-contour image space that has been segmented due to the retention of certain edges by the combined actions of the grouping criteria. The connected components of this graph therefore correspond to the skeletons of the segmented regions of the image.

Connected components of the trixel grouping graph are obtained via a graph traversal algorithm. The connected components yield agglomerations of pairwise-adjacent trixels that form polygonal segments of the image. Furthermore, the trixels of each connected component are assigned the average color of the trixels in that component. These polygonal region segments conform to the image edges as well as to the spectral uniformity of image regions (Fig. 6).

Fig. 5. Trixel grouping graph showing connected components

Fig. 6. Polygonal segmentation of input image

2.2 Discussion of Key Stages

Edge Detection. There are many methods for extracting edge pixels from images that may be employed in our algorithm framework. However, we prefer the Canny edge detector [10] because of the quality of edges it generates and the options it allows for varying the edge properties.

Contour Chain Extraction. The edge detection process yields a binary image consisting of edge pixels. To this we add all pixels on the boundary of the image as edge pixels. We then obtain contour chains by iteratively traversing contiguous edge pixels. Each contour chain is attributed a contour chain number, with all points in it inheriting this number.

Constrained Delaunay Triangulation. The planar straight line graph consisting of contour chains is subjected to a constrained Delaunay triangulation (CDT) [11]. This decomposes the image area into triangles that are anchored at vertices of the contour chains without any of the triangle edges intersecting the contour chains. The triangle edges may, however, coincide with edges of the contour chains.

Trixel Definition. As mentioned before, trixels are color-attributed triangles. The interior of each triangle is randomly sampled a few (say 5 to 10) times, to obtain an efficient and reliable estimate of average color of the underlying image pixels. In our experience, increasing the number of samples or considering all pixels in a triangle does not improve the estimation significantly. The triangle is explicitly attributed this average color to yield a trixel. The trixel also has other, implicit, attributes: since the trixel is geometrically specified by vertices, and vertices inherit the contour chain index attribute, the trixel contains information of what pairs of contour chains its edges connect. It also has information about its neighboring trixels via its edges and the triangulation.

Trixel Edge Attribution. Edges of trixels separate regions as well as connect contours. Thus edges serve as gates whose opening and closing execute the segmentation process in our framework. Edges are specified by their vertices which are attributed the indices of the contour chains they belong to. In addition, edges are attributed the indices (and hence the colors) of the two adjacent trixels they separate. These attributes help decide which edges should be retained as contour connecting, region separating elements. The set of all trixel edges that belong to two trixels will be subject to filtering, resulting in trixel groupings.

Perceptual Edge Properties. We model certain elementary perceptual cues via Boolean filters on edges. These cues, combined via Boolean expressions, yield composite edge filters that selectively retain edges to achieve image segmentation. We briefly describe some exemplary Boolean edge filters in Table 1. Four broad classes of perceptual cues are employed here, namely spatial proximity, spectral proximity, contour continuity, and region continuity. Each Boolean filter identified below characterizes a perceptual cue or the negation of it. The action of each Boolean filter on the edge list results in a Boolean vector with a '1' in the i^{th} position if the i^{th} edge satisfies the property modeled by the filter, and '0' otherwise.

Table 1. This table indicates the type of perceptual cue, the name of the Boolean filter addressing the cue, the filter's support of the cue (i.e., whether it affirms or negates the cue) and a brief description of the condition to be satisfied by the edge for the filter to take the value 1

Type	Name	Support	Description
Spatial proximity	TooLong	-	Length above a threshold (e.g., 2.5 times median triangle edge length)
	NotShortest	-	Shortest edge of neither flanking trixel
	Longest	-	Longest edge of at least one flanking trixel
Spectral proximity	NearColor	+	Scaled intensities of flanking trixels are within threshold distance for intensity images, or inner product of unit RGB vectors of flanking trixels is within threshold distance of 1 for color images
Contour-continuity	Canny	+	Coincides with a contour chain edge
	EndLink1	+	Connects (with minimum turn) end point of a contour to the interior of another contour (transversality)
	EndLink2	+	Connects two contour endpoints
Region continuity	SameContour	+	At least one flanking trixel has all its vertices lying on the same contour chain, and at least one vertex of the edge is not a contour end point
	Just2Contours	+	At least one flanking trixel has all its vertices on one of two contours and at least one vertex of the edge is not a contour end point
	Junction	-	At least one flanking trixel has all its vertices on different contours

Remark 2.2.1. It is possible to model many other perceptual cues via Boolean edge filters such as the ones presented in Table 1. In this paper, however, we have restricted ourselves to only the above few for simplicity and brevity of presentation.

Trixel Edge Filtering. Each of the filters listed in Table 1 models a pure criterion that determines the retention or deletion of edges in the absence of other criteria. Realistically speaking, however, all such criteria jointly come into play and a composite of these criteria ultimately decides whether an edge is retained or deleted. This is motivated by analogy to the human perceptual process wherein multiple perceptual organization criteria determine visual perception. We model this composite edge filter as a Boolean expression in the elementary edge filters. An example of a composite filter C is

$$\begin{aligned} C = (\ & TooLong \mid (\ NotShortest \ \&\ \sim EndLink2\ ) \mid Longest \mid NearColor \mid \ldots \\ & \ldots\ SameContour \mid Jus2Contours \mid (\ Junction \ \&\ NotShortest\ )\ ) \ \&\ldots \\ & \sim(\ \sim TooLong \ \&\ (\ Canny \mid (\ (\ EndLink1 \mid EndLink2\ ) \ \&\ \sim Longest)\ )\ ); \end{aligned} \tag{1}$$

There are many variations that produce satisfactory segmentation results. For example, a composite filter that does not use spectral proximity to group trixels is given by

$$\begin{aligned} C = TooLong \mid \sim(Canny \mid EndLink1 \mid EndLink2 \mid (Junction \ \&\ \sim(NotShortest \mid \\ \ldots\ Longest)\ )\ ); \end{aligned} \tag{2}$$

An edge for which the filter C takes the value 1 will be deleted, merging the two trixels flanking the edge. Thus the action of C on the set of all trixel edges results in mutually disjoint polygons, comprised of pairwise adjacent trixels, tiling the image.

The formulation of composite filters that are optimized to some measure of segmentation quality, that are adapted to a class of imagery (such as IR, X-ray, etc), or that are tuned to segment certain features, are all challenging problems. The algorithm presented here provides a general framework for developing improved and specialized segmentation algorithms to address these problems. For instance, it is possible to define elementary edge filters that take on continuous values. These filters can then be combined using fuzzy, neural, or evolutionary learning formalisms to yield high quality segmentation algorithms. In this sense, our algorithmic framework is a master algorithm template for image feature synthesis.

Analyzing the Trixel Grouping Graph. We represent the image trixels as vertices of a trixel grouping graph, with an edge between two vertices corresponding to adjacent trixels. It is easy to see that this graph is planar. An edge between two vertices is given the weight 1 if they represent adjacent trixels whose common edge is marked for deletion by the composite edge filter C. The remaining edges of the graph are given the weight 0. The edge-weight-1 connected components of this grouping graph correspond to contiguous sets of pairwise adjacent trixels that form polygonal image segments. This characterization of the grouping graph is slightly different from what we have alluded to earlier in this paper for simplicity of exposition, where edges connected only those vertices that corresponded to adjacent trixels whose common trixel edge was deleted by the composite perceptual grouping filter. The connected but edge-weighted trixel grouping graph described here later allows the creation of a

higher level quotient graph for further grouping of segmented regions. Thus graph-representations provide a natural and elegant paradigm for obtaining an irregular pyramidal hierarchy of successively higher order features and their interrelationships, as well as efficiently perform groupings using a rich lore of graph algorithms.

We use a depth-first-search traversal of the grouping graph to extract agglomerations of trixels that form the segments. All trixels of the same edge-weight-1 connected component are attributed a common component number. The edge-weight-1 subgraphs of the segments of the trixel grouping graph yield skeletons of the segments. The fact that the vertices represent Delaunay triangles is additionally helpful in characterizing the segment's shape via attributed skeletons such as the CAT [5, 6]. It also assists in readily and efficiently computing useful physical attributes such as area, average local diameter, perimeter, and genus (i.e., number of holes) of the segment for measuring feature saliency and recognizing objects. All the trixels belonging to a segment are reassigned the average color of the trixels in the segment. Thus the segment gets a uniform color attribute and the image is segmented into polygonal regions.

Image Vectorization and Compression. The polygonal segmentation thus obtained can be represented as a vector image by specifying the contour of each polygon along with its fill color. This gives a scalable representation for flexible rendering [7]. In fact, we have implemented a vectorized representation using the SVG (scalable vector graphics) standard proposed by the World Wide Web Consortium (W3C). Our segmentation scheme, apart from providing useful structural information of image features for image understanding, yields images of good visual quality (Fig. 7). In addition, the polygonization yields, on an average, a lossy image compression (for JPEG images) ratio of ~ 1:4. This makes it an attractive method for web-based and low-bandwidth communication applications.

Grouping Polygons into Objects. Following segmentation of the image into polygons, new, higher order perceptual criteria may be defined based on shape, contour, and color relationships of adjacent polygons. A segment grouping graph can be defined as the quotient graph of the trixel grouping graph with respect to the edge-weight-1 connected subgraphs obtained above. The higher order perceptual criteria may then be used to weight this quotient graph's edges (which are quotients of the weight-0 edges in the trixel grouping graph) and group segments into larger polygons corresponding to objects, features, or textures by edge-weight-based traversals and strong component extractions on the graph. For instance, based on similarity of hues, directionalities, and areas, a group of polygonal segments corresponding to blades of grass in a segmented image containing a lawn may be grouped into a single region of texture. A detailed discussion of such higher order grouping is outside the scope of this paper. We limit ourselves to the illustration of a simple example of grouping of segments by hue in Fig. 8.

Computational Issues. The Canny edge detector is a fast linear algorithm whose computational time complexity is a linear function of the total number of pixels in an image. Contour chaining is linear in the number of detected edge pixels which is typically an order of magnitude smaller than the total number of image pixels. Constrained Delaunay triangulation runs in $o(n.log(n))$ time as a function of the number n of contour points which is less than (in case of subsampling) or equal to the

Original JPEG image (259 KB)

Vectorized segmentation (61 KB)

Original JPEG image (365 KB)

Vectorized segmentation (160 KB)

Original JPEG image (37 KB)

Vectorized segmentation (11 KB)

Fig. 7. Sample JPEG images and their vectorized segmentations via trixel grouping

Fig. 8. A JPEG image, its vectorized segmentation, and grouping of segments by hue

number of edge pixels. The Monte Carlo sampling of triangle intensities is linear in the number of trixels, which is at most twice the number of contour points. Connected component analysis of the planar grouping graph is linear in the number of trixels as it uses a depth-first-search traversal. Thus the overall efficiency of our algorithm is high, making the method well suited for real-time applications.

3 Summary and Perspectives

We have briefly presented an algorithmic framework for segmenting images based on hybrid region-contour properties of constrained Delaunay triangulations of image edges. A key novelty is the geometric modeling of empirically observed elementary perceptual organization criteria, as evidenced in human vision, to perform segmentation. The application of the perceptual criteria to the triangle feature primitives results in cuts in the skeleton graph of the regions bounded by image edges. A connected component analysis on the excised skeleton graph yields perceptually meaningful polygonal segmentation of images.

While the basic criteria of perceptual organization have been known for a long time [12], the mechanism of their interactions in the process of perception is largely unknown. In conjunction with optimization and learning formalisms such as fuzzy logic, neural networks, and evolutionary algorithms, our framework offers a flexible paradigm to explore these interactions and construct useful perceptual filters.

It is quite likely that there is no single mode of interaction of the basic perceptual criteria that is applicable to all images, or even to the same image in all locations. A more plausible scenario is that there are a finite set of "legitimate" interactions that may be characterized by a library of composite filters C, with a different C required in different parts of an image to obtain a perceptually most meaningful segmentation. As part of our ongoing work we propose use the framework described in this paper to construct and employ multiple composite perceptual filters C as local constraints in the minimization of cost functions, such as the Mumford-Shah functional [13] for segmentation. Our goal is to reduce the solution search space as well as obtain spatially adaptive perceptually meaningful segmentations. The success of our effort will lead to the development of a new generation of segmentation algorithms that are better adapted to real-world imagery and specialized classes such as X-ray, infrared, and medical imagery.

Acknowledgements

This work has been fully supported by the U. S. DOE under contract No. W-7405-ENG-36, through an LDRD ER exploratory research grant (#20030162). The authors would like to thank the reviewers for their helpful comments.

References

1. J. Shi, J. Malik.: Normalized cuts and image segmentation, IEEE Trans. PAMI, 2000.
2. S. Mahamud, L. R. Williams, K. K. Thornber, K. Xu.: Segmentation of Multiple Salient Closed Contours from Real Images, IEEE Trans. PAMI 25(4), 2003

3. H. Blum.: A Transformation for Extracting New Descriptors of Shape. Symp. Models for Speech and Visual Form Weiant Whaten-Dunn (Ed) MIT Press. 1967
4. J. E. Goodman, J. O'Rourke. (Eds.): Handbook of Discrete and Computational Geometry, CRC Press, 1997
5. L. Prasad.: Morphological Analysis of Shapes, CNLS Newsletter, No 139, Los Alamos National Laboratory, July 1997
6. L. Prasad, R. L. Rao.: A Geometric Transform for Shape Feature Extraction, Proc. SPIE, vol. 4117, Vision Geometry IX 2000
7. L. Prasad, A. N. Skourikhine.: VISTA: Vectorized Image Segmentation via Trixel Agglomeration. Los Alamos Invention Disclosure 2003-064/S-100,632, June 2003
8. P. Tu, T. Saxena, R. Hartley.: Recognizing Objects Using Color-Annotated Adjacency Graphs, Lecture Notes in Computer Science, Vol. 1681, Springer, 1999
9. Q. Wu, Y. Yu.: Two-level Image Segmentation Based on Region and Edge Intergration. Proc. 7th Digital Image Computing. Sydney, Dec. 2003
10. J. Canny.: A Computational Approach to Edge Detection, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, November 1986
11. J. Shewchuck.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, ACM 1st Workshop on Appl. Comp. Geom. Philadelphia, PA,1996
12. M. Wertheimer.: Laws of Organization in Perceptual Forms, 1923 trans. in: A sourcebook of Gestalt psychology, W. Ellis, 1938
13. D. Mumford, J. Shah.: Optimal approximations by piecewise smooth functions and associated variational problems. Communications on Pure and Applied Mathematics, 42:577-684, 1989

Graph Transformation in Document Image Analysis: Approaches and Challenges

Dorothea Blostein

School of Computing,
Queen's University, Kingston, Ontario, Canada K7L 3N6
blostein@cs.queensu.ca

Abstract. Graphs and graph transformation are versatile tools for representing and interpreting the contents of document images. Three main components are involved: a graph representing the contents of a document image at some level of interpretation, a set of graph transformation rules (graph productions), and a mechanism for controlling the application of the graph productions. We review existing document analysis systems that use graph transformation, and discuss challenges and research opportunities in this area.

1 Introduction

Graph transformation has been used in a variety of document image analysis applications. In these applications, a document image is processed to produce an *initial graph*, representing the image primitives and the relations among the primitives. Nodes and edges can have *labels* (also called *types*). Auxiliary information is stored as *attributes* associated with nodes or edges. The distinction between a label and an attribute is as follows. Labels are part of the structure of the graph: labels must match in order for graphs to be considered isomorphic. In contrast, attributes contain information which is not part of the graph structure: attribute values are ignored during isomorphism testing. This is illustrated by an example. Bunke [Bunk82] converts a circuit diagram to an initial graph by representing the lines in the circuit diagram: each graph node represents a line endpoint, corner, or intersection, and node attributes record the (x, y) image coordinates of this feature. The node label is the degree of the node (1 for an endpoint, 2 for a corner, 3 for a T intersection). Due to this definition of node labels, subgraph isomorphism always matches an endpoint node to another endpoint node, never to a corner node or intersection node. However, the (x, y) image coordinates are ignored during the subgraph isomorphism test, because these coordinates are stored as attributes.

Graph productions (also called *graph transformation rules*) are used to inspect and update a host graph. A graph production specifies how to update a host graph by replacing one subgraph by another. Many different notations and terminology have been used for graph productions. The following exposition uses a minimum of notation in order to increase readability. More formal presentations, using set theory and category theory to model the semantics of graph replacement, are described in [Hand97] and [ICGT]. A graph production specifies: (1) conditions for production

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 23 – 34, 2005.

application, including conditions on the structure of the subgraph that will be replaced, (2) the form of the replacement subgraph, (3) the method of computing attribute values for the replacement subgraph, and (4) the method of attaching the replacement subgraph to the rest of the host graph.

Two main approaches have been taken in applying graph transformation to document image analysis [Blos98].

- **The Parsing Approach**
  Parse the initial graph according to a graph grammar. The interpretation of the document is provided directly by the parse tree, or is created as a byproduct of parse-tree construction. The Parsing Approach is discussed in Section 2.
- **The Transform Approach**
  Transform the initial graph into a final graph by application of graph productions. The final graph represents the interpretation of the document image. The Transform Approach is discussed in Section 3.

A lengthier informal introduction to graph transformation is provided in [BlSc99]. Thorough treatments of graph transformation are provided by [Hand97] and [ICGT].

We cannot expect simple solutions in using graph transformation (or any other approach) for document recognition. Document recognition is complicated, involving large amounts of expertise in various areas such as image processing, segmentation, symbol recognition, document layout, document syntax and document semantics. Graph transformation is an appealing approach that has been successfully used in small-scale systems for document image analysis. It remains to be seen whether these approaches scale up. Section 4 discusses challenges in the use of graph transformation.

2 The Parsing Approach

A graph grammar consists of a start graph and a set of graph productions. The graph grammar defines a language of graphs, consisting of all graphs that can be generated by applying sequences of productions to the start graph. For document analysis purposes, graph grammars provide a convenient way of enumerating constructs that can occur in a document (Section 2.1), or they can be used to analyze document syntax (Section 2.2) or delineate regions containing repetitive patterns (Section 2.3).

2.1 Graph Grammar to Enumerate Document Constructs

Dori et al. use a graph grammar to describe the syntax of dimension sets in engineering drawings [DoPn88] [Dori89]. An engineering drawing depicts the structure of a physical object, often using several orthogonal views. The dimension sets in an engineering drawing define the distance or angle between two points or lines on an object. Dimension sets can be drawn in many ways. The graphic components of a dimension set include text, shape, tail, contour, arrowhead, and witness line. These components form the terminal symbols of the graph grammar. Line descriptors are used to provide a compact description of the following line attributes: Geometry (straight, circular, or broken), Length (long, short, or zero), Font

(continuous, dash, or dash-dot), Width (wide, narrow, or zero), and Number of Arrows (zero, one, or two). The graph grammar is small and non-recursive. Approximately 10 graph productions suffice to generate (or parse) virtually all of the dimension-sets which can occur in machine drawings. The grammar is adapted for use in the Machine Drawing Understanding System (MDUS).

2.2 Graph Grammar to Produce a Parse Tree

Lavirotte et al. use a graph grammar to efficiently parse mathematical notation [LaPo97] [LaPo98] [KRLP99]. The initial graph encodes the relative positions of symbols. As illustrated in Figure 1, parsing creates graph nodes that contain abstract syntax trees of the recognized formulas. The parser operates bottom-up, without backtracking. The absence of backtracking is achieved because graph productions have carefully-constructed application conditions that allow the parser to uniquely determine which graph production to apply next, in cases where two different productions might apply to the same node. This process relies on the availability of accurate point-size information, which allows local identification of subscript and superscript relations. For example, point size can be used to decide whether the configuration ^{x}i is a subscript (as in the expression $x_i y_i$) or whether the spatial relationship is a byproduct of other operators (as in the expression $a^{x}i$). As a result, the graph grammar uses simpler, more efficient interpretation strategies than are used in Grbavec's approach [GrBl95] discussed in Section 3. Lavirotte's work inspired the graph grammar used in the math recognition system by Smithies et al. [SmNA01].

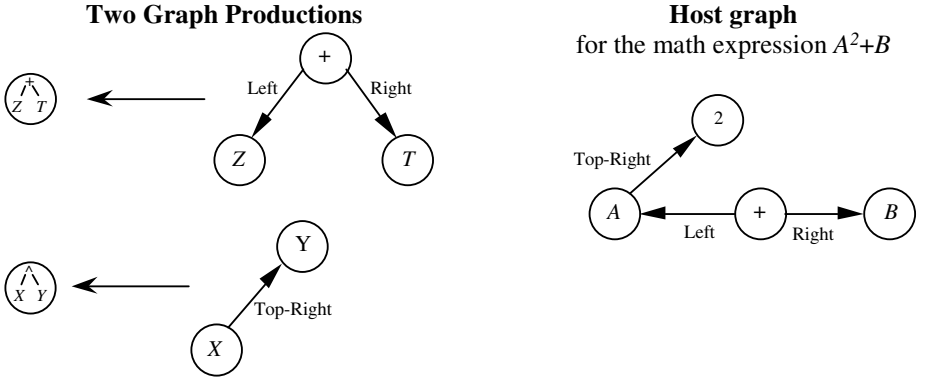


Fig. 1. These two graph productions describe addition and exponentiation, used for bottom-up parsing of 2D mathematical notation [LaPo97]. Both productions apply to the host graph shown on the right. To allow the parser to handle such situations without backtracking, productions have application conditions that identify the proper production to use in each case. These application conditions are defined during grammar construction, by analyzing all possible rule superpositions

Another application of document parsing is Amano and Asada's use of a graph grammar to analyze table-form documents [AmAs02] [AmAs03]. A table-form is a fully-ruled form that contains a mixture of entry boxes (blank or partially-blank boxes which are to be filled with information) and description boxes containing text such as "Name" or "Total" to indicate what information should go into the entry boxes. The graph grammar produces a parse tree that describes the structure of the table-form, relating description boxes to the corresponding entry boxes. In the initial graph, each box in the table form is represented by a node, and edges connect adjacent boxes. Two-part edge labels are used, with one part encoding the direction (left, right, up, or down), and the other part encoding relative size of the boxes. In the given example, 56 meta-rules are used to generate 6745 graph productions; the large number of rules arises from the various possible combinations of edge labels and box relations.

2.3 Graph Grammar to Recognize Tessellated Image Regions

Sánchez et al. use a graph grammar to recognize textured symbols, such as hatched or patterned regions, in a technical drawing [SáLl01] [SáLT02]. Given a starting point in the image, the result of parsing can be failure (the texture does not occur here), or success, with delineation of the maximal patterned region that obeys the grammar. Recursive grammars are used, with numerous productions to describe the repetitive nature of the texture, the configurations that can occur at the border where the textured image region ends, and the configurations that can occur due to drawing imperfections.

2.4 Infer a Graph Grammar to Verify Diagram Structure

Sanfeliu and Fu define tree-graphs, tree-graph grammars, an parsers [SaFu83]. A tree-graph is a type of hierarchical graph (Section 4.1.2). The use of tree-graph grammars is illustrated by the verification of the correct structure of circuit diagrams: the circuit is represented as a tree-graph, and a grammar is inferred for it.

3 The Transform Approach

In the Transform Approach, an input graph is transformed into an output graph, by applying productions. The ordering of productions is given by a control specification, which uses sequential, conditional, and looping constructs. No parser is required, because the control specification indicates how to transform the initial graph into the final graph. The term *programmed grammar* is also applied to this approach.

Bunke introduced the Transform Approach, and illustrated its use in analyzing circuit diagrams and flow charts [Bunk82]. The input graph contains edges which represent line segments in the document image, and nodes which represent intersections and endpoints. The output graph contains nodes which represent compound symbols such as *transistor*, *capacitor* and *resistor*. Computation begins with graph productions which reduce noise in the input graph. For example, one graph production closes small gaps in lines: it searches for two line-endpoint nodes that are separated by a small gap (using tests of the (x,y) node attributes), and closes the gap by removing the two line-endpont nodes. After the noise-reduction

productions have been applied, other graph productions look for configurations of nodes which represent compound symbols, and replace these by a single node.

Several authors have analyzed music notation using the Transform Approach. Fahmy et al. use a discrete initial graph [FaBl93], containing one node for each symbol in the image. Graph productions are applied in three phases. The *Build* phase adds edges that represent potentially interesting associations. The *Weed* phase uses context to identify and remove edges that represent uninteresting or conflicting associations. The *Incorporate* phase constructs the image interpretation, and deletes nodes and edges that are no longer needed. This *Build, Weed, Incorporate* organization is used repeatedly to produce the final graph, which specifies the pitch and duration of every note. Interactions among distant symbols must be analyzed, as when the pitch of a note is affected by an accidental earlier in the measure, or by an accidental in the key signature. This work assumes that correctly-recognized symbols are presented as input. Later work by the same authors introduces a generalized form of discrete relaxation, implemented via graph transformation [FaBl98]. Here, it is assumed that the symbol recognizer produces a set of possible identities for each symbol (possibly including *noise*). The symbol hypotheses are converted to an initial graph, with *exclusion* edges to indicate alternative interpretations. Contextual information, and constraints of music notation, are used to reduce the symbol-recognition ambiguity and to construct an interpretation of the music notation. In the standard formulation of discrete relaxation, all of the constraints are known *a priori*. Here, in contrast, some of the constraints can be formulated only after partial recognition results are known.

Baumann and Pies describe further work on music notation [Baum95] [Pies94]. They use a simplified type of graph production, to achieve high efficiency. Graph productions are restricted to having at most three nodes on either the Left Hand Side or the Right Hand Side, and edges are treated as node attributes, not as separate objects. Seven types of productions are defined, to increase the variety of operations that can be expressed using this restricted form of graph production. The production types, which are closely tailored to the application, are CD_START (start a control specification), LOCALIZE (apply productions to a subpart of the graph, where the subpart is defined relative to a tree structure imposed on the graph), DELOCALIZE (undo the effect of a previous LOCALIZE), REPLACE (replace the instance of LHS by the first node in RHS), GENERATE (keep the LHS instance and add the first node in RHS subject to various syntax conditions), ASSIGN (do not create new nodes, but possibly delete nodes), and REASSIGN (adjust the tree structure that is imposed on the graph). A complete system was implemented for a subset of music notation. First, an image is processed to remove staff lines. Next, each connected component is processed by a nearest-neighbour classifier, producing up to three hypotheses for which symbol is represented. This provides the input to the graph transformation. Approximately 110 productions (not the complete set) are shown in [Pies94].

Grbavec et al. use graph transformation to analyze handwritten mathematical notation, which may contain poorly-aligned symbols [GrBl95]. A second implementation of this system, using the PROGRES graph-transformation language, is described in [BlSc99]. Graph productions are applied in recognition phases. Productions in the *Build* phase create edges between any nodes that may share a significant spatial relationship. Productions in the *Constrain* phase examine graph

context in order to remove unneeded spatial relationships; this is the trickiest part of the computation, due to the possibility of mis-aligned symbols. Once the *Constrain* phase has identified the important spatial relationships, subsequent phases rather easily extract the structure of the mathematical expression.

Rahgozar et al. use graph transformation to recognize the geometry and logical structure of tables [RaCo96]. The control structure provides a horizontal bias: a preference to apply horizontal rules rather than vertical rules reflects the empirical knowledge that most tables have a horizontal reading order. No backtracking is used. The authors state that an advantage of the graph-transformation approach is the ability to trade off generality and efficiency, due to the separation of entity recognition and graph transformation. This table recognizer became part of a commercial product.

4 Challenges

Graph transformation is a powerful and versatile mechanism, but one that can be confusing and complex to use in a large-scale application. In order to keep the system intellectually manageable, it is necessary to have a clear structure to the host graph (Section 4.1), as well as a clear approach to structuring and organizing the graph productions (Section 4.2). Since most document recognition applications involve noise or ambiguity, it is also necessary to have a means to represent and process alternative interpretations (Section 4.3). Related discussion appears in [BlFG96].

4.1 Structuring the Host Graph

4.1.1 Graph Schemas, and a Class Hierarchy of Node Labels

A graph schema is a type declaration for a graph, defining node and edge labels, node and edge attributes, and connectivity restrictions. Attributes can be any data type, such as *integer*, *string*, or *graph*. Examples of connectivity restrictions are "nodes labeled X have a maximum degree k", or "edges labeled Y may only connect to nodes labeled Z". A compile-time or runtime error can be generated if application of a graph production produces a graph violating these constraints. This is helpful for debugging the graph schema and graph productions.

The graph schema may define a class hierarchy for node and edge labels. For example, the graph schema in a math-recognition application (Figure 8 in [BlSc99]) defines that LowercaseLetter, AscendingLetter, and DescendingLetter have the following ancestors in the specialization hierarchy: LETTER, TERMINAL_OPERAND, OPERAND, OBJECT. Thus, when the graph schema declares an attribute for a TERMINAL_OPERAND, this attribute is inherited by LowercaseLetter as well.

The label hierarchy plays an important role in the definition of graph productions, allowing nodes to be matched with greater or lesser specificity. For example, a graph production can match for a specific node label, such as LowercaseLetter or Plus, or for a more general node label, such as OPERAND or OPERATOR. An OPERAND node can match host graph nodes with labels such as Factor, Term, Base, and LowercaseLetter. Thus, the author of a production is able to choose node labels that impose the desired level of specificity on the subgraph matching.

4.1.2 Hierarchical Graphs

Hierarchical graphs provide a prominent representation of one or more hierarchies that are present among host-graph nodes. (Here, we are talking about a hierarchy, such as a "tree-backbone", in the graph itself; this is quite distinct from the label hierarchy discussed in Section 4.1.1.) In the layout of a hierarchical graph, node nesting is often used to depict the hierarchy visually; if multiple hierarchies are present, they can be distinguished through node shape, color, line type, or annotation [Hare88] [SiGJ93]. Internally, the hierarchical nature of the graph can be represented using edges with special labels, by collecting nodes into super-nodes [MaKl92], or by using labels that themselves are complex objects such as graphs [Schn93].

Only a few systems reviewed in Sections 2 and 3 make use of hierarchical graphs: tree-graphs [SaFu83] (see Section 2.4) and tree structure imposed on the host graph [Pies94] (see section 3). Graph hierarchies help address the control problems discussed in Section 4.2.2: they provide a framework for traversing the host graph, and for delineating portions of the hostgraph that need further processing.

4.1.3 (Non)Induced Subgraph Isomorphism: Brevity Versus Explicitness

Two definitions of subgraph are in common use. In *induced subgraph* matching, both presence and absence of edges must be matched. In *non-induced subgraph* matching, only the presence of edges must be matched; Skiena uses the term *vanilla subgraph* for non-induced subgraph [Skie98]. The difference between vanilla and induced subgraph matching is illustrated in Figure 2.

The choice between vanilla and induced subgraph matching has a big effect on the set of production rules needed to carry out a computation. If induced subgraph matching is used, then a large number of production rules are needed, to enumerate all possible combinations of edges involving LHS nodes. For example, to achieve the intended effect of rule (a) in Figure 2, use a set of rules that enumerates the various combinations of edges BA, BC, CB, and CA. This enumeration can be cumbersome, particularly when new edge types are added to the host graph (see Section 4.1.4). On the other hand, if vanilla subgraph matching is used, a single rule suffices to cover all situations in which the host graph contains extra edges; however, care must be taken that these extra edges aren't inadvertently lost, as in Figure 2(c). Also, edge absence (when it is desired) cannot be tested for by the LHS; instead, an explicit application condition must be used.

4.1.4 Scaling Up: Effect of New Node/Edge Labels on Existing Productions

As a document recognition system is developed, the developers need the ability to introduce representations for new kinds of objects and relations. In the case of a graph transformation system, this means adding new node and edge labels to the graph schema. Special care must be taken to make sure that existing production rules continue to function in the presence of the new edge labels. There is a risk that production rule application will fail (in the case of induced subgraph matching), or that the new edges will be inadvertently deleted (in the case of vanilla subgraph matching). This is a failure of modularity: the old production set works fine with the old graph schema, but extensions to the graph schema cannot be shielded from the old productions. Perhaps a better way to address this problem can be found.

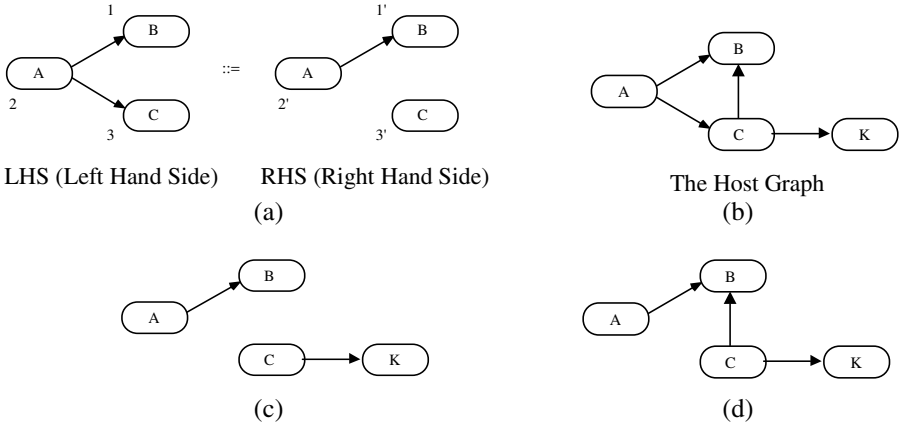


Fig. 2. Effect of induced versus vanilla (non-induced) subgraph matching. Production rule (a) is being applied to host graph (b). If induced subgraph matching is used, then the LHS of rule (a) does not match any subgraph of the host graph; therefore rule application fails and the host graph is left unchanged. In contrast, if vanilla subgraph matching is used, then a match is found: edges that are absent in LHS are allowed to be present in the host graph. Using traditional semantics for graph productions, the subgraph that matches LHS is removed from the host graph, and is replaced by a copy of RHS. (These are the LEARRE steps: Locate, establish Embedding Area, Remove, Replace, Embed [Roze87].) The result is graph (c): the loss of the AC edge is explicit in the production rule, and the loss of the CB edge arises implicitly as a side effect of production application. Some graph transformation languages reduce this problem of implicit edge deletion by allowing nodes that occurs in both LHS and RHS to be *preserved* (rather than deleted and reinserted). An edge that connects two preserved nodes is also preserved. In this example, if nodes B and C are preserved, the result is graph (d): the host-graph edge between B and C is preserved even though it is not explicitly mentioned in LHS or RHS

4.2 Organizing the Computation

Document recognition is a complex computation. In order to produce a scalable, maintainable solution, the computation must be organized in a clear, modular way. More work remains to be done in finding ways to organize large sets of graph productions. A few ideas are mentioned here.

4.2.1 Use of Passes

A graph transformation can be organized into phases or passes. Examples discussed in Section 3 include a pass for error correcting productions, followed by a pass for symbol-recognition productions [Bunk82], or Build, Weed, Incorporate passes for constructing and interpreting symbol associations [FaBl93].

While a division into passes is helpful, care must be taken in choosing effective criteria for the division into passes. In our personal experience, we found that the graph productions used in [GrBl95] were difficult to maintain. Even with the organization into Build, Constrain, Rank, Incorporate phases, it remains difficult to

understand the dependencies between productions. For example, the control diagram requires that Constrain production B1 is applied exhaustively before production B2 is applied. Is this ordering restriction really necessary, or could be removed? Answering this question involves careful thought to find a particular example of math notation that is misanalyzed if production B2 is applied before B1. Years later, we have concluded that one source of difficulty is that individual graph productions in [GrBl95] deal with a mixture of layout issues, syntax issues and semantics issues. Since then, we have experimented with a tree transformation approach that strictly separate the recognition of layout, lexical analysis, syntax, and semantics [BlCZ02].

4.2.2 Traversing Graphs; Graph Scopes

Computations on graphs require constructs for keeping track of locations within the graph, for traversing graphs, and for creating and using graph scopes. A *scope* delineates a portion of a graph; graph productions can be applied within a given scope.

Most current graph transformation languages treat the host graph as a monolithic unit, with every graph production potentially applicable anywhere in the host graph. In such an environment, one can traverse a graph by marking nodes as *visited* (using a node attribute created for this purpose). One can also keep track of a particular location in the graph by inserting a *cursor node* (e.g. [Gött92]). These are low-level representational techniques that place a significant burden on the programmer.

There is need for better language constructs to support graph traversal and scoping. Prototype implementation of such language constructs are presented in [ShBC98]. Also, as discussed in Section 3, Pies uses REASSIGN productions to (re)define a tree structure on the host graph, with LOCALIZE productions using the tree structure to define a subpart of the graph that later productions apply to [Pies94].

4.2.3 Efficiency Considerations

Graph transformation can be computationally expensive for two reasons. One reason is the time needed to apply a single production, searching the host graph to find a subgraph isomorphic to LHS. The second reason is because there might be a large number of productions under consideration; this is particularly true of the pure parsing approach, in which all productions are under consideration at any time.

The execution speed depends heavily on the characteristics of the graph productions, as well as on the control structures which invoke them. While it is true that general subgraph isomorphism is an NP-complete problem, the LHS of a typical production is very small. The small size of the subgraph allows for fairly efficient application of a graph production, particularly when node and edge labels greatly reduce the search space. Many techniques can be used to optimize subgraph-isomorphism testing [BuGT91] [Zünd96]. Also, parsers can be optimized to track changes in the set of applicable rules as productions are applied [BuGT91].

Even in cases where efficiency is critical, graph transformation can be a useful conceptual tool in the specification and design stages of a recognition system.

4.2.4 Debugging

Debugging graph transformation systems can be difficult. It is a challenge to find effective ways of displaying a large host graph, and displaying the changes made to

the host graph as productions are applied. Graph schemas (Section 4.1.1) can help catch some errors. More methods and tools are needed.

4.3 Representing Alternatives and Uncertainty

Document image analysis involves noise and ambiguity. A few graph-transformation approaches were mentioned in Section 3. One approach is to use error-correcting graph productions [Bunk82]; this works in cases where errors can be anticipated, and productions can be written to detect and correct the errors. Another approach uses a generalization of discrete relaxation to create and evaluate alternative interpretations [FaBl98].

Fuzzy graphs could be used to represent uncertainty in recognition processes. A taxonomy of fuzzy graphs [BlBP97] includes (1) a fuzzy set of crisp graphs, (2) a graph with a crisp vertex set and fuzzy edge set, (3) a graph with crisp vertices and edges, where the connectivity is fuzzy, (4) a graph with a fuzzy vertex set and crisp edge set, and (5) a crisp graph with fuzzy weights. We are not aware of any research on the transformation of fuzzy graphs in pattern recognition applications.

5 Conclusions

The document recognition systems surveyed in this paper illustrate the power and flexibility that graph transformation offers for the processing and interpretation of structured data. Section 4 discussed research issues which should be addressed in order to encourage more widespread use of graph transformation in document recognition. Graph transformation is competing against other powerful approaches, such as Hidden Markov Models [KoCh94], blackboard systems, stochastic grammars, and tree transformation. While I am a fan of graph transformation, I cannot make claims that graph transformation is currently superior to competing approaches. Managing intellectual complexity remains a problem: while it is fairly easy to understand a single graph production, it remains difficult to define a large collection of graph transformation rules to carry out a complex task such as document recognition. There is need for further research into software architectures based on graph transformation. Practitioners need clear guidance for how to organize their computations, to produce effective, maintainable, scalable, graph-transformation systems.

Acknowledgments

This paper is influenced by discussions with Hoda Fahmy, Ann Grbavec, George Nagy, Andy Schürr, and Richard Zanibbi. Financial support from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

References

[AmAs02] A. Amano, N. Asada, “Complex Table Form Analysis Using Graph Grammar,” LNCS Vol. 2423, Springer Verlag, 2002, pp. 283-286.

[AmAs03] A. Amano, N. Asada, "Graph Grammar Based Analysis System of Complex Table Form Document," *Proc. Seventh Int'l Conf. on Document Analysis and Recognition, ICDAR 2003.* Edinburgh, Scotland, Aug. 2003, pp. 916-920.

[Baum95] S. Baumann, "A Simplified Attributed Graph Grammar for High-Level Music Recognition," *Proc. Third Intl. Conf. on Document Analysis and Recognition*, Montreal, Canada, 1995, pp. 1080-1083.

[BlBP97] M. Blue, B. Bush, J. Puckett, "Applications of Fuzzy Logic to Graph Theory," Los Alamos National Lab report LA-UR-96-4792, August 1997.

[BlCZ02] D. Blostein, J. Cordy, R. Zanibbi, "Applying Compiler Techniques to Diagram Recognition," *Proc. 16th Intl. Conf. on Pattern Recognition*, Quebec City, Canada, August 2002, Vol. III, pp. 123-126.

[BlFG96] D. Blostein, H. Fahmy, A. Grbavec, "Issues in the Practical Use of Graph Rewriting," LNCS Vol. 1073, Springer Verlag, 1996, pp. 38–55.

[Blos98] D. Blostein, "Application of Graph Rewriting to Document Image Analysis," *Proc. Theory and Application of Graph Transformations – TAGT'98*, Paderborn, Germany, Nov. 1998, pp. 16-23.

[BlSc99] D. Blostein, A. Schürr, "Computing with Graphs and Graph Transformation," *Software – Practice and Experience*, Vol. 29, No. 3, 1999, pp. 197-217.

[BuGT91] H. Bunke, T. Glauser, T. Tran, "An Efficient Implementation of Graph Grammars Based on the RETE Matching Algorithm," LCNS Vol. 532, pp. 174–189.

[Bunk82] H. Bunke, "Attributed Programmed Graph Grammars and Their Application to Schematic Diagram Interpretation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 4, No. 6, Nov. 1982, pp. 574-582.

[DoPn88] D. Dori, A. Pnueli, "The Grammar of Dimensions in Machine Drawings," *Computer Vision, Graphics and Image Processing,* Vol. 42, pp. 1-18, 1988.

[Dori89] D. Dori, "A Syntactic/Geometric Approach to Recognition of Dimensions in Engineering Drawings," *Computer Vision, Graphics, and Image Processing,* Vol. 47, pp. 271–291, 1989.

[FaBl93] H. Fahmy, D. Blostein, "A Graph Grammar Programming Style for Recognition of Music Notation," *Machine Vision and Applications*, Vol. 6, No. 2, 1993, pp. 83-99.

[FaBl98] H. Fahmy, D. Blostein, "A Graph-Rewriting Paradigm for Discrete Relaxation: Application to Sheet-Music Recognition," *Intl. Journal of Pattern Recognition and Artificial Intelligence*, Vol. 12, No. 6, Sept. 1998, pp. 763-799.

[Gött92] H. Göttler, "Diagram Editors = Graphs + Attributes + Graph Grammars," *Intl. Journal of Man-Machine Studies*, Vol. 37, No. 4, Oct. 1992, pp. 481-502.

[GrBl95] A. Grbavec, D. Blostein, "Mathematics Recognition Using Graph Rewriting," *Proc. Third Intl. Conference on Document Analysis and Recognition,* Montreal, Canada, August 1995, pp. 417-421.

[Hand97] *Handbook of Graph Grammars and Computing by Graph Transformation*: *Volume 1, Foundations*, Ed. G. Rozenberg; *Volume 2, Applications, Languages, and Tools*, Eds. H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg; *Volume 3, Concurrency, Parallelism, and Distribution*, Eds. H. Ehrig, H.-J. Kreowski, U. Montanari, G. Rozenberg. World Scientific, 1997 and 1999.

[Hare88] D. Harel, "On Visual Formalisms," *Communications of the ACM*, Vol. 31, No. 5, May 1988, pp. 514–530.

[ICGT] *Int'l. Confs. on Graph Transformation*: Rome 2004, Barcelona 2002. *Int'l Workshops on Theory and Application of Graph Transformation*: LNCS Vols. 73, 153, 291, 532, 1073, 1764, published in 1979, 1983, 1987, 1991, 1996, 2000.

[KoCh94] G. Kopec and P. Chou, "Document Image Decoding Using Markov Source Models," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 16, No. 6, June 1994, pp. 602–617.

[KRLP99] A. Kosmala, G. Rigoll, S. Lavirotte, L. Pottier, "Online Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars," *Proc. Fifth Int'l Conf. on Document Analysis and Recognition*, Bangalore India, Sept. 1999, pp. 107-110.

[LaPo97] S. Lavirotte, L. Pottier, "Optical Formula Recognition," *Fourth Intl. Conf. on Document Analysis and Recognition*, Ulm, Germany, August, 1997, pp. 357-361.

[LaPo98] S. Lavirotte, L. Pottier, "Mathematical Formula Recognition using Graph Grammar," *Document Recognition V*, SPIE Vol. 3305, 1998, pp. 44-52.

[MaKl92] J. Mauss, C. Klauck, "A Heuristic Driven Parser Based on Graph Grammars for Feature Recognition in CIM," *Advances in Structural and Syntactic Pattern Recognition*, Ed. H. Bunke, World Scientific, 1992, pp. 611–620.

[Pies94] A. Pies, *Repräsentation und Verarbeitung von Musikalischem Wissen – Eine Attributierte Programmierte Graph-Grammatik zur Erkennung Gedruckter Partituren,* Diplomarbeit, DFKI Kaiserslautern, Fachbereich Informatik, Aug. 1994.

[RaCo96] M. A. Rahgozar, R. Cooperman, "A Graph-based Table Recognition System," *Document Recognition III*, *SPIE Proceedings Series, Volume 2660*, San Jose, California, Jan. 1996 pp. 192–203.

[RoPa95] J. Rocha, T. Pavlidis, "Character Recognition without Segmentation," *IEEE PAMI*, Vol. 17, No. 9, Sept. 1995, pp. 903-909.

[Roze87] G. Rozenberg, "An Introduction to the NLC Way of Rewriting Graphs," LNCS Vol. 291, Springer Verlag, 1987, pp. 55–70.

[SaFu83] A. Sanfeliu, K. S. Fu, "Tree-graph Grammars for Pattern Recognition," LNCS Vol. 153, Springer Verlag, 1983, pp. 349-368.

[SáLl01] G. Sánchez, J. Lladós, "A Graph Grammar to Recognize Textured Symbols," *Proc. Sixth Int'l Conf. on Document Analysis and Recognition,* Seattle, Washington, Sept. 2001, IEEE Computer Society Press, pp. 465-469.

[SáLT02] G. Sánchez, J. Lladós, K. Tombre, "An Error-Correction Graph Grammar to Recognize Textured Symbols," LNCS Vol. 2390, Springer, 2002, pp. 128-138.

[Schn93] H. Schneider, "On Categorical Graph Grammars Integrating Structural Transformations and Operations on Labels," *Theoretical Computer Science*, Vol. 109, 1993, pp. 257–275.

[ShBC98] M. Shukla-Sarkar, D. Blostein, J. Cordy, "GXL – A Graph Transformation Language with Scoping and Graph Parameters," *Proc. Theory and Application of Graph Transformations – TAGT'98*, Paderborn, Germany, Nov. 1998, pp. 65-71.

[SiGJ93] G. Sindre, B. Gulla, H. Jokstad, "Onion Graphs: Aesthetics and Layout," *Proc. 1993 IEEE Symposium on Visual Languages*, Bergen, Norway, 1993, pp. 287–291.

[Skie98] S. Skiena, *The Algorithm Design Manual*, Springer Telos, 1998.

[SmNA01] S. Smithies, K. Novins, J. Arvo, "Equation Entry and Editing via Handwriting and Gesture Recognition," *Behaviour & Information Technology*, Vol. 20, No. 1, 2001, pp. 53–67.

[Zünd96] A. Zündorf, "Graph Pattern Matching in PROGRES," LNCS Vol. 1073, Springer Verlag, 1996, pp. 454–468.

Graphical Knowledge Management in Graphics Recognition Systems

Mathieu Delalandre[1], Eric Trupin[1], Jacques Labiche[1], and Jean-Marc Ogier[2,♣]

[1] PSI Laboratory, University of Rouen, 76 821 Mont Saint Aignan, France
{first name, last name}@univ-rouen.fr
[2] L3i Laboratory, University of La Rochelle, 17042 La Rochelle, France
jean-marc.ogier@univ-lr.fr

Abstract. This paper deals with the problem of graphical knowledge management (formalization, modelling, representation and operationalization) in graphics recognition systems. We present here a "generic" formalism for graphical knowledge, allowing various modellings for a given graphical shape. We use a modelling library based on this formalism for the management of our graphical knowledge. The use of this library allows to request graphical knowledge databases, according to the processings' requirements on graphical primitives. Like this, this approach allows interoperability between processings, especially for their combination. We present a "short" system use-case of our approach to illustrate the interoperability between processings.

1 Introduction

This paper deals with the problem of graphical knowledge management in graphics recognition systems. This knowledge corresponds to graphical primitives used by systems during the recognition process. We present here a "generic" formalism for graphical knowledge. Indeed, this formalism allows various modellings of a given graphical shape. Based on this formalism, we have developed a modelling library for the representation and the operationalization of our graphical knowledge. We use this library in graphics recognition systems to request graphical knowledge databases, according to the processings' requirements on graphical primitives. Like this, this approach allows the interoperability between processings, especially for their combination. In the paper's follow-up, we present in section (2) an overview on graphical knowledge management in graphics recognition systems. In section (3), we present our approach for graphical knowledge management with our formalism, its representation and operationalization through our modelling library. In section (4), we present a "short" use-case of our approach with a graphics recognition system and its application. Finally, in section (5) we conclude and give some perspectives.

♣ The authors wish to thank Sébastien Perin and Mustapha Hamidou (Rouen University, France) for their contribution to this work.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 35 – 44, 2005.

2 Graphical Knowledge Management: Overview

Graphics recognition [19] is a stage of document image interpretation that is used for different purposes like: technical document interpretation [1], symbol recognition [10], handwriting recognition (especially Asian handwriting [18]), and so on. It is a well-known problem and several commercial applications exist [1]. A graphics recognition process can be decomposed into two parts [14]: the extraction part of graphical primitives, and the system part.

The system part uses various approaches in order to supervise the extraction process [4]. These approaches come from pattern recognition and artificial intelligence domains. This paper deals especially with the extraction step of graphical primitives [22] [5]. This one extracts graphical primitives from document images corresponding to graphical shapes of documents. It employs many methods in order to extract different primitive types from images. In a previous work [5], we have proposed a classification of these methods in some families (Fig. 1). The methods are based on skeletonization (a), contouring (b), tracking (c), run (d), region (e), mesh (f), object segmentation (g), connected component grouping (h).

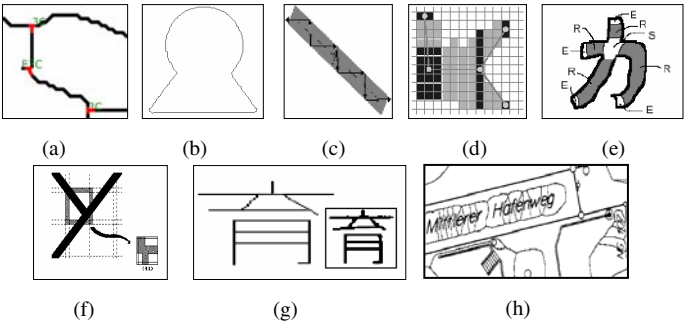


Fig. 1. (a) skeletonization (b) contouring (c) tracking (d) run(e) region (f) mesh (g) object segmentation (h) component grouping

We do not discuss in this paper about the presentation and the comparison of these methods[1], but about common graphical primitives extracted between these methods as we show in Table 1. These graphical primitives can be grouped in four primitive classes: pixel, vectorial (vector, arc, and curve), region (subset of connected pixels on image), and symbol (a symbolic label). In the same way, some methods can be used to extract different types of primitive [5]. For example, the skeletonization and contouring are often used with a polygonisation method to extract vectorial data (in the "*two steps*" vectorisation systems [15]). Also, the run decomposition methods can be used to extract the skeleton and contours [24], and in this way used with polygonisation method, and so on.

[1] It is not the purpose of this paper to do this, we report the reader to [1] [5] [10] and [22].

Table 1. Comparison of methods for graphical primitive extraction

Graphical primitives	Methods
Pixel	skeletonization (a), contouring (b), run (d)
Vectorial	skeletonization (a), contouring (b), tracking (c), run (d), object segmentation (g)
Region	Run (d), region (e), component grouping (h)
Symbol	*all*

So, a system can use different methods in order to extract some given graphical primitives. In a previous work [5], we have presented the drawbacks and advantages of all these methods (Fig. 1). So, their combination can help a system for the graphical primitive extraction. From our point of view, it is an important research perspective of graphics recognition. However, this perspective raises the problem of *graphical knowledge exchange* between the system's processings based on these methods.

In computer science systems "in the large" [20], the *knowledge* corresponds to semantic data (example, data: "37.5", semantic: "a temperature") with their exploitation processes (the system parts based on knowledge use). The use of these exploitation processes corresponds to the *operationalization* of systems' knowledge [4]. In these systems, the knowledge is used [20] in an internal way (in the algorithms) or in external way (outside of algorithms). The external knowledge is based on knowledge *representation* methods [16] like: representation languages, databases, and formats. In the internal and external cases, the knowledge used is based on a *formalism* [16]. Several formalisms exist[2] like: algebraic (list, matrix, number, and so on.), rule, graph, frame, and so on. Based on these formalisms, the systems use *modellings* of their knowledge. A modelling corresponds to a possible use of a given formalism. In the literature [16], we talk about knowledge *management* for the formalization, the modelling, the representation, and the operationalization of knowledge.

Different types of knowledge are used in graphics recognition systems [14]. This paper deals only with graphical knowledge [11]. This knowledge corresponds to graphical primitives used in systems. We resume on Table 2 formalisms commonly used for the graphical knowledge in some research systems, and standard formats of vector graphics.

Table 2. Formalisms of graphical knowledge

Systems	Formalisms	Formats	Formalisms
ADIK [12]	vectorial, rule, symbol	CGM [8]	vectorial, graph
ANON [1]	vectorial, rule, symbol	DXF [2]	vectorial, list
DMOS [3]	region, vectorial, rule	SVG [21]	vectorial, list
OOPSV [15]	vectorial, graph, symbol		
QGAR [9]	vectorial, graph		

[2] We don't present here these formalisms, and report the reader to [16] and [20].

From our point of view, the graphical knowledge in graphics recognition systems is based on two formalism levels (Table 2). A low level is used to describe the graphical primitives. It is based on general formalisms used in graphic file formats [11]: vectorial, and raster (for the region representation). A high level is used to structure these graphical primitives. Different formalisms are then used. Among them the most used are the lists and the graphs [17], the rule formalism is often used too. However this one is more adapted to recognition problem than modelling problem [1] [3] [12]. The graph and list formalisms correspond to structural descriptions of graphical primitives. Indeed, the graphical shapes of documents represent themselves, in a natural way, according to a structural description [1] [10] [19].

However, based on these structural formalisms, the graphics recognition systems use fixed modellings of their knowledge [3] [9] [12] [15]. The modellings are chosen according to the recognition approaches of these systems. Based on these fixed modelling, these systems can't deal with an adaptable combination of processings for the graphical primitive extraction. To solve "a part" of this problem, some systems perform their combinations through a low level formalism (image) [15], or a high level formalism (rule) [3].

In the following section (3), we present a "generic" formalism and a modelling library for the graphical knowledge management. This library allows to request a graphical knowledge databases, according to the processings' requirements on graphical primitives. Like this, this approach allows the interoperability between processings, especially for their combination.

3 Our Approach for Graphical Knowledge Management

We present here our approach for graphical knowledge management. We first present in subsection (3.1) our formalism. Next, in subsection (3.2), w e present a modelling use-case of a given graphical shape. In subsection (3.3), we present the knowledge representation and operationalization through our modelling library.

3.1 Used Formalism

Our formalism is based on object-oriented concepts for knowledge formalization [13]. We have based our approach especially on works described in [23]. Our graphical knowledge (k_g) is represented (1) by a single graphical object (o). This graphical object is an instance (i) of a generic (and abstract) graphical object class (o_g) which is specialized in several graphical object classes ($\{1,.,u\}$) according to an inheritance (I) relationship. In this way, this representation exploits some important properties of inheritance [23], polymorphism and extensibility. The graphical objects are composed (2) of a set of data (D) and methods (M). These data (D) can be composed (2) of specific data (d_i), or other graphical objects (o_i) through a composition (or aggregation) relationship.

In our approach (as we have concluded in our overview of section (2)) we have decomposed the different graphical object classes, in an implicit way, into two formalism levels (3). The first one is a low level formalism for the description of graphical primitive (p). So, this description is based on vectorial and raster formal-

isms [11]. We have considered some standard graphical primitives like: *point*, *junction*, *line*, *arc*, *curve*, *region*, and *quadrilateral*. However, these standard graphical primitives can be easily extended thanks to the polymorphism and extensibility properties [23] of our approach (1). The second one is a high level formalism, based on list (*l*) and graph (*g*) formalisms, to structure the graphical objects corresponding to graphical primitives.

$$\exists k_g = \{o\} \quad o \xrightarrow{i} o_g \xleftarrow{I} \{o_{g_1},..,o_{g_u}\} \tag{1}$$

$$o = \{D,M\} \quad D = \{\{d_0,..,d_u\},\{o_0,..,o_v\}\} \quad M = \{P,\{r,w\}\} \tag{2}$$

$$o_{g_i} \in \{\{p_1,..,p_u\},\{l,g\}\} \tag{3}$$

The list formalism (*l*) defines (4) sets of ordered graphical object (*O*) and ordered attribute object (*A*). A given attribute object (a_i) describes a relationship (5) between two successive graphical objects (o_i) and (o_j) of the list. These attribute objects (*a*) are defined in the same way (6) (7) than the graphical objects (*o*) (1) (2). In the same way, these ones exploit the polymorphism and extensibility properties [23]. According to the list's looping (4), the (*A*) size may be of (*u*) or (*u-1*). We have considered some standard attributes like the *labelling*, *angle*, *length*, and so on.

$$l = \{O,A\} = \{\{o_0,..,o_u\},\{a_0,..,a_v\}\} \quad v = ((u-1) \vee u) \tag{4}$$

$$\forall\, a_i \quad \exists \{o_i,o_j\} \quad f(o_i,o_j) = a_i \tag{5}$$

$$a \xrightarrow{i} a_g \xleftarrow{I} \{a_{g_1},..,a_{g_u}\} \tag{6}$$

$$a = \{D,M\} \quad D = \{\{d_0,..,d_u\},\{a_0,..,a_v\}\} \quad M = \{P,\{r,w\}\} \tag{7}$$

The graph formalism (*g*) defines (8) sets of graphical object (*O*) and edge object (*E*). A given edge object (e_q) describes a directed (or undirected) relationship (9) between any graphical objects (o_i) and (o_j). This relationship is defined (9) according to a given attribute object (a_q).

$$g = \{O,E\} = \{\{o_0,..,o_u\},\{e_0,..,e_v\}\} \tag{8}$$

$$\forall\, e_q = \{\{i,j\},a_q\} \quad \exists \{o_i,o_j\} \quad f(o_i,o_j) = a_q \tag{9}$$

Through these object-oriented concepts for knowledge formalization, our graphical knowledge (k_g) (1) is structured according to a *hierarchical relational graph* [17]. Indeed, our graphical objects (*o*) (1) included into the list (*l*) (4) and graph (*g*) (8) objects can be other list (*l*) and graph (*g*) objects. The list (*l*) objects are used here to reduce the size of graph (*g*) objects. Indeed, the list formalism can be considered as graph formalism [17], this one is commonly used in graph representation models of document shapes [24].

3.2 Modelling Use-Case

Our formalism (subsection (3.1)) allows various modellings of a given graphical shape. In order to illustrate this "generic" aspect, we present here a modelling use-case of linked-squares (Fig. 2 (a)).

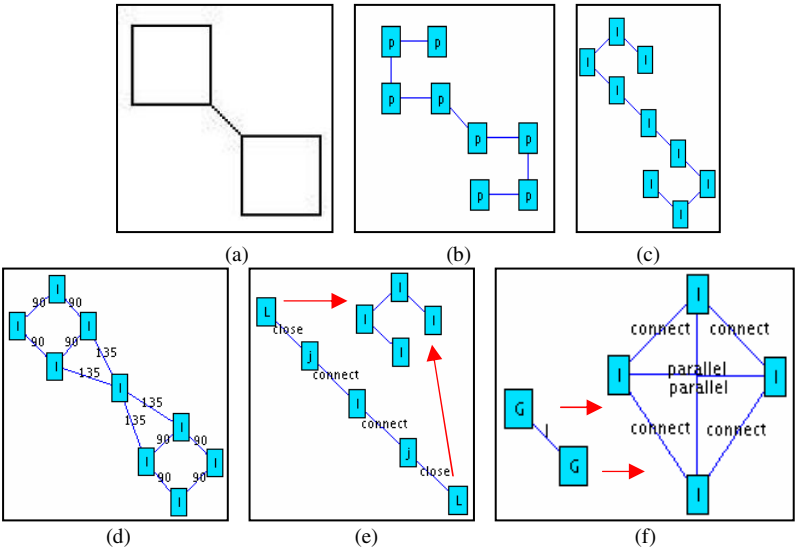


Fig. 2. (a) linked-squares (b) point list (c) line list (d) line graph (e) hierarchical lists (f) hierarchical graphs

The Fig. 2 (b), (c), and (d) present three no-hierarchical modellings of linked-squares. The (b) (c) modellings are based on the list formalism, for the point (*p*) (b) and line (*l*) (c) objects. Indeed, it is possible to describe the linked-squares according to successive points or lines. The (d) modelling is based on graph formalism for the line (*l*) objects. In this graph, the connected lines (*l*) are linked by angular attributes.

The Fig. 2 (e) and (f) present two hierarchical modellings of linked-squares. The (e) modelling is based on list formalism for the line (*l*), junction (*j*), and (*L*) objects. In this modelling, the (*L*) objects represent sub-lists of line (*l*) object. Each sub-list corresponds to a square object. The (*L*), (*j*), and (*l*) objects are linked by the labelling attributes (*close*) and (*connect*). The (f) modelling is based on graph formalism for the line (*l*) and (*G*) object. In this modelling, the (*G*) objects represent sub-graphs of line (*l*) objects. Each sub-graph corresponds to a square object. In these sub-graphs, the line (*l*) objects are linked by labelling attributes (*connect*) and (*parallel*). The (*l*) and (*G*) objects are linked by an attribute corresponding to a graphical primitive (*l*).

Like this, the Fig. 2 presents some of possible modellings (b-f) of linked-squares (a). Based on our formalism (subsection (3.1)), it is still possible to define several modellings. From our point of view, it doesn't exist a best modelling to describe a given

graphical shape. Indeed, an adopted modelling by a graphics recognition system depends of its process aims [14]. Then, it is important to allow the exchange of "similar" graphical knowledge between graphics recognition systems, in spite of differences between adopted modellings.

3.3 Graphical Knowledge Representation and Operationalization

We present in this subsection the representation and the operationalization of our graphical knowledge through our *Graphical Object Modelling Library*[3].

In our formalism (subsection (3.1)) each graphical and attribute object is composed of a set of method (*M*) (2) (7). This set of method is composed of process methods (*P*) on object's data, and read (*r*) write (*w*) methods (2) (7). In this way, each object supports its outsourcing. The Fig. 3 (a) gives an example of representation in XML used in our library of point object. The outsourcing properties of objects can be then used by other objects, like the (*l*) (4) and (*g*) (8) objects, or any other graphical or attribute objects using a composition relationship (2) (7). The Fig. 3 (b) gives an outsourcing example of point object used through a composition relationship into the line object.

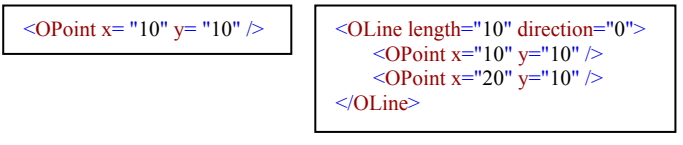


Fig. 3. XML representation: point (a) line (b)

We use our library in the graphics recognition processings. Like this, our library allows the graphical knowledge operationalization into the processings for, the graphical primitive management, and their read/write into graphical knowledge databases represented in XML (Fig. 3).

The aim of our approach is to allow the interoperability between processings. We have developed a request based approach, in order to extract graphical knowledge from XML databases according to the processings' requirements on the graphical primitives. This approach exploits request methods, based on list or/and graph search algorithms. So, these requests are "content based" like the FLoWeR[4] requests. Indeed, our requests do not allow to structure search like sub-lists or/and sub-graphs.

The Fig. 4 presents our requests based approach through an example. In this example, a processing (*Processing*) performs a read request method (R_r) on a graphical knowledge database (k_g). This request uses a set of "content constraints" corresponding to the request (r_r): list of point (l_p) and size ($s_{\geq 2}$). So, we can translate this request into natural language like this: "**for** graphical knowledge (k_g) **return** lists **where** a list

[3] GOMLib, available on http://site.voila.fr/mdhws/
[4] For Let Where Return.

is only composed of point object (l_p) **and** the list's size is upper than one point ($s_{\geq 2}$)". Then, a read graphical object (o_r) is extracted corresponding to request's result. Following the execution of processing (*Processing*), an object to write (o_w) is obtained. The processing (*Processing*) performs then a write request method (R_w) in order to update the graphical knowledge database (k_g) with this result object (o_w). During the (R_w) execution, (r_r) is used like trace to locate the objects to update, in this example the line list objects (l_l) update the point list object (l_p).

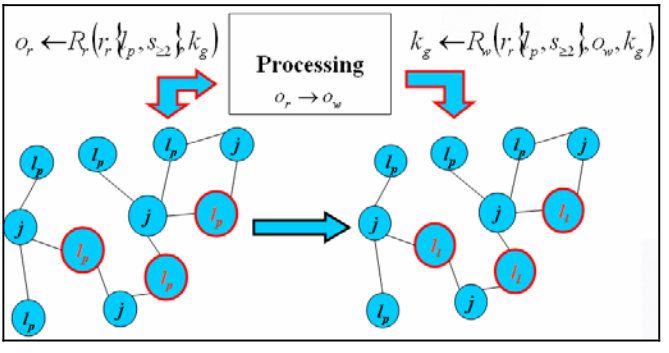


Fig. 4. example of request process on a graphical knowledge database

4 System Use-Case

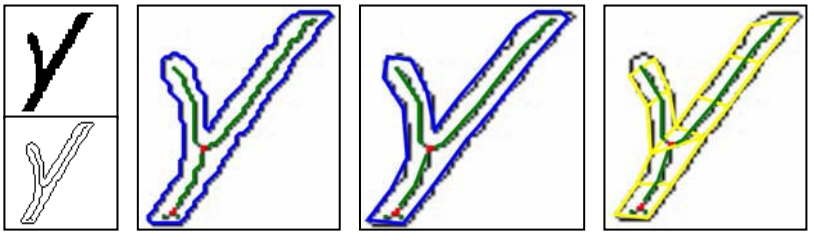
In order to illustrate our approach for graphical knowledge management (section (3)), we present here a "short" system use-case of graphics recognition. The Fig. 5 (a-high) gives a network's part extracted from an utility map [6]. For our graphics recognition system, we have developed the well-known contouring/skeletonisation approach [11]. We don't discuss here about the processing abilities[5], but about the interoperability between processings through the graphical knowledge database.

In a first step (Fig. 5), our system performs a chaining processing (b) on the resulting image of skeletonisation/contouring processing (a-low). So, our graphical knowledge database (k_g) is updated (Table 3) from a raster object (r) to a graph (g) object composed of junction (j) and point list (l_p) objects. In a second step, our system performs a polygonisation processing (Fig. 5 (c)). A request on point list (l_p) objects is then used to extract these (l_p) objects form graph (g) object. These (l_p) objects are then updated (Table 3) in (k_g) by line list (l_l) objects. In the final step, our system performs a contour matching processing (Fig. 5 (d)). A request on closed line list (l_l) objects is then used to extract these (l_l) objects form graph (g) object. These (l_l) objects are updated (Table 3) in (k_g) by quadrilateral list (l_q) objects. The result graphical knowledge database (k_g) is then composed of (g), (j), (l_l), and (l_q) objects.

[5] It is not the purpose of this paper to do this, we report the reader to [6].

Table 3. update of graphical knowledge database for processings interoperability

skeletonisation/contouring	chaining	polygonisation	matching
$k_g = \{r\}$	$k_g = \{g,j,l_p\}$	$k_g = \{g,j,l_l\}$	$k_g = \{g,j,l_l,l_q\}$



(a) (b) (c) (d)

Fig. 5. (a) skeletonisation/contouring (b) chaining (c) polygonisation (d) matching

5 Conclusion and Perspectives

In this paper we have presented an approach for graphical knowledge management in graphics recognition systems. This approach is based on a "generic" formalism allowing various modellings of a given graphical shape. This formalism is based on object-oriented concepts, especially for the inheritance, polymorphism and extensibility properties. We represent and operationalize this formalism through our modelling library. We use this library into graphics recognition systems to request graphical knowledge databases, according to the processings' requirements on graphical primitives. Like this, this approach allows the interoperability between processings, especially for their combination. For the perspectives, in a first step we wish to develop a complete platform of graphics recognition processing based on our formalism. We would like to exploit the interoperability between processings to develop some strategic approaches [6]. Next, we wish to extend our request based approach with request language to extract graphical object structures, through graph request [7].

References

1. S. Ablameyko and T. Pridmore. Machine Interpretation of Line Drawing Images. Springer Verlag Publisher, 2000.
2. Autodesk. Drawing Interchange and File Formats, Release 12. Autodesk Inc, 1992.
3. B. Coüasnon. Dmos : a generic document recognition method, application to an automatic generator of musical scores, mathematical formulae and table structures recognition systems. In International Conference on Document Analysis And Recognition (ICDAR), 2001.
4. D. Crevier and R. Lepage. Knowledge-based image understanding systems: A survey. Computer Vision and Image Understanding (CVIU), 67(2):161-185, 1997.

5. M. Delalandre, E. Trupin, and J. Ogier. Local structural analysis: A primer. Lecture Notes in Computer Sciences (LNCS), 3088:220-231, 2004.
6. M. Delalandre, Y. Saidali, J. Ogier, and E. Trupin. Adaptable vectorisation system based on strategic knowledge and xml representation use. Lecture Notes in Computer Sciences (LNCS), 3088:196-207, 2004.
7. R. Giugno and D. Shasha. Graphgrep : A fast and universal method for querying graphs. In International Conference on Pattern Recognition (ICPR), 2002.
8. L. Henderson and A. Mumford. The CGM Handbook. Academic Press, 1993.
9. X. Hilaire and K. Tombre. Improving the accuracy of skeleton-based vectorisation. In Workshop on Graphics Recognition (GREC), 2001.
10. J. Lladós, E. Valveny, G. Sánchez, and E. Martí. Symbol recognition : Current advances and perspectives. In Workshop on Graphics Recognition (GREC), 2001.
11. J. Murray and W.V. Ryper. Encyclopedia of Graphic File Formats. Editions O'Reilly, 1996.
12. B. Pasternak and B. Neumann. The role of taxonomy in drawing interpretation. In International Conference on Document Analysis And Recognition (ICDAR), 1995.
13. J. Pesonen. Concepts and object-oriented knowledge representation. Master's thesis, Department of Cognitive Science, University of Helsinki, Finland, 2002.
14. Y. Saidali, S. Adam, J. Ogier, E. Trupin, and J. Labiche. Knowledge representation and acquisition for engineering document analysis. Lecture Notes in Computer Science (LNCS), 3088:25-36, 2004.
15. J. Song, F. Su, C. Tai, and S. Cai. An object-oriented progressive-simplification based vectorisation system for engineering drawings: Model, algorithm and performance. Pattern Analysis and Machine Intelligence (PAMI), 24(8):1048-1060, 2002.
16. J. Sowa. Knowledge Representation: Logical, Philosophical, and Computational Foundations. Cole Publishing Co, 1999.
17. J. Spinrad. Efficient graph representations. In Fields Institute Monographs, volume 19. American Mathematical Society, 2003.
18. C. Suen, S. Mori, S. Kim, and C. Leung. Analysis and recognition of asian scripts - the state of the art. In International Conference on Document Analysis and Recognition (ICDAR), pages 866- 878, 2003.
19. K. Tombre and B. Lamiroy. Graphics recognition - from re-engineering to retrieval. In International Conference on Document Analysis and Recognition (ICDAR), 2003.
20. J. Ullman. Principles of Data-Base and Knowledge Base Systems, volume 1-2. Computer Sciences Press, 1989.
21. W3C. Scalar Vector Graphics (SVG) 1.0 Specification. 2001.
22. L. Wenyin and D. Dori. From raster to vectors : Extracting visual information from line drawings. Pattern Analysis and Applications (PAA), 2(2):10-21, 1999.
23. T. Williams. Object architecture dealing with the unknown - or - type safety in a dynamically extensible class library. Technical report, Microsoft Corporation, 1988.
24. H. Xue. Building skeletal graphs for structural feature extraction on handwriting images. In International Conference on Document Analysis And Recognition (ICDAR), 2001.

A Vascular Network Growth Estimation Algorithm Using Random Graphs

Sung-Hyuk Cha[1], Michael L. Gargano[1], Louis V. Quintas[2], and Eric M. Wahl[2]

[1] Computer Science Department Pace University,
New York, NY 10038 USA
{scha, mgargano, lquintas}@pace.edu
[2] The New York Institute for Bioengineering and Health Science,
New York, NY 10011 USA
ericwahlmd@aol.com

Abstract. Vascular networks develop by way of angiogenesis, a growth process that involves the biological mechanisms of vessel sprouting (budding) and splitting (intussusception). Graph theory is a branch of discrete mathematics that is excellently suited to model vascular networks and to analyze their properties (invariants). A random graph process model can simulate the development of a vascular network that has been modeled using graph theory. The renal glomerulus is one example of such a vascular network. Here the correlation between the invariants of this vascular network modeled as a graph and the mechanisms of the growth of the network are studied. It is proposed that the relative frequencies of sprouting and splitting during the growth of a given renal glomerulus can be estimated by the invariants (root distance, radius, and diameter) of the graph representing the renal glomerulus network. Experimental evidence is given to support this conjecture.

1 Introduction

Angiogenesis is the biological process of blood vessel growth and vascular network development and has been observed to have two mechanisms: *sprouting* (budding) and *splitting* (intussusception) [1,2]. Predicting how a vascular network develops or describing how a network has grown is of great interest in the study of angiogenesis. Here we consider the problem of estimating the proportion of splitting in the development of a given renal glomerular network and present a solution using a random graph process.

Graph theory methods have been used to analyze the complex network of the renal glomerulus by representing vessels as edges and the branch points of the network as vertices [3-5]. A random graph process model can simulate the growth process of a vascular network and can be applied to any vascular network that can be represented by edges and vertices.

This study will demonstrate how the generation of graphs by a random graph process can simulate the development of vascular networks. It is further shown how certain graph theory invariants may be used to estimate the relative frequencies of

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 45–53, 2005.

their growth mechanisms (i.e., sprouting and splitting) that occurred in arriving at a given vascular network. This is done as follows.

Using a random graph process that models the growth of a vascular network, a large set of random graphs is generated. The proportion of sprouting and splitting during the generation is kept as part of the associated data. A selected set of graph invariants is computed for the graphs in this set and compared with the same set of invariants of a given biological vascular network. This procedure suggests the two interesting findings:

(1) There exists a correlation between certain graph invariants and the growth mechanisms of sprouting and splitting, and
(2) The sprouting and splitting mechanisms are conjectured to have approximately occurred with equal probability in the renal glomerular networks of six normal adult rats and a higher proportion of splitting to have occurred in an adult uremic rat case.

Details are provided in the sections that follow. In Section 2 a random graph process model for angiogenesis together with the graph theoretical invariants used in this study are defined. In Section 3, the algorithm to estimate the growth pattern for glomerular networks is described. Section 4 contains the experimental data used to support our results.

The rest of the paper is organized as follows. Section 2 illustrates a statistically inferable approach to establish the individuality using the *dichotomy* model, showing the dichotomy transformation process. Section 3 presents features and distance measures used for iris authentication in the literature. Section 4 compares the experimental results of various classifiers using different features and distance measures. Finally, Section 5 draws some conclusion.

2 Graph Theory: Definitions

A *graph* G consists of a nonempty set V whose elements are called *vertices* and a second set E made up of pairs of vertices. The elements of E are called *edges* and the graph G is defined by the pair (V,E).

A graph model for a vascular network is obtained by assigning the branch points of the network to be the vertices of the graph and the vessels between branch points to be the edges of the graph.

A *random process* consists of a transformation (physical or simulated) that can be repeatedly applied to some entity and such that the transformation always produces a clearly identified outcome from some fixed known set of possible outcomes. The outcome is not known in advance but is determined by precisely specified rules of probability. In particular, a random process can be described by its set of possible outcomes, called the *states* of the process, together with their *transition probabilities*, the latter being the probabilities of moving from a given state to any other state of the process. A *step* in a random process starts by applying the underlying transformation of the process to the resulting state of the previous step and ends by obtaining the next state in the process.

Such processes are considered under the general heading of stochastic processes (see [6] p. 419) and when other specific rules are included have more specialized names, for example, such as Markov processes, and the following.

A *Random Graph Process* is a random process for which the states are graphs (see [7]). Using the two graph transformations associated with splitting and connecting as the steps in a random graph process a step by step simulation of angiogenesis is obtained. Note the initial state is a single vessel. This process produces simulated vascular networks at various stages of development that can be compared to experimentally obtained biological vascular networks. An example of one result obtained by this approach is the conjectured proportion of splitting and connecting in arriving at biological or simulated vascular network with particular values of specified invariants.

2.1 Two Mechanisms: Splitting and Connecting

In this section, we formulate two mechanisms in recursive definition forms. A biological renal glomerular network grows starting from a single vessel whose two nodes are called the afferent (A) and efferent (E) nodes. Thus, the initial state (at step 1) of a network is formulated as follows:

$$E_1 = \{(u,v)\}$$
$$V_1 = \{u,v\}$$
$$RF_1 = 0$$

A consensus of biological researchers indicates that *sprouting* and *splitting* are two important growth mechanisms. One observation of angiogenesis is that sprouting is followed quickly by the sprout connecting to a vessel. We shall call the latter the *connecting* mechanism. Thus when we refer to the two growth mechanisms we mean splitting and connecting. These two mechanisms are simulated with graph theory transformations and correspond to steps in a random graph process (see Figure 1). The process is applied to generate a large set of random graphs. Although the graphs generated are directed graphs, here we need only consider invariants of their underlying undirected graphs.

A *state* of a random graph process is described as $S_i = (V_i, E_i, RF_i)$, where i indicates the *number of steps*, V_i, E_i, and RF_i are the set of *nodes*, the set of *arcs*, and the *relative frequency of splitting* after i steps. Specifically, RF_i is the number of splits divided by the number steps.

The splitting transformation (see Figure 1(a)) is defined as follows:

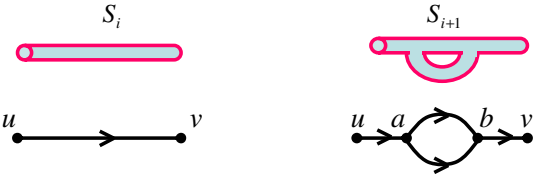
Given a state $S_i = (V_i, E_i, RF_i)$ with graph (V_i, E_i) having order n, size t, select an arc (u, v) uniformly with probability $1/t$ and introduce a split at arc (u, v) to obtain the state S_{i+1} at step $i + 1$.

$$E_{i+1} = (E_i - (u,v)) \cup \{(u,a),(a,b),(a,b),(b,v)\}, \text{ where } a,b \notin V_i, (u,v) \in E_i$$
$$V_{i+1} = V_i \cup \{a,b\}$$
$$RF_{i+1} = \frac{RF_i \times i + 1}{i+1}$$

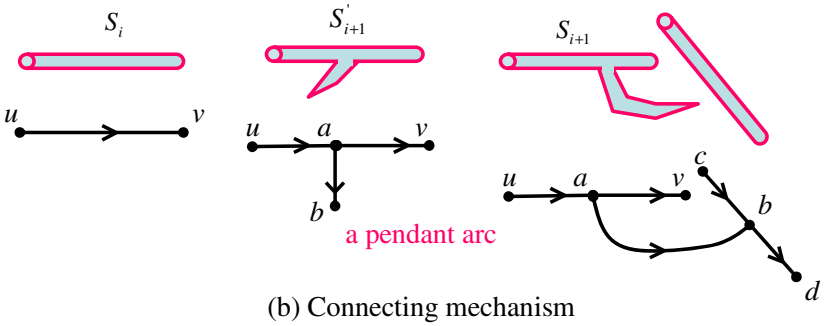
The connecting transformation (see Figure (1b)) is defined in two steps as follows: Given a state $S_i = (V_i, E_i, RF_i)$ select an arc (u, v) uniformly with probability $1/t$ and introduce a sprout at arc (u, v) to obtain the intermediate state S'_{i+1}. This sprouting

introduces a pendant arc. Now given the intermediate state S'_{i+1} having order $n + 2$, size $t + 2$, with the newly introduced pendant arc (a, b), select an arc (c, d) uniformly with probability $1/(t + 2)$ and connect the pendant node b to arc (c, d).

$$E'_{i+1} = (E_i - (u,v)) \cup \{(u,a),(a,b),(a,v)\}, \text{ where } a,b \notin V_i, (u,v) \in E_i$$
$$V'_{i+1} = V_i \cup \{a,b\}$$
$$E_{i+1} = (E'_{i+1} - (c,d)) \cup \{(c,b),(b,d)\}, \text{ where } (c,d) \in E'_{i+1}$$
$$V_{i+1} = V'_{i+1}$$
$$RF_{i+1} = \frac{RF_i \times i}{i+1}$$



(a) Splitting mechanism



(b) Connecting mechanism

Fig. 1. Two mechanisms

Remark. The connecting transformation can be partitioned into four types depending on where the target arc (c, d) is located relative to the sprouted node b.

(i) (c, d) is at least distance 2 from b.
(ii) $(c, d) = (a, v)$; connection results in parallel arcs indistinguishable from a split.
(iii) $(c, d) = (u, a)$; connection results in a 2-cycle.
(iv) $(c, d) = (a, b)$; connection results in a loop.

See Figure 2 for an illustration of the four types of connections. The figure shows a few random graphs generated starting with an arc and steps corresponding to splitting with probability ½.

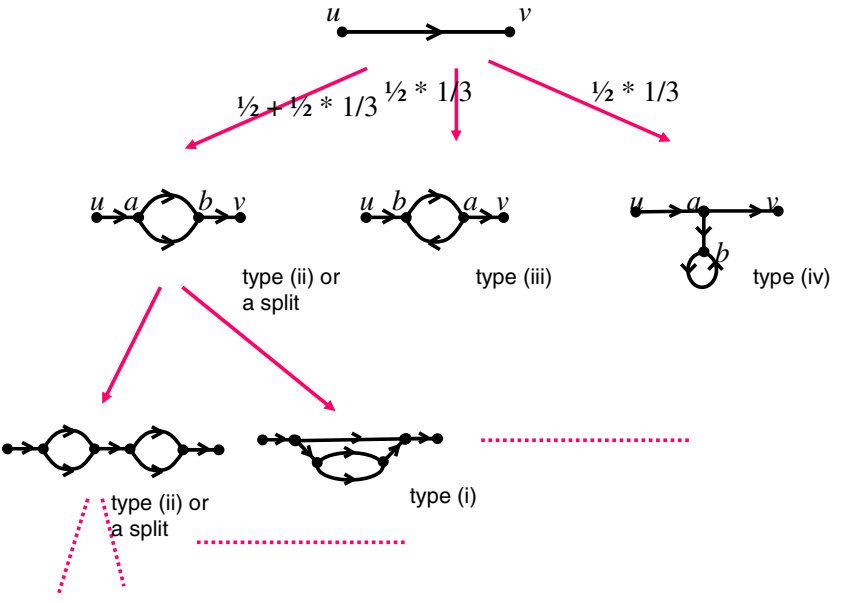


Fig. 2. Random graphs using two transformations

Fact 1. The number of nodes in a generated random graph after i steps is $|V_i| = 2i + 2$.

Fact 2. The number of arcs in a generated random graph after i steps is $|E_i| = 3i + 1$.

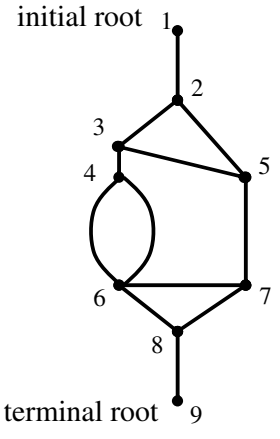
Fact 3. If the splitting relative frequency is RF_i after i steps, then the connecting relative frequency is $1 - RF_i$.

Facts 1 and 2 are easily proven by induction. Fact 3 is by definition.

2.2 Graph Invariants

We shall use three graph invariants: *radius (R), diameter (D)*, and *root distance (rd)* defined for a birooted undirected graph [9]. They will be used in a feature vector to estimate the growth process, specifically RF_i, of a biological renal glomerular microvascular network. Note that RF_i at step i is also interpreted as the probability of splitting at each step leading up to step i. The *radius* and *diameter* are the minimum and maximum eccentricities among the set of all vertices of a graph, respectively, where the *eccentricity of a vertex u* in a connected graph G is the maximum possible-distance d(u,v) from u to any other vertex v in G. In other words, the eccentricity of u is the length (number of edges) of a shortest path connecting u to a vertex furthest from it. In the distance matrix shown in Figure 3, the bottom highlighted row gives the eccentricity distribution of the vertices of the graph. The *root distance* is the length of a shortest path between a vertex designated as the initial root and a second

vertex called the terminal root. In physiological terms, for a biological vascular network, these roots correspond to what are called afferent and efferent nodes. Figure 3 gives the values of these three graph invariants for the given graph.



Undirected distance matrix

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
1	0	1	2	3	2	4	3	4	5
2	1	0	1	2	1	3	2	4	5
3	2	1	0	1	2	2	3	3	4
4	3	2	1	0	3	1	2	2	3
5	2	1	2	3	0	2	1	2	3
6	4	3	2	1	2	0	1	1	2
7	3	2	3	2	1	1	0	1	2
8	4	4	3	2	2	1	1	0	1
9	5	5	4	3	3	2	2	1	0
eccentricities	5	5	4	3	3	4	3	4	5

$n = 9$
$t = 12$
R = min (eccentricity) = 3
D = max (eccentricity) = 5
rd = root distance, d(1,9) = 5

Fig. 3. Graph Invariants for the given graph example

3 Network Growth Estimation Algorithm

We consider the problem of estimating the proportion of splitting that has produced a mature adult renal glomerular network. Our solution is via an algorithmic approach. The input to the problem is the graph representation of a biological renal glomerular network and the output is an estimation of the splitting and connecting mechanisms that occurred during its development. The proposed algorithm uses a random graph process. There are two parts to the algorithm: (a) the random graph generation and (b) the nearest neighbor classification phases.

First, the random graph generation phase is used to obtain a reference sample set of random graphs to compare with a given biological network. It takes an input of a renal glomerular network in a graph representation $G_q = (V_q, E_q)$ whose RF_q, the relative frequency of splitting is unknown. Our goal is to generate a large set of random graphs whose size is similar to the biological network. Let n be the step size to generate random graphs. In order to have similar order random graphs, we apply the following procedure. We know the order of the vertices in the biological network, that is $|V_q|$. By Fact 1, we can decide the one possible step size $n_v = \frac{|V_q| - 2}{2}$ which will generate random graphs whose order is the same as the biological one. However,

the size of these graphs, $|E_v^{'}| = 3n_v + 1$, by Fact 2, might be different from the biological one. Another possible step size is $n_e = \frac{|E_q| - 1}{3}$ which will generate random graphs whose size is the same as the biological one. Here the random graphs' order $|V_e^{'}| = 2n_e + 2$, by Fact 1, this order might be different from the biological network order. In order to minimize the difference, we choose the *step size* by the following equation.

$$n = \underset{n_i \in \{1,\cdots,|E_q|/3\}}{\arg\min} \sqrt{(2n_i + 2 - |V_q|)^2 + (3n_i + 1 - |E_q|)^2} \tag{1}$$

Once the step size, *n* is decided, a large number *m* of random graphs whose relative frequency of splitting is known are generated by the pseudo code in Figure 4. As a result, a large set of randomly generated graphs are produced and the splitting *RF* distribution is roughly uniform.

```
for pr = 1% to 100% incrementing by 100/m%
    S_1 = (V_1, E_1, 0)  where V_1 = {1,2} and E_1 = {(1,2)}
    for i = 2 to n, the step size
        Expand the graph by either splitting or connecting
          by the probability of pr.
        S_i = GetNextState(S_{i-1}, Pr)
    end
    store S_n = (V_n, E_n, RF_n) to the reference set.
end
Outputs:
    m number of random graphs whose splitting rates are known.
```

Fig. 4. Procedure for generating random graphs

The next phase is the nearest neighbor classification. Each random graph is represented by a vector of its graph invariants $G_i = (R_i, D_i, rd_i, RF_i) \quad i = 1, \cdots m$. A vascular network in question is represented by $G_q = (R_q, D_q, rd_q, RF_q)$ where RF_q is unknown. We can estimate RF_q by the RF_i of the random graph whose invariants are the most similar to the biological network's graph invariants.

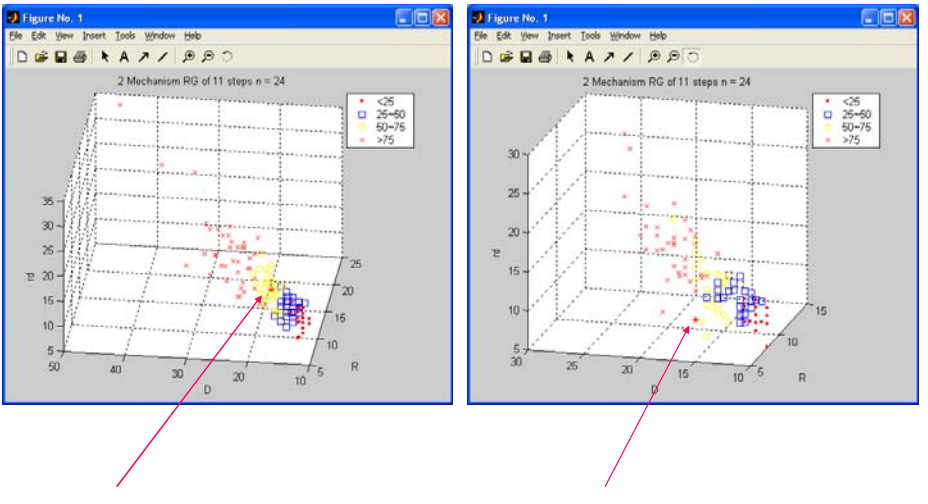
$$RF_q = \underset{RF_i \in \{RF_1,\cdots,RF_m\}}{\arg\min} \sqrt{(R_i - R_q)^2 + (D_i - D_q)^2 + (rd_i - rd_q)^2} \tag{2}$$

4 Experiments and Observations

Two major experiments were conducted to support our claim that the proportion of splitting of the renal glomerulus can be estimated by its invariants. The first experiment is used to show the correlation between graph invariants and the proportion of splitting and the latter one is to show the result on the biological renal glomerulus networks.

Table 1. Rat glomerular networks

study	|*V*|	*t*	*R*	*D*	*rd*
Shea	195	322	9	15	11
Remuzzi	247	403	9	15	12
Nyengaard	256	426	9	15	11
Antiga	302	460	12	18	12
Winkler	312	466	11	16	13
Wahl	358	595	12	19	11
Uremic	159	276	9	17	7
Newborn	24	35	4	7	5



Antiga's case (302, 460, 159, 12, 18, 12)
vs.
Closest one (302, 451, 150, 11, 17, 12)., i.e., 50%

Uremic case (159, 276, 119, 9, 17, 7)
vs.
Closest one (162, 241, 80, 9, 17, 8).
i.e., 65%.

Fig. 5. Graph invariant plots for renal glomerular networks and random graphs

Our algorithm is based on the hypothesis that there may be a correlation between graph invariants and the relative frequency of splitting. In order to demonstrate the correlation, $m = 1000$ random graphs whose step size, $n = 80$, are generated.

Observation 1. As the probability of splitting gets bigger, the graph invariants such as root distance, diameter and radius get bigger.

This phenomena is observed when the step size and the number of random graphs are sufficiently large; $n > 40$ and $m > 100$. The step size limit is sufficient for biological adult renal glomerular networks which we shall see in the next experiment.

We considered six normal adult rat renal glomerular networks, one adult rat with renal failure (uremic), and a five day old newborn rat. Their graph invariants are shown in Table 1. Their step sizes can be computed by eqn 1 and they are $n = 96$, 122, 127, 150, 155, 178, 78, and 11 respectively. Figure 5 shows the graph invariant plots for Antiga's case and uremic case. Random graphs are grouped into four clusters for visualization purposes. Our experimental results suggest that all normal adult renal glomerular networks have approximately ½ probability of splitting whereas the uremic case suggests a higher proportion of splitting.

5 Conclusion

We considered the problem of estimating splitting proportion of a given biological renal glomerulus network. A random graph process model was used to simulate the development of a vascular network. The correlation between the invariants of this vascular network modeled as a graph and the mechanisms of the growth of the network were studied. It was shown that the relative frequencies of sprouting and splitting during the growth of a given renal glomerulus can be estimated by the invariants (root distance, radius, and diameter) of the graph representing the renal glomerulus network.

References

1. W. Rosai, "Mechanisms of Angiogenesis," Nature 386, 17April, 671-674, 1997.
2. P. Carmaliet, "Angiogenesis in Health and Disease," Angiogenesis 9(6), 653-660, 2003.
3. M.L. Gargano, L.L. Lurie, L.V. Quintas,and E.M. Wahl, "A graph theory analysis of renal glomerular microvascular networks," *Microvascular Research* **67** 223-230, 2004.
4. E.M. Wahl, F.H.,Daniels, E.f.,Leonard, C. Levinthal, S. Cortell, "A Graph Theory Model of the Glomerular Capillary Network and Its Development," Microvasc. Res. 27, 96-109, 1984.
5. L.V. Quintas,and E.M. Wahl, "Random Graph Process Models for Angiogenesis," CSIS Pace University Technical Report No.183, 2002.
6. W. Feller, *An Introduction to Probability Theory and its Applications*, Vol. I, 3rd ed., John Wiley & Sons, New York, 1968.
7. K.T. Balińska and L.V. Quintas, The random *f*-graph process. *Quo Vadis, Graph Theory*, *Annals of Discrete Math*. **55**,, 333–340, 1993.
8. P. Carmeliet. Mechanisms of Angiogenesis and Arteriogenesis. Nature Med. 6(3), 389-395, 2000.
9. F. Buckley and F. Harary, Distance in Graphs. Addison-Wesley Pub Co., Reading, Massachusetts, 1990.

A Linear Generative Model for Graph Structure

Bin Luo[1,2], Richard C. Wilson[2], and Edwin R. Hancock[2]

[1] School of Computer Science and Technology,
Anhui University, P.R. China
[2] Department of Computer Science,
University of York,York YO10 5DD, UK
{luo, erh}@cs.york.ac.uk

Abstract. This paper shows how to construct a linear deformable model for graph structure by performing principal components analysis (PCA) on the vectorised adjacency matrix. We commence by using correspondence information to place the nodes of each of a set of graphs in a standard reference order. Using the correspondences order, we convert the adjacency matrices to long-vectors and compute the long-vector covariance matrix. By projecting the vectorised adjacency matrices onto the leading eigenvectors of the covariance matrix, we embed the graphs in a pattern-space. We illustrate the utility of the resulting method for shape-analysis.

1 Introduction

The literature describes a number of attempts at developing probabilistic models for variations in graph-structure. Some of the earliest work was that of Wong, Constant and You [11], who capture the variation in graph-structure using a discretely defined probability distribution. Bagdanov and Worring [10] have overcome some of the computational difficulties associated with this method by using continuous Gaussian distributions. For problems of graph matching Christmas, Kittler and Petrou [1], and Wilson and Hancock [2] have used simple probability distributions to measure the similarity of graphs. There is a considerable body of related literature in the graphical models community concerned with learning the structure of Bayesian networks from data [12]. However, despite this effort the methods fall well short of contructing genuine generative models from which explicit graph structures can be sampled. In this respect the study of graph-structures is less advanced than the study of pattern-vector or shape spaces. The reasons for limited progress are two-fold. First, graphs are not vectorial by nature. While conventional pattern recognition techniques construct shape-spaces from vectors, it is not straightforward to convert graphs into vectors. Second, in practice there usually exists structural noise or disturbance, and graphs are of different size.

To overcome these two problems, in a recent paper [4] we have explored how ideas from spectral graph theory can be used to construct pattern-spaces for

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 54–62, 2005.

sets of graphs. The idea here has been to extract features that are permuatation invariants from the adjacency matrices of the graphs under study. Pattern spaces may then be constructed from the feature-vectors using techniques such as principal components analysis, or more recently developed ones from manifold learning theory such as local linear embedding [3]. However, this work does not lead to a generative model. The reason for this is that it is not possible to reconstruct the graph adjacency matrices from the feature-vectors used to construct the pattern-spaces. Hence, although the embedding procedure can account for the statistical variations in the graph feature vectors, it does not account for the statistical variation in graph-struture that gave rise to the feature-vectors. In other words, the statistical variation in graph-structure remains hidden.

The aim in this paper is to combine ideas from the spectral analysis of graphs and linear deformable models to construct a simple generative model for graph-structure. One of the problems that limits the use of the probabilistic models described above, is that they are suffer from exponential complexity and are therefore not easily sampled from. To overcome this problem of exponential complexity we turn to the shape-analysis literature where principal components analysis has proved to be a powerful way of capturing the variations in sets of landmark points for 2D and 3D objects [5]. The idea underpinning the model is to convert graphs to pattern vectors by stacking the columns of their adjacency matrices to form long-vectors. From the covariance matrices for the long-vectors, there are a number of ways in which to construct pattern-spaces. The simplest of these is to construct an eigenspace by projecting the long-vectors onto the leading eigenvectors of the covariance matrix. The distribution of graphs so produced can be further simplified by fitting a manifold or a mixture model. However, here we use the eigenvectors of the covariance matrix to construct a linear model for variations in the adjacency matrices.

To do this we borrow ideas from point distribution models. Here Cootes and Taylor[5] have shown how to construct a linear shape-space for sets of landmark points for 2D shapes. We use a variant of this idea to model variations in the long-vectors for the standardised covariance matrices. The graphs are deformed by displacing the mean adjacency matrix long-vectors in the directions of the leading eigenvectors of the covariance matrix. Our method allows the pattern of edge-deformations to be learned and applied at the global level. This model may be both fitted to data and sampled. Distances between pairs of graphs can be defined as the Euclidean distance between two vectors of deformation parameters.

2 Graph Adjacency Matrices Vectorisation

In this paper, we are concerned with a set of N graphs $G_k, k = 1, 2, \ldots, N$. The kth graph is denoted by $G_k = (V_k, E_k)$, where V_k is the set of graph nodes and $E_k \subseteq V_k \times V_k$ is the edge-set of the graph. Since we adopt a graph-spectral approach in this paper, we characterise the edge-structure of graphs using adjacency matrices. The adjacency matrix A_k of graph G_k is a $|V_k| \times |V_k|$ matrix whose element with row index i and column index j is

$$A_k(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E_k \\ 0 & \text{otherwise} \end{cases} . \tag{1}$$

To construct our generative model of variations in graph structure, we will convert the adjacency matrices into a vector form where the entries are in a standard order. If the nodes of the graphs are ordered, i.e. they are labelled in a standard way, then can simply stack the columns of the adjacency matrix to form a $|V_k| \times |V_k|$ long-vector which we denote by z_k. Unfortunately, the standarised label order or correspondence order is rarely to hand in partical problems. Hence to construct the long-vector z_k, we need to permute the order of the rows and columns of the corresponding adjacency matrix A_k. To do this we use the graph-matching method reported by Luo and Hancock[6]. We represent the set of correspondences between the nodes of pairs of graphs using a correspondence matrix. For the graphs indexed k and l, the correspondence matrix is denoted by $S_{k,l}$. The elements of the matrix convey the following meaning,

$$S_{k,l} = \begin{cases} 1 & \text{if node } i \in V_k \text{ is in correspondence with node } j \in V_l \\ 0 & \text{otherwise} \end{cases} . \tag{2}$$

The graph matching method recovers the correspondence matrix using the EM algorithm and singular value decomposition of an adjacency structure correlation matrix. The algorithm commences from a Bernoulli model for the correspondence indicators, which are treated as missing data. From this distribution an expected log-likelihood function for the missing correspondence indicators is consructed. In the maximisation step of the algorithm, a singular value decomposition method is used to recover the correspondence matrix which satisfies the condition

$$S_{k,l} = arg \max_{S} Tr[A_k^T S A_l S^T]. \tag{3}$$

In other words, the maximisation likelihood correspondence matrices are those that maximise the correlation of the two adjacency matrices.

If we treat graph G_k as a reference graph, the permuted adjacency matrix of graph G_l can then be written as

$$A_l = S_{k,l} A_l S_{k,l}^T. \tag{4}$$

and its columns stacked to form the long-vector z_l. Differences in the numbers of graph nodes may be accommodated by selecting the reference graph G_k so that it has the largest number of nodes in the set under investigation, and allowing null-entries in $S_{k,l}$.

3 Linear Deformable Model for Graphs

Our aim is to construct a linear deformable model which can be used as a generative model for the variations is graph edge-structure. To do this, we represent the variations present in the set of graphs using the mean long-vector and the covariance matrix for the long-vectors. Deformations in graph structure are modelled

by perturbing the mean long-vector in the directions of the principal eigenvectors of the covariance matrix.

To be more formal, we commence by calculating the mean long-vector ($\hat{z}$) and the long-vector covariance matrix (Σ) for the set of permuted adjacency matrices using the following formulae

$$\hat{z} = \frac{1}{N}\sum_{k=1}^{N} z_k \qquad\qquad \Sigma = \frac{1}{N}\sum_{k=1}^{N}(z_k - \hat{z})(z_k - \hat{z})^T. \tag{5}$$

To construct the deformable model, we commence by computing the eigenvalues and eigenvectors for the covariance matrix Σ. The eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_N$ are found by solving the polynomial equations $|\Sigma - \lambda I| = 0$, where the I is the identity matrix. The associated eigenvectors $\phi_1, \phi_2, \ldots, \phi_N$ are found by solving the linear eigenvector equation $\Sigma\phi_k = \lambda_k\phi_k$. From the eigenvectors we construct a modal matrix. The eigenvectors are ordered in decreasing eigenvalue order to form the columns of the modal matrix, denoted by $\Phi = (\phi_1|\phi_2|\ldots|\phi_N)$. The linear deformable model allows the components of the adjacency matrix long-vectors to undergo displacement in the direction of the eigenvectors of the covariance matrix. For the long-vector of the graph G_k, the displaced vector is given by

$$\tilde{z_k} = \hat{z} + \Phi\gamma_k, \tag{6}$$

where γ_k is a vector of modal coefficients for graph G_k. The parameter vector γ_k measures the degree of displacement for the different vector components along the directions of the corresponding eigenvectors of the covariance matrix. The modal coefficient vectors are also a useful measure of the degree of graph deformation from the mean since they can be used for computing a measure of distance between graphs.

Once trained, i.e. the mean and covariance estimated, then the deformable model can be easily fitted to data by searching for the least-squares parameter vector. Suppose that the modal is to be fitted to the graph with standardised adjacency matrix long-vector z_k. The least-squares parameter vector satisfies the condition

$$\gamma_k = arg\min_b (z_k - \hat{z} - \Phi b)^T (z_k - \hat{z} - \Phi b) \tag{7}$$

and the solution is

$$\gamma_k = \frac{1}{2}\Phi^T \left\{z_k - \hat{z}\right\}. \tag{8}$$

Which is simply the projection of the centred long-vectors onto the space spanned by the covariance matrix long-vectors.

4 Experiments

In this section we provide experiments. We commence with examples on synthetic data, and then present real-world experiments on images of human faces.

4.1 Synthetic Data

We commence our experimental study with a synthetic example in which correspondences between nodes are known and the graphs have the same number of nodes, but a different edge structure. The set of synthetic images used in this study are shown in Figure 1. This is a set of perspective views of a house as it rotates. The associated graphs are shown in Figure 2. It is important to note that although the number of feature-points in this sequence remains the same, there are significant structural differences in the graphs in the different views.

In Figure 3 we show the eigespace projection of the graphs. The trajectory is well behaved and does not exhibit kinks, or fold back on itself. In addition, the points corresponding to neighbouring views are always closer to one-another than views that are not adjacent. This feature is underlined by the interpoint distance function which is shown in the right panel of the figure.

4.2 Real World Experiments

The data used in our experiments are provied by images in the BioID face databases [9]. For a subset of 27 images from the data-bases we have extracted 20 feature points. The graphs used in out studies are the Delaunay triangulations of the feature points. In Figure 4 we show a sample of the face images used, and the corresponding graphs generated are shown in Figure 5. Due to changes is expression, the relationships between the feature points changes and hence they give rise to different graph structures.

Once, the matrix Φ is estimated, then we can generate new graph-instances by selecting a parameter vector γ. The corresponding long-vector adjacency matrix is $z_g = \hat{z} + \Phi\gamma$. The long-vector z_g can be folded to give the adjacency matrix A_g.

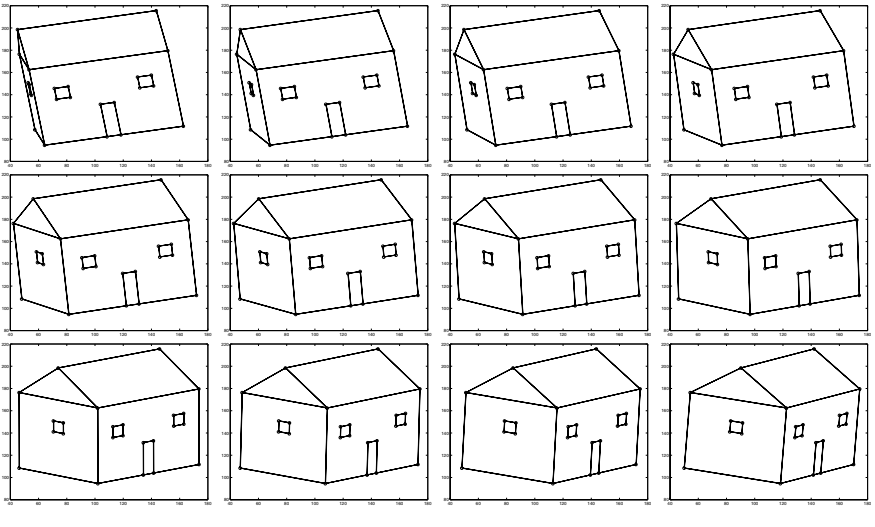


Fig. 1. Model sequence with feature points

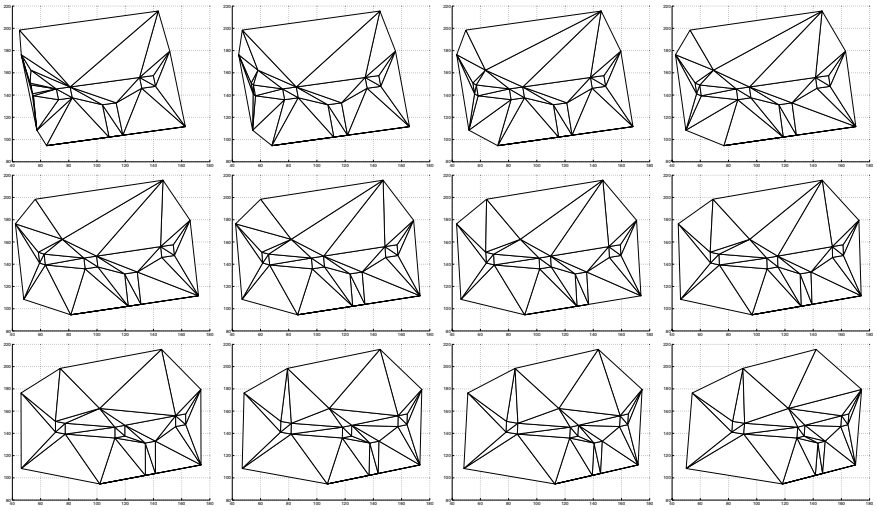


Fig. 2. Graph representation of the model sequence

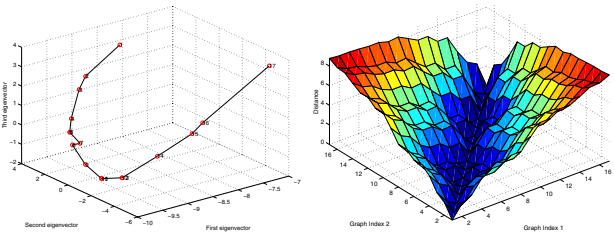


Fig. 3. Left eigenspace analysis of unweighted adjacency matrix, right distance matrix

Fig. 4. Face sequence

The parameter vector γ can be sampled from a prior distribution, however here we simply vary its components by hand. By setting the components correspoding

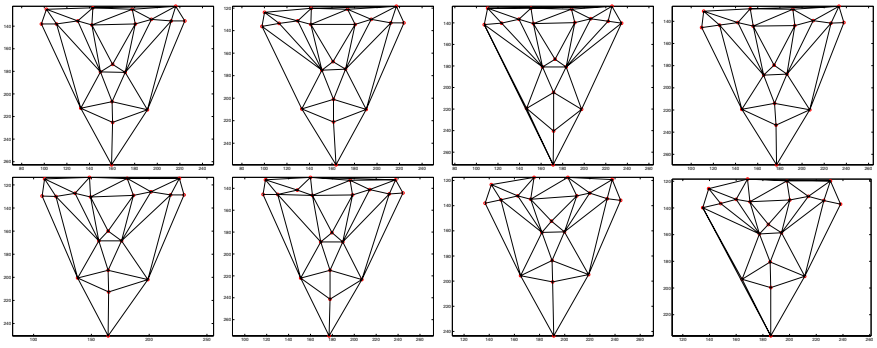


Fig. 5. Graph representation of the face sequence

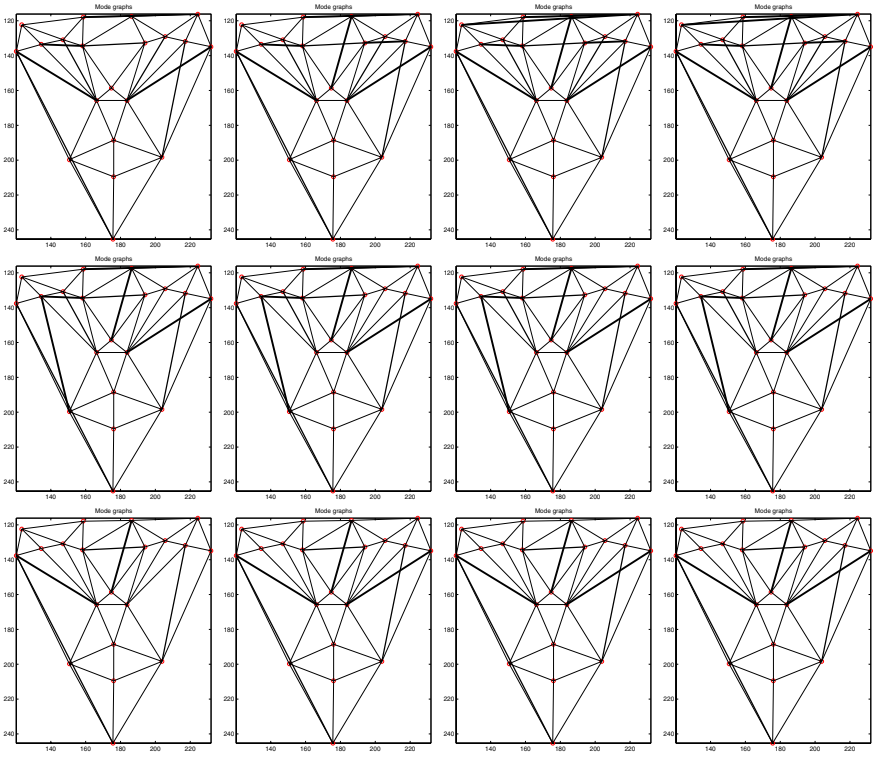


Fig. 6. Three modes of graph set BioID

to all but one of the covariance matrix vectors to zero, we can systematically explore the deformation modes of the learned structural model. The rows in Figure 6 shows the variations modes of along the three eigenvector directions corresponding the largest three eigenvalues. The different panels in the rows are

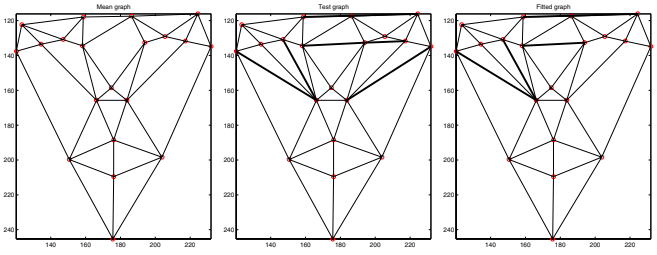


Fig. 7. Mean, test and fitted graphs

obtained by varying the relevant component of γ. There are clear differences in the structures captured by the different eigenmodes. The first mode to captures seprately the complex edge-structure around the eyes and the lips. The second eigenmode, represents the edge-connections between the eyes and the lips. The third eigenmode introduces left-right asymmetries.

Next we investigate the fitting of the model to data. The left panel of Figure 7 shows the mean graph of the BioID graph data-set. The middle panel is the test graph which is the 10th of the subset of BioID face images (and was not used in training). The right panel shows the least squares fit of the model to the test graph. Bold lines are used to show the differences of the mean graph with the mode graphs. The model fits the test graph well.

Finally, we show the projection of the face-graphs onto the space spanned by the three leading eigenvectors of the adjacency matrix in Figure 8. Two clear clusters emerge. Examination of the raw data shows that these correspond to the differences between neutral and exagerated expressions.

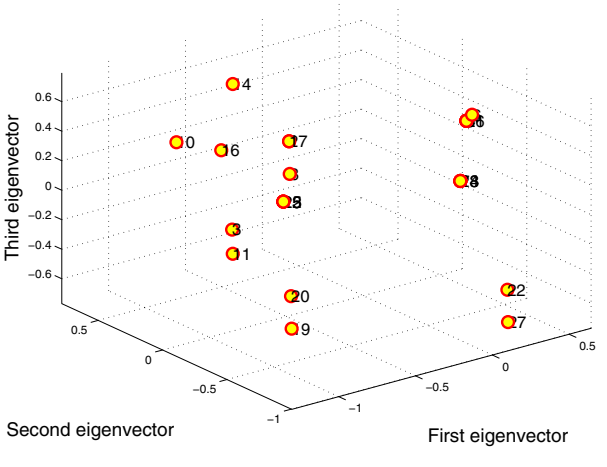


Fig. 8. Graph embedding

5 Conclusions

In this paper, we have presented a linear generative model that can be used to capture variations in graph-structure. The method involves converting the adjacency matrix into a long-vector and computing the sample mean and covariance matrix for the long vectors. New graph instances are generated by deforming the mean long-vector in the directions of the eigenvectors of the covariance matrix. Explicit adjacency matrices can by recovered by folding the long-vector. Experiments show that the method is capable of capturing plausible modes of variation in graph-structure.

There are a number of ways we intend to develop this work. First, we aim to explore in detail the structure of the shape manifold generated by the embedding procedure. Second, we aim to use the generative modal for organising large structural data-bases.

References

1. W.J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):749–764, 1995.
2. R.C. Wilson and E.R. Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634–648, June 1997.
3. Sam Roweis and Lawrence Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326 ,Dec.22, 2000.
4. Bin Luo, Richard C. Wilson and Edwin R. Hancock. Spectral embedding of graphs. Pattern Recognition, 36:2213–2230, 2003.
5. T.F Cootes, C.J. Taylor, D.H. Cooper, and J. Graham. Active Shape Models - Their Training and Application. *Computer Vision and Image Understanding*, 61:38–59, 1995.
6. B. Luo and E.R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *IEEE PAMI*, 23(10):1120–1136, 2001.
7. H. Murase and S.K. Nayar. Illumination planning for object recognition using parametric eigenspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(12):1219–1227, 1994.
8. Chatfield C. and Collins A.J. *Introduction to multivariate analysis*. Chapman & Hall, 1980.
9. Robert Frischholz and Ulrich Dieckmann. BioID: A multimodal biometric identification system. *IEEE Computer*, 33(2):64–68, 2000.
10. A.D. Bagdanov and M. Worring, First Order Gaussian Graphs for Efficient Structure Classification *Pattern Recogntion*, **36**, pp. 1311-1324, 2003.
11. A.K.C Wong, J. Constant and M.L. You, Random Graphs *Syntactic and Structural Pattern Recognition*, World Scientific, 1990.
12. D. Heckerman, D. Geiger and D.M. Chickering, Learning Bayesian Networks: The combination of knowledge and statistical data *Machine Learning*, **20**, pp. 197-243, 1995.

Graph Seriation Using Semi-definite Programming

Hang Yu and Edwin R. Hancock

Department of Computer Science,
University of York, UK

Abstract. Graph seriation is concerned with placing the nodes of a graph in a serial order so that edge consecutive constraints are generally preserved. It is an important task in network analysis problem in routine and bioinformatics. In this paper we show how the problem of graph seriation can be solved using semi-definite programming (SDP). This is a convex optimisation procedure that has recently found widespread use in computer vision. The main contribution of the paper is to detail the matrix representation needed to cast the graph-seriation problem in a matrix setting so that it can be solved using SDP. We illustrate the utility of the method for graph-matching and graph-clustering, where it is shown to offer advantages to the graph-spectral approach to seriation.

1 Introduction

The problem of placing the nodes of a graph in a serial order is an important practical problem that has proved to be theoretically difficult. The task is one of practical importance since it is central to problems such as network routing, the analysis of protein structure and the visualisation or drawing of graphs. Theoretically, the problem is a challenging one since the problem of locating optimal paths on graphs is one that is thought to be NP-hard [12]. The problem is known under a number of different names including "the minimum linear arrangement problem" (MLA) and "graph-seriation". There are also close links with the problem of locating steady state random walks in graphs.

Stated formally, the problem is that of finding a permutation of the nodes of a graph that satisfies constraints provided by the edges of the graph. The recovery of the permutation order can be posed as an optimisation problem. It has been shown that when the cost-function is harmonic, then an approximate solution is given by the Fiedler vector of the Laplacian matrix for the graph under study [8]. Thus, the solution to the seriation problem is closely akin to that of finding a steady state random walk on the graph, since this too is determined by the Laplacian spectrum. However, the harmonic functions does not necessarily guarantee that the nodes are arranged in an order that maximally preserves edge connectivity constraints. In a recent paper, Robles-Kelly and Hancock [2] have reformulated the problem as that of recovering the node permutation order to edge connectivity constraints, and have provided an approximate spectral solution to the problem.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 63–71, 2005.

Although spectral methods [14] are elegant and convenient, they are only guaranteed to locate solutions that are locally optimal. Recently, semidefinite programming (SDP) [11] has been developed as an alternative methods for locating optimal solutions couched in terms of a matrix representation. Broadly speaking, the advantage of the method is that it has improved convexity properties, and is less likely to became trapped in a non-global optimum. The method has been applied to a number of graph-based problems in pattern recognition including graph partitioning [6], segmentation [7] and the subgraph isomorphism problem [3]. For a mathematically detailed account see the survey by Alizadeh [4] which explains how semidefinite programming may be applied to combinatorial optimization.

The aim in this paper is hence to investigate whether SDP can be applied to the graph-seriation problem. We commence by illustrating how the cost-function of Robles-Kelly and Hancock can be encoded in a matrix form to which SDP can be applied. With this representation to hand, then standard SDP methods can be applied to extract the optimal serial ordering. To do this we lift the cost function to a higher-dimensional space. Here the optimization problem is relaxed to one of convex optimization, and the solution recovered by using a small set of random hyperplanes. In the experimental section, we illustrate the method on the problem of graph-matching [15], where it appears to outperform the spectral method used by Robles-Kelly and Hancock.

2 Graph Seriation

We are concerned with the undirected graph $G = (V, E)$ with node index-set V and edge-set $E \subseteq V \times V$. The adjacency matrix A for the graph is the $V \times V$ matrix with elements

$$A(i,j) = \begin{cases} 1 & \text{if}(i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The graph seriation problem has been formally posed as one of optimisation in the work of Atkins *et al* [8] and Robles-Kelly [2]. Formally, the problem can be stated as finding a path sequence for the nodes in the graph using a permutation π which will minimize the penalty function

$$g(\pi) = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} A(i,j)(\pi(i) - \pi(j))^2 \tag{2}$$

Since the task of minimizing g is NP-hard due to the discrete nature of the permutation, a relaxed solution is sought using a function h of continuous variables x_i. The relaxed problem can be posed as seeking the solution of the constrained optmisation problem

$$\min h(x) = \sum_{(i,j)} f(i,j)(x_i - x_j)^2 \tag{3}$$

$$s.t. \sum_i x_i = 0, \text{and} \sum_i x_i^2 = 1$$

Using graph-spectral methods, Atkins and his coworkers showed that the solution to the above problem can be obtained from the Laplacian matrix of the graph. The Laplacian matrix is defined to be $L = D - A$ where D is a diagonal matrix with $d_{i,i} = \sum_{j=1}^{n} A_{i,j}$. The solution to the relaxed seriation problem (4) is given by the Fiedler vector, i.e. the vector associated with the smallest non-zero eigenvalue of L. The required serial ordering of the is found by sorting the elements of the Fiedler vector into rank-order. Recently, Robles-Kelly and Hancock [2] have extended the graph seriation problem by adding edge connectivity constraints. The graph seriation problem4 is restated as that of minimising the cost-function

$$h_E(x) = \sum_{i=1}^{|V|-1} \sum_{k=1}^{|V|} (A(i,k) + A(i+1,k))x_k^2 \tag{4}$$

By introducing the matrix

$$\Omega = \begin{bmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 \\ 0 & 2 & 0 & 0 \cdots 0 & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & 0 \cdots 2 & 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 \end{bmatrix}$$

the path connectivity requirement is made more explicit. The minimiser of $h_E(\boldsymbol{x})$ satisfies the condition

$$\lambda = arg \min_{x_*} \frac{\boldsymbol{x_*}^T \Omega A \boldsymbol{x_*}}{\boldsymbol{x_*}^T \Omega \boldsymbol{x_*}} \tag{5}$$

Although elegant and convenient, spectral methods are are only guaranteed to find a locally optimal solution to the problem. For this reason in this paper we turn to the more general method of semidefinite programming to locate an optimal solution which utilizes the convexity properties of the matrix representation.

3 Semidefinite Programming

Semidefinite programming (SDP) is an area of intense current topical interest in optimization. Generally speaking, the technique is one of convex optimisation that is efficient since it uses interior-point methods. The method has been applied to a variety of optimisation tasks including combinatorial optimization, matrix completion and dual Lagrangian relaxation on quadratic models. Semidefinite programming is essentially an extension of ordinary linear programming, where the vector variables are replaced by matrix variables and the nonnegativity elementwise constraints are replaced by positive semidefiniteness. The standard form for the primal problem is:

$$\begin{aligned} \min \quad & traceCX \\ s.t.\ & traceF_iX = b_i \ i = 1...m \\ & X \succeq 0 \end{aligned} \tag{6}$$

where C, F_i and X are real symmetric $n \times n$ matrices and b_i is a scalar. The constraint $X \succeq 0$ means that the variable matrix must lie on the closed convex cone of positive semidefinite solutions. To solve the graph seriation problem using semidefinite programming, we denote the quantity $\Omega^{1/2}A\Omega^{-1/2}$ appearing in Equation 5 by B and $\Omega^{1/2}x_*$ by y. With this notation the optimisation problem can be restated as

$$\lambda = arg \min_{y^Ty=1} y^TBy \tag{7}$$

Noting that $y^TBy = trace(Byy^T)$ by letting $Y = yy^T$ in the semidefinite programming setting the seriation problem becomes

$$\begin{aligned} \min \quad & \text{trace } BY \\ s.t.\ & \text{trace } EY = 1 \end{aligned} \tag{8}$$

where the matrix E is the unit matrix, with the diagonal elements set to 1 and all the off-diagonal elements set to 0. Note that $Y = yy^T$ is positive semidefinite and has rank one. As a result it is convex and we can add the positive semidefinite condition $Y \in S_n^+$ where S_n^+ denotes the set of symmetric $n \times n$ matrices which are positive semidefinite.

3.1 Interior Point Algorithm

To compute the optimal solution Y^*, a variety of iterative interior point methods can be used. By using the SDP solver developed by Fujisawa et.al [9], a primal solution matrix Y^* can be obtained. Using the solution Y^* to the convex optimization problem (9), we must to find an ordered solution y to the original problem (7). To do this we use the randomized-hyperplane technique proposed by Goemans and Williamson [13].

Since $Y^* \in S_n^+$, by using the Cholesky decomposition we have that $Y = V^TV, V = (v_1, ...v_n)$.Recalling the constraint $y^Ty = 1$, the vector y must lie on a unit sphere in a high dimensional space. This means that we can use the randomized hyperplanes approximation. This involves choosing a random vector r from the unit sphere. An ordered solution can then be calculated from $Y^* = V^TV$ by ordering the value of v_i^Tr. We repeat this procedure multiple times for different random vectors. The final solution y_* is the one that yields the minimum value for the objective function y^TBy. This technique can be interpreted as selecting different hyperplanes through the origin, identified by their normal r, which partition the vectors $v_i, i = 1....n$.

The solution vector x_* can be obtained using the equation $\lambda^{1/2}x_* = y$, and the elements of the vector x_* then can be used to construct the serial ordering of the nodes in the graph. Commencing from the node associated with the largest component of $x*$, we sort the nodes in so that the nodes are ordered so that the

components of $x*$ are decreasing magnitude and also satisfy edge connectivity constraints on the graph. We iteratively proceed in the following. Let us denote the list of the nodes visited by Sk at the kth iteration. Initially $S_1 = i_i = argmax_i x_(i)$. We proceed by searching the set of the first neighbours of i_1, i.e. $N_{i1} = \{j|(i1, j) \in E\}$, to locate the node which is associated with the largest remaining component of x_*. This node is then appended to the list of nodes visited list and satisfies the condition $i_2 = argmax_{l \in N_{i_1}} x * (l)$. This process is repeated until every node in the graph is visited. At termination the sorted list of nodes is S_G.

4 Experiments

In this section, we provide an experimental evaluation of our new algorithm for graph seriation. Our experimental evaluation is divided into two parts. First, we explore how the serial graph can be used for graph matching. Second, we present results for the clustering of graphs using edit distances between seriated node sequences.

4.1 Graph Matching

By converting graphs to strings, they can be matched by minimising string edit distance. Here we use the probabilistic framework described by Robles-Kelly and Hancock [2]. For our experimental evaluation we use the COIL image database [16]. We compare the matching results with some alternative graph spectral matching algorithms. The investigated methods include the original spectral seriation method of Robles-Kelly and Hancock [1], the method of Shapiro and Brady [10] and that of Scott and Longuet-Higgins [5]. To extract graphs from the images, we first detect feature points using the Harris corner detector. The graphs used in our study are the Delaunay triangulations of the point sets. The reason for using the Delaunay graph is that it incorporates important structural information from the original image.

In the images studied there are rotation, scaling and perspecitve distortions present. Example images from the sequences are shown in Fig 1 and correspond to different camera viewing directions of the objects. The detected feature points and their Delaunay triangulations are overlayed on the images. In Fig 2, we show the correspondences located by performing string matching on the seriations of the graphs extracted from the images.

To test whether the algorithm is robust when confronted by corruption and noise, we also perform experiments on synthetic data. We have randomly generated a set of 50 2D points and constructed the associated Delaunay triangulations. We have simulated the effects of noise and structural error. We first add Gaussian errors to the point positions, while keeping the number of the points fixed. The parameter of the noise process is the standard deviation of the positional jitter. In Figure 3, we show the fraction of correct correspondences as a

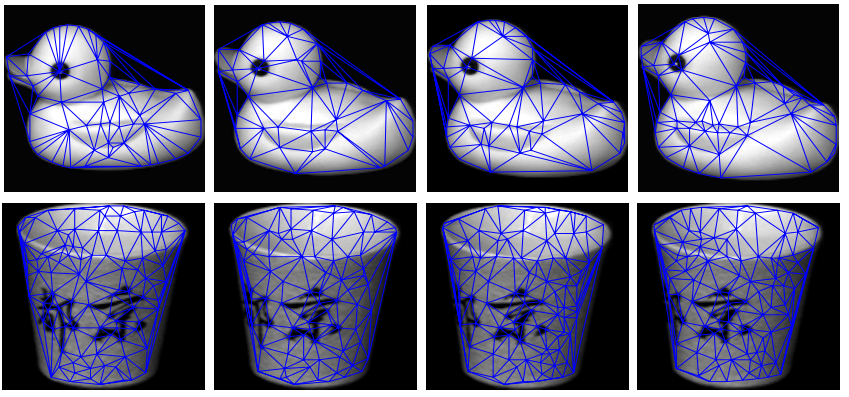


Fig. 1. Delaunay graphs overlayed on coil data

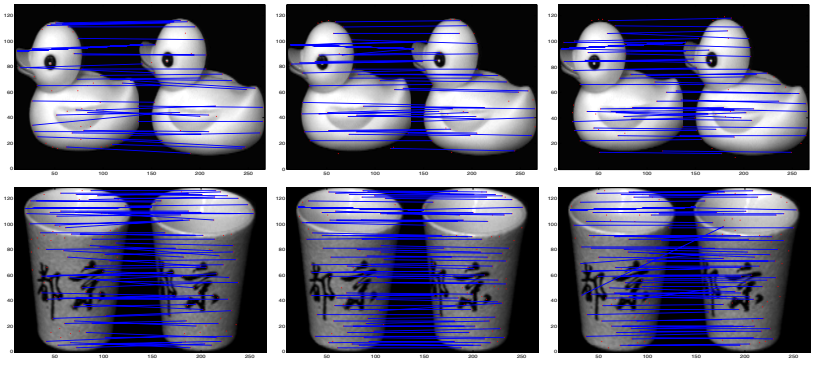


Fig. 2. Our algorithm for coil image sequences

function of the noise standard deviation for our method, Robles-Kelly and Hancock's spectral seriation method, Shapiro and Brady's [10] method, and Scott and Longuet-Higgins' method [5]. In Figure 4, we investigate the effect of structural noise. Here we have added a controlled fraction of additional nodes at random positions and have recomputed the Deluanay triangulations. We plot the fraction of correct correspondences as a function of the fraction of added nodes. The plot again compares the result of applying our method to the data, and the results obtained using spectral seriation, Scott and Longuet-Higgins' method, and Shapiro and Brady's method. The main feature to note from the plot is that the two seriation methods outperform the spectral methods of Scott and Longuet-Higgins and Shapiro and Brady by a significant margin. Our SDP seriation method offers a margin of improvement over the spectral seriation method.

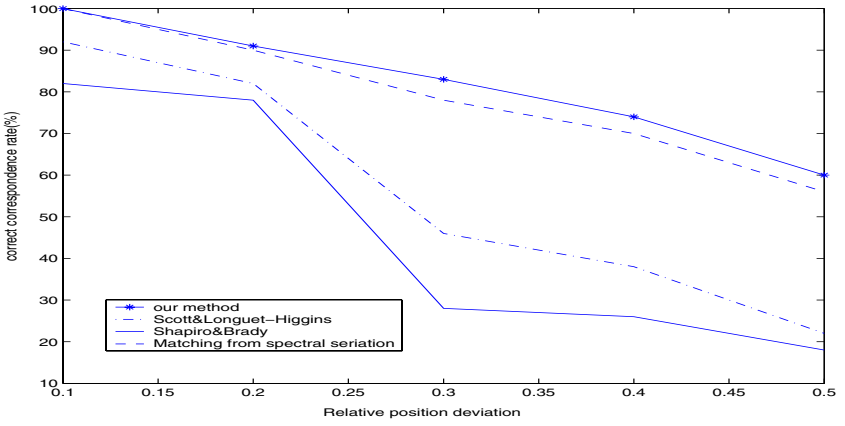


Fig. 3. Comparison of four methods for graphs with same number of nodes

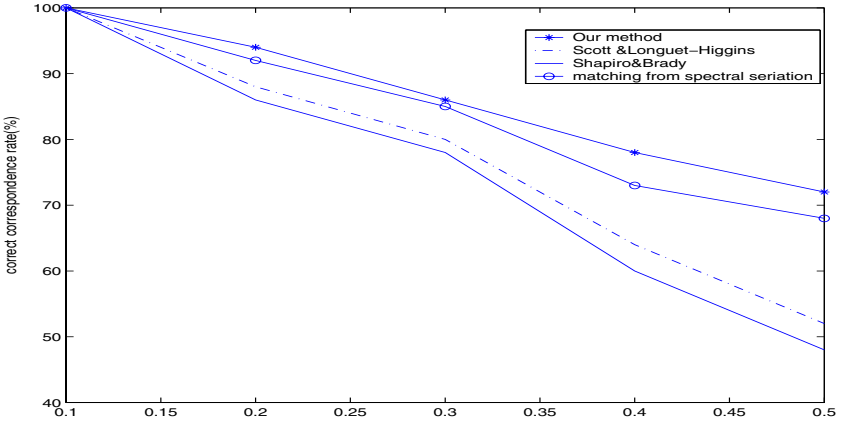


Fig. 4. Correspondences for three methods for graph matching with differing numbers of nodes

4.2 Graph Clustering

We have selected for objects from the COIL database. For each object there are 72 different views. For the 288 graphs in the data-set, we have computed the complete set of distances between each pair of graphs. We have explored clustering of the graphs using the following procedure. First, we convert the graphs into strings using our SDP seriation method. Second, the pair-wise correspondences between two different graphs in the set are located. Finally we compute the edit distances by using the correspondences on the serialized strings. We apply multidimensional scaling [17] on the matrix. The results are shown in Figure 5. The different views of the same object are shown as points of the same colour. From the figure it is clear that the different objects are well separated and form distinct clusters.

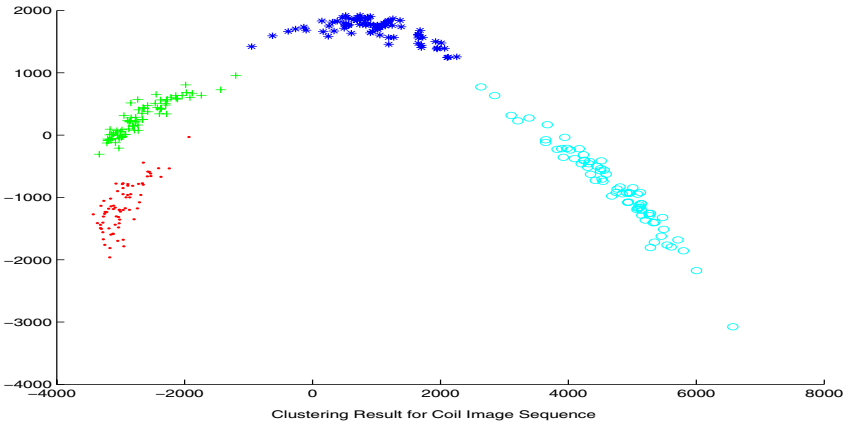


Fig. 5. Clustering Result

5 Conclusions

In this paper we have shown how the graph-seriation problem can be solved using semi-definite programming. This is convex optimisation procedure that uses randomised hyperplanes to locate the solution. We have applied the resulting technique to the problem of computing graph edit distances. Both the graph-matches and graph clusters produced by the method are significantly better than those delivered by standard spectral methods, and also offer a useful margin of improvement over the standard spectral method of seriation.

There are a number of ways in which the work reported in this paper can be extended. First, we aim to explore whether a more sophisticated string matching method based on hidden Markov models can be used in place of the simple error model used in this paper. Second, we plan to use the method to learn a generative model of graph variation.

References

1. A.Robles-Kelly and E.R.Hancock. Graph matching using spectral seriation. *Energy Minimisation Methods in Computer Vision and Pattern Recognition*, pages 517–532, 2003.
2. A.Robles-Kelly and E.R.Hancock. Graph Edit Distance from Spectral Seriation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, To appear, 2004.
3. C.Schellewald and C.Schnőrr. Subgraph Matching with Semidefinite Programming. *Proceedings IWCIA (International Workshop on Combinatorial Image Analysis), Palermo, Italy*, 2003.
4. F.Alizaheh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J.Optim*, 5:13–51, 1995.
5. G.Scott and H.Longuet-Higgins. An algorithm for associating the features of two images. *Proceedings of Royal Society of London Series*, 244:21–26, 1991.

6. Henry Wolkowicz and Qing Zhao. Semidefinite Programming relaxation for the graph partitioning problem. *Discrete Appl. Math*, pages 461–479, 1999.
7. J.Keuchel,C.Schnőrr,C.Schellewald,and D.Cremers. Binary Partitioning, Perceptual Grouping, and Restoration with Semidefinite Programming. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 25(11):1364–1379, 2003.
8. Jonathan E. Atkins, Erik G. Boman and Bruce Hendrickson. A Spectral Algorithm for Seriation and the Consecutive Ones Problem. *SIAM Journal on Computing*, 28(1):297–310, 1998.
9. K.Fujisawa,Y.Futakata,M.Kojima,K.Nakata and M.Yamashita. Sdpa-m user's manual. *http://sdpa.is.titech.ac.jp/SDPA-M*.
10. L.S.Shapiro and J.M.Brady. Feature-based correspondence-an eigenvector approach. *Image and Vision Computing*, 10:283–288, 1992.
11. L.Vandenberghe and S.Boyd. Semidefinite Programming. *SIAM Review*, 38(1):49–95, 1996.
12. M.Veldhorst. Approximation of the consecutive one matrix augmentation problem. *J.Comput.*, 14:709–729, 1985.
13. M.X.Goemans and D.P.WIlliamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J.ACM*, 42(6):1115–1145, 1995.
14. Fan R.K.Chung. *Spectral Graph Theory*. AMS, 1997.
15. Umeyama. S. An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. PAMI*, 10(5):695–703, 1988.
16. S.K.Nayar S.A.Nene and H.Murase. Columbia object image library (coil-100). *Technical Report, CUCS-006-96*, 1996.
17. T.F.Cox and M.A.A.Cox. *Multidimensional scaling*. Chapman and Hall, 1993.

Comparing String Representations and Distances in a Natural Images Classification Task

Julien Ros[1], Christophe Laurent[1], Jean-Michel Jolion[2], and Isabelle Simand[2]

[1] France Telecom R&D - TECH/IRIS,
4, rue du Clos Courtel, 35512 Cesson Sévigné Cedex - France
{julien.ros, christophe2.laurent}@francetelecom.com
[2] LIRIS, FRE CNRS 3672 INSA,
Bât. J. Verne, INSA Lyon,
69621 Villeurbanne cedex - France
{jean-michel.jolion, isabelle.simand}@liris.cnrs.fr

Abstract. This paper shows how strings can be used in a natural images classification task. We propose to build an attributed string from a set of regions of interest detected thanks to an interest point detector. These salient zones are characterized by local signatures describing singularities and they are linked by using graph seriation algorithms and perceptual methods. Once each image is represented by a string of signatures, we propose to use string-based edit distances and an ordered histograms-based distance in order to perform the classification task. Experiments have shown that whereas seriation algorithms give approximately the same results, the ordered histogram based distance is more efficient for the considered application.

1 Introduction

Nowadays, digital images are more and more present in the cyberworld. Indeed, peer to peer sharing, digital camera and Internet network provide an access to a lot of images for most people. As image databases grow exponentially, people need to have powerful solutions to manage them. Image classification is one such solution allowing to group images into semantically meaningful categories. It can thus be helpful for daily tasks such as browsing, annoting, indexing, etc. In this paper, we are interested in classification methods using low level image features. Such image clustering approaches perform first a feature extraction step in order to reduce the amount of data and to extract relevant and discriminating measures used during the classification step. This extraction phase results in a feature vector (also called *signature*) describing the image content.

Classically, image recognition approaches extract the image signature by considering the image content as a whole. Signatures can describe color by using e.g. classical histograms [16] or even texture [11] by using e.g. Gabor filter banks. However, during the last decade, it has been shown that better classification

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 72–81, 2005.

rates can be obtained by computing signatures around only a limited number of pixels called *interest points*. Recognition is then performed thanks to registering algorithms. In this case, the problem is clearly the lack of ordering between the interest points because the image can no longer be considered as a vector, increasing thus the complexity of the classification step. In this paper, we propose to define such an order thanks to two main approaches : a spectral graph seriation approach and a saliency-based approach. The image is then described by a string of local signatures, each one characterizing singularities in the region of interest thanks to a foveal wavelet descriptor [13]. Finally, the last classification step can be performed by a distance between strings. We have tested some of them and present the classification results.

The paper is organized as follows. Section 2 presents the salient points detector used in order to define the string nodes. Section 3 describes the different methods that can be envisaged to generate a string from a set of salient points. These strings are then compared thanks to some distances presented in section 4. Experiments comparing these approaches are presented in section 5 and finally, section 6 concludes this paper.

2 Interest Points Detection

The use of interest points for image retrieval was proposed in [3, 14] and was motivated by the definition of special points which capture only the relevant information of the signal. Consequently, assigning a local signature to each region of interest centered on an interest point could be more discriminative of the image content than computing a global signature. Nevertheless, finding interest points is quite difficult because it requires the definition of what is perceptually relevant in a signal.

Many approaches have been proposed in the literature to detect interest points. In [5], an algorithm using the local auto-correlation of the image localizes them on corners. Although this detector is very often used [14], it has the drawback of positioning the points on textured regions omitting other regions which can be critical for the classification. Moreover, there is no perceptual justification about the importance of corners. In [2], the authors propose a detector that locates interest points in high contrast area. Finally, observing that multiresolution, orientation and frequency analysis are of prime importance for the Human Visual System, some wavelet-based detectors have been proposed in [7, 9] that locate points on sharp region boundaries.

The detector proposed in [7] is used in our system and proceeds as follows:

- a discrete wavelet transform [10] is firstly performed on the image I up to a resolution level 2^r $(r \leq -1)$;
- the obtained wavelet coefficients are zerotree represented [15] resulting in a hierarchical data structure (tree) of wavelet coefficients;
- this tree is traversed a first time from leaves to the root node by computing at each resolution 2^j $(j \leq -1)$ a saliency map $S^I_{2^j}$ reflecting the perceptual

Original image Wavelet detector Harris detector

Fig. 1. Interest points detection

relevance of the wavelet coefficients present in the level 2^j. The saliency value $S^I_{2^j}(x,y)$ at the location (x,y) is defined by:

$$\begin{cases} S^I_{2^{-1}}(x,y) = \alpha_{-1}\left(\frac{1}{3}\sum_{s=1}^{3}\frac{|w^s_{2^{-1}}(x,y)|}{|Max(D^s_{2^{-1}})|}\right) \\ S^I_{2^j}(x,y) = \frac{1}{2}\left(\alpha_j\left(\frac{1}{3}\sum_{s=1}^{3}\frac{|w^s_{2^j}(x,y)|}{|Max(D^s_{2^j})|}\right)\right. \\ \left. + \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} S^I_{2^{j+1}}(2x+u,2y+v)\right) \end{cases} \quad (1)$$

where $w^s_{2^j}(x,y)$ stands for the wavelet coefficient of the subband $D^s_{2^j}$ located at (x,y), $Max(D^s_{2^j})$ $(s=1,2,3)$ denotes the maximum wavelet coefficient value over the detail subband $D^s_{2^j}$, and α_k (with $k \in [r,-1]$ and $0 \leq \alpha_k \leq 1$) is a weighting factor balancing the importance of saliency values with respect to the resolution level;

– from the saliency maps previously computed, the tree is traversed a second time from the root to the leaves in order to choose, at each tree level, the most salient wavelet coefficients.

The final result of these different steps is the construction of a saliency map S^I with the same resolution as I and that reflect the perceptual importance of the pixels. Indeed, the higher is $S^I(x,y)$, the more the pixel (x,y) is perceptually important. If N interest points are needed, then the N pixels with highest coefficients $S^I(x,y)$ in the saliency map are chosen.

As it can be seen on Figure 1, this interest point detector locates points on sharp region boundaries. The points are also more spread than the classical Harris corner detector [5] in the case of textured images.

3 Strings Construction

Interest points are usually mixed with registration techniques [14] for assessing similarity between images. In these approaches, each point is considered independently of each other and the dependencies or correlation that may exist between them are not used. However, it is well known that human eyes are able to classify an image from a set of focus of attention and saccadic eye movements. We

propose thus to link the detected interest points in order to construct a string composed of local signatures which describe each region of interest centered on an interest point. Therefore, images comparison can be performed by a string comparison. Several techniques to construct such strings can be considered and we propose to compare some of them.

3.1 Graph Seriation

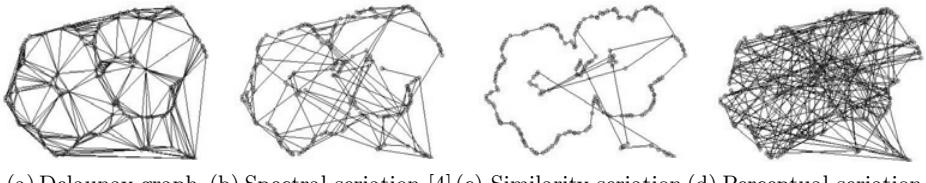
From a set of interest points, an attributed graph can be generated thanks to a Delaunay triangulation (see Figure 2(a)) in which each node of the graph is described by a local signature describing the image content in the neighborhood of the node. This graph can then be transformed into a string by a graph seriation approach [1, 4]. Two kinds of graph seriation are studied in this paper.

Spectral Graph Seriation. In the spectral graph seriation approach, the string is constructed by only considering the adjacency matrix of the graph and implicitly its structure. In [4], the authors propose an algorithm which performs the graph seriation by using the eigenvector ϕ^* corresponding to the leading eigenvalue of the adjacency matrix. Nodes are then ordered in the decreasing order of their magnitude in the leading eigenvector components (see Figure 2(b)).

Similarity Spectral Graph Seriation. In the case of attributed graphs, it seems relevant to consider nodal values and thus the similarity between them. The idea was first proposed in [1] where the seriation is performed thanks to the Fielder vector of the Laplacian matrix. In the following, we propose an alternative of it.

As in [4], we begin from the node associated with the largest component of ϕ^*. Next, we search through the set of the nearest neighbors, the node which is the most similar to the previous one in the sense of a L_2 distance between the foveal wavelets signatures [13] associated to the nodes being compared. This step is repeated until all nodes have been visited.

This method permits to generate strings following edges (see Figure 2(c)). Indeed, a foveal signature characterizes orientation and regularity of an edge, thus two signatures are similar if they belong to the same edge. Note that this method is similar to [1] where the Laplacian matrix is replaced by a similarity



(a) Delaunay graph (b) Spectral seriation [4] (c) Similarity seriation (d) Perceptual seriation

Fig. 2. String construction from a spatial distribution of interest points. The orignal image is shown on Figure 1

matrix $A = [a_{ij}]$ where $a_{ij} = d(s_i, s_j)$ with s_i being the local signature associated to the node x_i and d the distance between local signatures.

3.2 Perceptual Seriation

The second method presented in this paper consists in building a string by considering only the saliency values of the detected interest points (see equation 1). In this case, interest points are ordered in the decreasing order of the saliency value magnitude (see Figure 2(d)).

4 Strings Comparison

Once strings are constructed by one of the methods exposed in section 3, the last step consists in matching them to assess similarity between images.

4.1 Edit Distance and its Variants

Strings comparison can be first performed by using distances proposed in the field of automatic spelling correction or texts comparison. Such distances are based on the work of Levenstein presented in [8]. If we denote by Σ a finite alphabet and by $X = (x_1 x_2 ... x_n)$ and $Y = (y_1 y_2 ... y_m)$ two finite strings whose elements are in Σ then the string edit distance $D(X, Y)$ is the minimum cost needed to transform X into Y using elementary edit operations. These edit operations are of three kinds:

- $(x_i \rightarrow y_j)$ is the substitution of the symbol x_i by y_j;
- $(x_i \rightarrow \epsilon)$ denotes the suppression of the symbol x_i;
- $(\epsilon \rightarrow y_j)$ denotes the insertion of the symbol y_j.

If a cost function γ is assigned to each of these edit operations, then the string edit distance can be efficiently computed in $O(mn)$ thanks to a dynamic programming algorithm [18] based on the following recursive property:

$$D(i,j) = min \begin{cases} D(i-1, j-1) + \gamma(x_i, y_j) \\ D(i-1, j) + \gamma(x_i, \epsilon) \\ D(i, j-1) + \gamma(\epsilon, y_j) \end{cases} \tag{2}$$

where $D(i, j)$ is the edit distance between the sub-strings $(x_1 ... x_i)$ and $(y_1 ... y_j)$. Nevertheless, in [12], the authors show that the classical string edit distance lacks some normalization because it does not consider the length of the strings to be compared. For example, if X and Y are two strings of length 2, they can have the same edit distance as two strings of length 50. However, it seems that in the second case, they are more similar. Consequently, the normalization of the edit distance by the length of the edit path was proposed. It leads to the following definition:

$$d(X, Y) = min_P \left(\frac{W(P)}{L(P)} \right) \tag{3}$$

where:

- P is an edit path from X to Y, $W(P)$ is the cost of this edit path;
- $L(P)$ is the length of the edit path.

The computation of the normalized edit distance is performed thanks to a fractional programming algorithm [17] with the same complexity than the edit distance algorithm.

Finally, a major improvement of the classical string edit distance is proposed in [19] where the authors propose to use the neighborhood of each symbol in the string to compute the distance. This work is based on the Markov field theory because it is shown that the classical string edit distance algorithm can be seen as a zero order Markov edit distance.

In the following, we use edit operations costs defined in [13]. The substitution cost $\gamma(x_i, y_j)$ is the L_2 distance between the two foveal signatures x_i and y_j and the insertion and deletion costs $\gamma(\epsilon, y_j)$ and $\gamma(x_i, \epsilon)$ are defined by the L_2 distance between the foveal signature considered (i.e. y_j or x_i) and the null signature (i.e. the signature filled with 0) corresponding to the signature of an homogeneous region.

4.2 Ordered Histograms-Based Distance

Histograms are known to be very powerful in the case of content-based image retrieval [16]. They permit to capture the essential statistics present in the images and the comparison of them is less expensive than string-based edit distance algorithms. Consequently, coupling these two approaches for comparing strings of local signatures can be of interest. In [6], the authors propose a distance which considers the order and the distribution of symbols present in a string. Nevertheless, this distance must be adapted in the case of strings of signatures whose values are continuous in a k-dimensional space. For this purpose, we propose to compute k distances defined in [6], each one for a component in a signature. Then, we sum them to get the final distance between the two strings. Furthermore, as underlined in [7], each image can be represented by a different number of salient points depending on the complexity of the image content. Consequently, the two strings to be compared can have different lengths breaking thus the triangular inequality of the distance proposed in [6]. This modification leads to obtain a dissimilarity measure.

The algorithm proceeds as follows for two attributed strings X and Y:

1: $d \leftarrow 0$
2: **for** $j : 1$ to k **do**
3: $\quad H_1^j \leftarrow 0$; $H_2^j \leftarrow 0$;$W \leftarrow 0$
4: $\quad$ **for** $i : 1$ to $\min(m, n)$ **do**
5: $\quad\quad H_1^j(x_i[j]) \leftarrow H_1^j(x_i[j]) + (\min(m, n) - i + 1) * c(x_i[j], y_i[j])$
6: $\quad\quad H_2^j(y_i[j]) \leftarrow H_2^j(y_i[j]) + (\min(m, n) - i + 1) * c(x_i[j], y_i[j])$
7: $\quad\quad W \leftarrow W + (\min(m, n) - i + 1) * c(x_i[j], y_i[j])$
8: $\quad$ **end for**

9: $d \leftarrow \sum_i |H_1^j(i) - H_2^j(i)|/W + d$
10: **end for**
11: return d

where $c(x_i[j], y_i[j])$ is the substitution cost between j^{th} elements of the two signatures x_i and y_i and H_1 and H_2 are respectively the histograms associated to X and Y. This algorithm is computationally less expensive than a string edit distance because it computes the distance in $O(\min(m, n)k)$. However, it makes the strong assumption that the two strings to be compared are perfectly aligned.

5 Experiments

In order to compare the different approaches, a supervised image classification system based on the k-nearest neighbors algorithm has been developed. We use a training images database which is divided into six clusters (Alps, football, cars, ships, flowers, space) (see Figure 3), each cluster being composed of 15 natural images. To test our system, we picked 90 different images that are proposed to the system for classification (see Figure 4). Regarding the parameters used during the experiments, the L_2 distance has been used to compute $c(x_i[j], y_i[j])$ in the ordered histograms-based distance algorithm (see section 4.2) and the histograms H_1^j and H_2^j are composed of 100 bins.

We have firstly compared the classification results obtained by each distance presented in section 4 for each seriation algorithm separately. Figure 5 presents the classification rates obtained by each method using different number of interest points and the computing times needed to compare two images using the different distances on a Pentium IV 3GHZ. It is clear that ordered histograms distance outperforms all other distances proving that the strong assumption discussed before is not so hard. Moreover, ordered histograms distance is computationally more efficient and it is essential in the case of large images databases. However, the main reason of these better classification rates is that our insertion and deletion costs for the edit distances are not well adapted and so they degrade

Fig. 3. Some image samples present in the training database

Fig. 4. Some image samples present in the test database

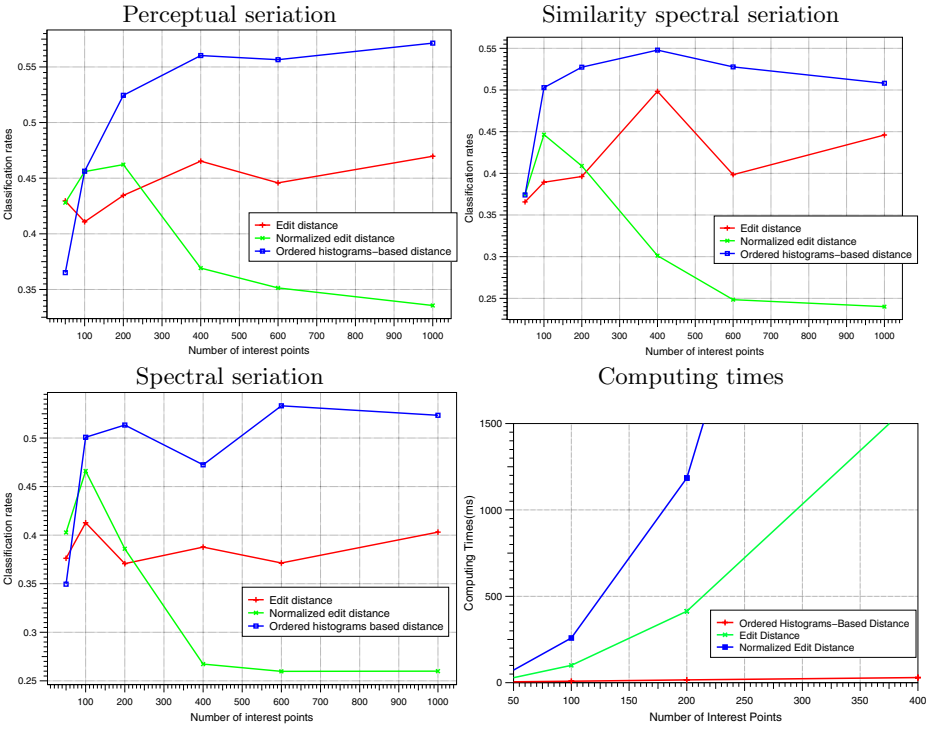


Fig. 5. Comparison of distances for the three seriation approaches

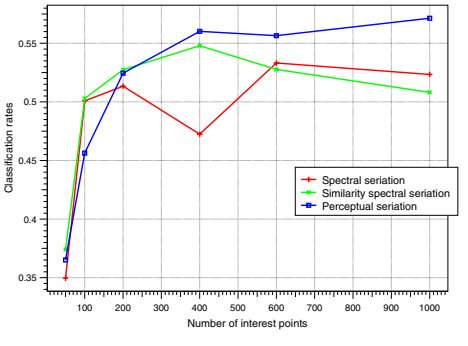


Fig. 6. Comparison of seriation methods

results. Thus, it could be interesting to take into account the probability density function of the foveal signatures in a cluster in order to define these edit costs.

If we compare the three seriation approaches using only the ordered histograms based distance (see Figure 6), we can see that the perceptual seriation gives the best result when more than 200 interest points are used. Nevertheless, classification rates are not so different. Finally, if we consider computing times,

it seems not relevant to use graph seriation approaches because they do not improve classification rates compared to a perceptual approach and they are more complex.

Finally, it is important to note that when color is not able to discriminate a cluster, our approach is well suited. In the dataset we used, it appears to be the case for the cars and ships clusters which obtain respectively 41% and 38% of good classification rates with a global histogram approach whereas our method permits to obtain 85% and 54% for 400 interest points.

6 Conclusion and Perspectives

In this paper, we have presented a comparison of seriation techniques and string distances that can be used in the case of a natural images classification task using foveal signatures. It has been shown that histograms-based distance gives better classification rates than the others edit distances presented. Nevertheless, it could be interesting to implement a training algorithm as in [6] in order to learn the edit costs which are very difficult to establish and on which depends critically the classification rates.

It has also been shown that seriation approaches give approximately the same classification rates indicating that string order is not very important in the recognition task. However, it could be interesting to implement the same approach in order to perform small substring matching and it is quite sure that we are also interested in other seriation approaches.

References

1. Atkins J.E., Boman E.G., and Hendrickson B. A Spectral Algorithm for Seriation and the Consecutive Ones Problem. *SIAM Journal on Computing*, 28(1):297–310, 1998.
2. Bres S. and Jolion J.M. Detection of Interest Points for Image Indexation. In *3rd Int. Conf. on Visual Information Systems*, pages 427–434, 1999.
3. Gouet V. and Boujeema N. Object-based queries using color points of interest. In *IEEE Workshop on Content-Based Access of Image and Video Libraries*, pages 30–36, December 2001.
4. Hancock E.R. and Vento M. Graph Matching Using Spectral Seriation and String Edit Distance. In *4th IAPR Int. Workshop (GbRPR 2003)*, volume 2726 of *Lecture Notes in Computer Science*. Springer, 2003.
5. Harris C. and Stephens M. A Combined Corner and Edge Detector. In *4th Alvey Vision Conference*, pages 147–151, 1988.
6. Jolion J.M. and Simand I. Representation d'Images par des Chaines de Symboles: Application à l'Indexation d'Images. In *COmpression et REprésentation des Signaux Audiovisuels (french workshop)*, 2004.
7. Laurent C., Laurent N., and Visani M. Color Image Retrieval Based on Wavelet Salient Features Detection. In *3rd Int. Workshop on Content-Based Multimedia Indexing*, pages 327–334, 2003.
8. Levenstein A. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Phys. Dokl.*, 10:707–710, 1966.

9. Loupias E., Sebe N., Bres S., and Jolion J.M. Wavelet-based Salient Points for Image Retrieval. In *IEEE Int. Conf. on Image Processing*, pages 518–521, 2000.
10. Mallat S. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.
11. Manjunath B.S. and Ma W.Y. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, 1996.
12. Marzal A. and Vidal E. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):926–932, 1993.
13. Ros J. and Laurent C. Natural Image Classification Using Foveal Strings. In *The International Workshop on Multidisciplinary Image, Video, and Audio Retrieval and Mining*, October 2004.
14. Schmid C. and Mohr R. Local Grayvalue Invariants for Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–535, 1997.
15. Shapiro J.M. Embedded Image Coding Using Zerotrees of Wavelet Coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.
16. Swain M.J. and Ballard D.H. Color Indexing. *International Journal on Computer Vision*, 7(1):11–38, 1991.
17. Vidal E., Marzal A., and Aibar P. Fast Computation of Normalized Edit Distances. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(9):899–902, September 1995.
18. Wagner R.A. and Fischer M.J. The String-to-String Correction Problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, 1974.
19. Wei J. Markov Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):311–321, March 2004.

Reduction Strings: A Representation of Symbolic Hierarchical Graphs Suitable for Learning

Mickaël Melki and Jean-Michel Jolion

Lyon Research Center for Images and Information Systems (LIRIS),
Bât. J. Verne, INSA Lyon, 69621 Villeurbanne Cedex, France
{melki, jolion}@rfv.insa-lyon.fr

Abstract. This paper proposes a new possible way to represent a symbolic graph pyramid built by successive applications of some rules. This representation is targeted for learning the rules, since it contains information about which rules were applied, but also because it can be used to easily define the space of all possible pyramids as a *reduction tree*. We also consider restraining the possible rules to some basic ones in order to make the reduction trees easy to build, and a way to use them for learning will be presented.

1 Introduction

The growth of computing capability of modern computers allows us to use more and more complex tools for pattern recognition. They include graph representations, among which hierarchical graphs which are the main frame of this work. Hierarchical graphs, or graph pyramids, have already been the purpose of many works for this kind of application, like in [3, 5, 4, 1, 2].

In [6], we proposed a way to build pyramids with symbolic features by using a set of rules considering both the structure and the data. The building process consisted in going through three phases for each level: an *exploration phase*, which purpose was to produce all the possible reduction hypotheses that apply to the graph, a *selection phase*, that was used to keep the best set of compatible hypotheses, and then a *reduction phase* in which the selected hypotheses were applied.

This process involved complex rules which should be learned. The scope of this paper is to deal with the learning process. It is a theoretical work, in which we propose to represent the pyramid as a string of hypotheses. This allows us to define a structure representing the space of all the possible pyramids. Then the learning process will consist in searching the best element in this space.

The first part of this paper deals with the definition of those strings of hypotheses, or *reduction strings*. The second part will define the *reduction trees*, which are the structures representing all the possible reduction strings. Then we will define some simple rules, less complex than the one we used in our previous work but more appropriate for building the reduction trees. Finally, we will explain how they are used for learning.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 82–91, 2005.

2 Reduction Trees

2.1 From Pyramids to Reduction Strings

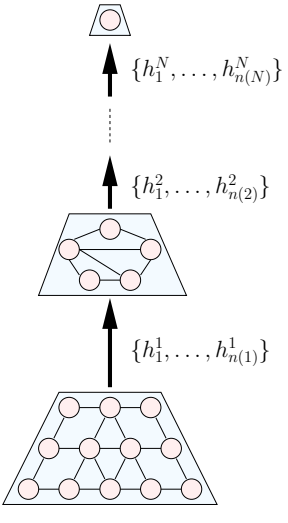
Let $G_0, \ldots, G_n$ be the levels of a pyramid, *e.g.* a hierarchy of graphs, built by the process described in the introduction. For any $i > 0$, let $h^i_1, \ldots, h^i_{n(i)}$ be the hypotheses selected during the selection phase for building level i. By construction, for each $i > 0$, we know that the hypotheses h^i_j ($j \in [\![1, n(i)]\!]$) are compatible, which means that we can apply them in any order (or in parallel).

The sequence $\mathcal{R}_i = h^i_1, \ldots, h^i_{n(i)}$ will be called a *reduction string from G_{i-1} to G_i*. It contains all the information needed for building level i from level $i-1$. We can also notice that any permutation of a reduction string from G_{i-1} to G_i is still a reduction string from G_{i-1} to G_i.

The concatenation of all the reduction strings $\mathcal{R}_1, \ldots, \mathcal{R}_n$ results in a sequence of hypotheses which, when applied in this order, allows us to build the top G_n of the pyramid from its base G_0. Such a sequence will be called *reduction string from G_0 to G_n*, or *complete reduction string of the pyramid*. Note that a permutation of a complete reduction string may not be a complete reduction string. The complete reduction string is an alternative way to represent the whole pyramid without having to store all the individual graphs: they contain both the contraction kernels, as defined by W. Kropatsch in [4], and the symbols to put in the contracted nodes. Figure 1 illustrate how reduction strings relate to the graph pyramids.

2.2 Standard Reduction Trees

Let G be a level of the pyramid, and $\mathcal{H}$ the set of all the possible hypotheses applicable to graph G (the ones produced during the exploration phase of the



Reduction strings:

- From G_0 to G_1:

$$\mathcal{R}_1 = h^1_1, \ldots, h^1_{n(1)}$$

- From G_1 to G_2:

$$\mathcal{R}_2 = h^2_1, \ldots, h^2_{n(2)}$$

- Complete reduction string:

$$\mathcal{R} = \underbrace{h^1_1, \ldots, h^1_{n(1)}}_{\mathcal{R}_1}, \underbrace{h^2_1, \ldots, h^2_{n(2)}}_{\mathcal{R}_2}, \ldots, \underbrace{h^N_1, \ldots, h^N_{n(N)}}_{\mathcal{R}_N}$$

Fig. 1. Pyramids and reduction strings

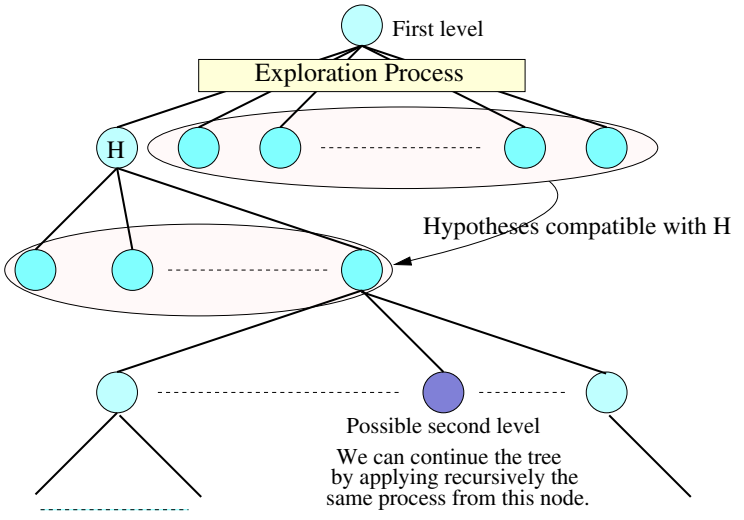


Fig. 2. Standard reduction tree

reduction process). The *reduction tree of level* G is the tree that represents all the possible reduction strings from graph G to any possible next level G' as paths from the root to the leaves. It can be built by induction using the following algorithm:

- Depth 0: the root is not labelled
- Depth 1: there are $|\mathcal{H}|$ elements of depth 1, each of which is labelled by one element of $\mathcal{H}$.
- If depth i is built, depth $i+1$ is built as follows: for each node s of depth i, let $\mathcal{H}_s$ be the hypotheses which label all the brothers of s, and $\mathcal{H}_s^*$ the subset of $\mathcal{H}_s$ containing only the hypotheses which are compatible with the hypothesis of s. Then the node s has $|\mathcal{H}_s^*|$ children, each of which is labelled by an element of $\mathcal{H}_s^*$ (if $\mathcal{H}_s^* = \emptyset$, then s has no child and is a leaf).

See figure 2 for an illustration of a standard reduction tree.

For each leaf s of the reduction tree of level G, the path from the root to s defines a reduction string, which once applied to graph G produce a new graph G_s. We can then repeat the process for each G_s, with leaf s becoming the root of the reduction tree of level G_s, and so on, until no more hypothesis can be found. The resulting tree is called the *standard reduction tree of graph* G, and the paths from the root to the leaves represent all the possible complete reduction strings of the pyramids of base G.

2.3 Extended Reduction Trees

The standard reduction trees represent all the possible reduction strings for a given graph G, and so will be useful in the learning process as they define a

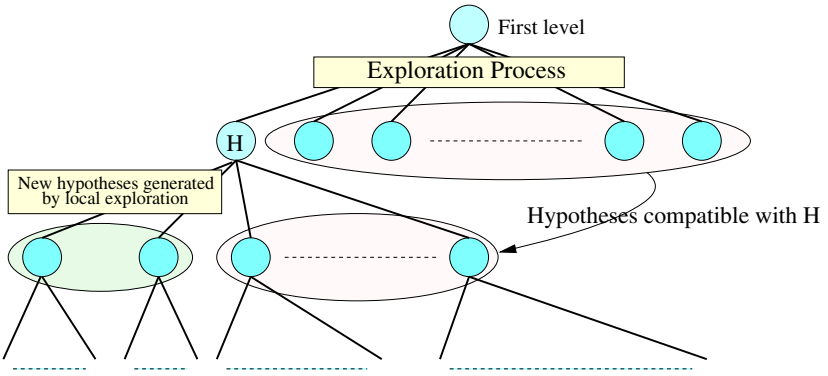


Fig. 3. Extended reduction tree

possible search space. But their construction is not regular, since some nodes in the tree are built by keeping compatible hypotheses from previous levels, whereas some other nodes are built by running an exploration phase. We propose another approach combining both a selection of compatible hypotheses and a local exploration for each level of the reduction tree, so that the construction of the tree is regular.

The *extended reduction tree* is built using the following algorithm:

1. Root has no label, and has a child for every hypothesis produced by the exploration algorithm over the initial graph.
2. For each node s at depth i, we build its children as follows:
   - We create a child for every hypothesis from the brothers of s that is compatible with the hypothesis of s.
   - We then do a local exploration for finding all the new hypotheses that can be produced once the hypothesis of s has been applied. We create a child for any of those hypotheses.
   - If no child has been created by both of these two steps, s is a leaf of the extended reduction tree.
3. We repeat step 2 for depth $i + 1$, until every node at depth i is a leaf.

The standard reduction tree is a subgraph of the extended reduction tree, which means that the latter allows us to consider some reduction strings that could not be considered if we used the former. They correspond to consecutive application of rules to the same region of the graph, which could not occur in the standard construction process. For that reason, a reduction string obtained from an extended reduction tree cannot always be equivalent to a pyramid built by the standard process. Figure 3 shows an example of an extended reduction tree.

3 Basic Rules

Extended reduction trees can be useful since they not only offer more possible reduction schemes than the standard ones, but they are also more convenient to build provided that the local exploration can easily be done. That is why we proposed to use some basic rules instead of the complex ones we used in our previous works. The complex rules can be emulated by applying sequentially many of those basic ones.

3.1 The Rules

We will use three kinds of basic rules:

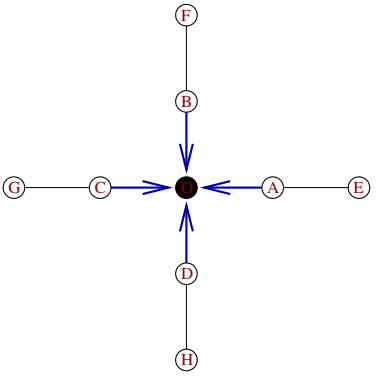
1. Relabelling rule: it transforms symbol S into symbol D without doing any contraction (notation: $S \rightarrow D$).
2. Contextual rule: it transforms symbol S into symbol D, only if the node to which it is applied has a neighbor with symbol S', without doing any contraction (notation: $S + S' \rightarrow D + S'$).
3. Contraction rule: it contracts a node with symbol S and one of its neighbor that has symbol S', and the resulting node gets symbol D (notation $S+S' \rightarrow D$).

3.2 Complex Rules

In order to emulate more complex rules like the ones introduced in [6] , we will need to add some new symbols, the *temporary symbols*, to the symbol set. Then, we will need to consider four sets of (basic) rules:

1. The *initialization set*, which usually contains only one contextual rule. This rule will identify one part of the configuration, and will mark one node with a temporary symbol.
2. The *propagation set*, which only contains contextual rules. Those rules will identify the whole configuration by marking any node of the configuration with a particular temporary symbol by propagating information.
3. The *contraction set*, which contains the contraction rules that are used to contract the contraction kernel of the complex rule. For complex rules whose contraction kernel is empty, the contraction set contains only one relabelling rule.
4. The *restoration set*, which contains the relabelling rules used to replace the remaining temporary symbols by the original symbols in the nodes.

To apply a complex rule, we will first apply the rule from the initialization set, which will introduce some temporary symbols in the graph. Then, the rules from the propagation set will be applied and will propagate temporary symbols through the whole configuration of the rule. A particular temporary symbol should then be produced by the last applicable propagation rule, which will make the rules from the contraction set applicable. Finally, the rules from the restoration set are applied to put the graph in the correct state. Note that if the



– Initial rule:
$0 + A \leftarrow \varepsilon_O$
– Propagation rules:

$$\begin{array}{ll} A + \varepsilon_O \rightarrow \varepsilon_A + \varepsilon_0 & E + \varepsilon_A \rightarrow \varepsilon_E + \varepsilon_A \\ \varepsilon_A + \varepsilon_E \rightarrow \varepsilon'_A + \varepsilon_E & \varepsilon_O + \varepsilon'_A \rightarrow \varepsilon'_O + \varepsilon'_A \\ B + \varepsilon'_O \rightarrow \varepsilon_B + \varepsilon'_0 & F + \varepsilon_B \rightarrow \varepsilon_F + \varepsilon_B \\ \varepsilon_B + \varepsilon_F \rightarrow \varepsilon'_B + \varepsilon_F & \varepsilon'_O + \varepsilon'_B \rightarrow \varepsilon''_O + \varepsilon'_B \\ \vdots & \vdots \end{array}$$

– Contraction rules:

$$\begin{array}{ll} \varepsilon''''_O + \varepsilon'_A \leftarrow S'''' & S'''' + \varepsilon'_B \leftarrow S''' \\ S''' + \varepsilon'_C \leftarrow S'' & S'' + \varepsilon'_D \leftarrow S \end{array}$$

– Restoration rules:
$\varepsilon_E \leftarrow E \quad \varepsilon_F \leftarrow F \quad \varepsilon_G \leftarrow G \quad \varepsilon_H \leftarrow H$

Fig. 4. From complex rules to basic rules (S being the output symbol of the rule)

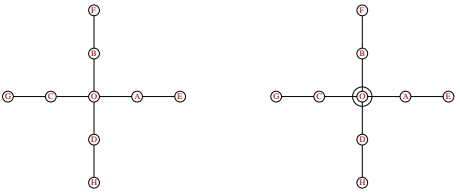
particular temporary symbol is not produced by the last propagation rule, the complex rule is not applicable. In that case, we have to go back in the reduction string up to the rule from the initialization set, and try to select another rule. Figure 4 illustrates how a complex rule is transformed into elementary basic rules, and Figure 5 shows how the rules are applied on a graph to produce the correct result.

In order to get the correct behavior with minimal special treatment, we decided to split all the basic rules we use into four sets, which correspond to the sets we have described (though the first set will both contain the rules from the initializing sets of the complex rules, and the individual basic rules). Rules get priority depending on which set they are in (the usual priority mechanism still applies within each set though), in the following order, from the higher to the lower priorities: contraction set, propagation set, restoration set, initialization set.

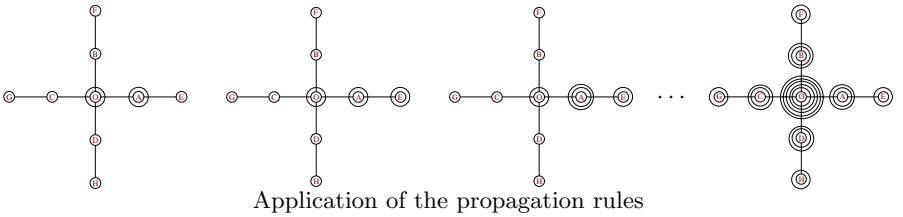
3.3 Transformation Cycles

We also need a mechanism for preventing cycle in the transformations sequence in order to get finite reduction strings. Since contraction rules decrease the number of nodes in the graph and no other rule can increase it, they are not involved into cycles, and we will only have to consider sequences of relabelling and contextual rules.

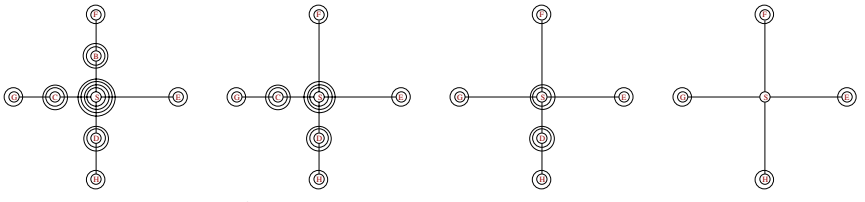
During building process, rules are applied according to their priority, so that if the graph returns to in the same state after some rules where applied, the same hypotheses will apply again in the same order. This means that we can detect cycles in the reduction process by finding cycles in the reduction string. This can easily be done using standard string search algorithms into the part of the reduction string from the last contraction rule to the end of the string. Each time a cycle is detected we remove the cycle from the reduction string and select another hypothesis.



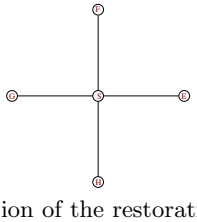
Left: initial graph, right: initial rule applied (circles are used to represent temporary symbols)



Application of the propagation rules



Application of the contraction rules



Application of the restoration rules

Fig. 5. How the rules from figure 4 apply

But during the learning process, the selection of the rules to apply may not be the same in each occurrence of the cycle. As only strings of relabelling and contextual rule can contain cycle, and since those two kinds of rules do not change the graph but only the labels in the nodes, we just have to consider how the labels evolve to detect cycles. This can still be complex, specially if the graph has many nodes, and so we need a more efficient solution. One possibility is to limit the size of sequences with no contraction rules: if a branch of the reduction tree that is being built (cf. next section) contains too many contextual or relabelling rules, the branch is deleted.

4 Learning Process

The goal of the learning process is to set the weights of the rules in such a way that the paths of the reduction tree corresponding to the strings that produce the best results according to the ground truth are promoted. Unfortunately the full reduction tree is usually too big to be built, so we have to build only parts of it using the current weights of the rules.

The learning is done by the following process:

1. We build a part of the reduction tree using current weights of the rules to know which branches to build.
2. Then, we promote the rules of the branches leading to the best results, whereas we dismiss the rules of the branches leading to the worst results
3. We repeat step 1, using the new weights.

4.1 Construction of a Partial Reduction Tree

In order to prevent the exponential growth of the reduction tree, we use the following algorithm to build a partial tree. Each node s contains a label $h(s)$ which is the hypothesis that should be applied if we go through this node. It also corresponds to the graph $G(s)$ resulting from the application of the reduction string obtained by following the path from the root of the graph to s. It also contains a set of applicable hypotheses $\mathcal{H}(s)$. Finally, it contains the length $l(s)$ of the sequence of contextual and relabelling hypotheses since the last contraction hypothesis.

1. Root r has the following properties: $h(r) = \emptyset$, $G(r) = G_0$, $\mathcal{H}(r)$ is obtained by applying the exploration process on graph G_0, and $l(r) = 0$.
2. For each node s in the last computed depth:
   - Select K hypotheses $h_1, \ldots, h_K$ from $\mathcal{H}(s)$ (using both their priority and a random decimation).
   - For each h_i, if h_i is a contraction hypothesis, s gets a son s_i such that $l(s_i) = 0$, $h(s_i) = h_i$, $G(s_i)$ is the graph obtained by applying h_i on $G(s)$ and $\mathcal{H}(s_i)$ is obtained by keeping all the element of $\mathcal{H}(s)$ which are compatible with h_i, and adding all the hypotheses produced by the local exploration on graph $G(s_i)$.
   - For each h_i, if h_i is not a contraction hypothesis and $l(s) < M$, s gets a son s_i such that $l(s_i) = l(s)+1$, $h(s_i) = h_i$, $G(s_i)$ is the graph obtained by applying h_i on $G(s)$ and $\mathcal{H}(s_i)$ is obtained by keeping all the elements of $\mathcal{H}(s)$ which are compatible with h_i, and adding all the hypotheses produced by the local exploration on graph $G(s_i)$.
3. Among all the nodes produced during step 2, only the best N nodes (for example the one which leads to the bigger receptive fields) are retained, the other being deleted.
4. Step 2 is repeated until N' branches have reached their end, or the depth of the tree has exceeded a certain fixed value.

(in this algorithm, M is the maximal length of contextual and relabelling rules string, N is the maximal number of branches that are allowed to live at a certain depth, N' the maximal number of branches to consider and K the maximal number of sons to consider for any node)

4.2 Weights' Correction

Once a partial reduction tree has been built, we will use it to modify the priorities of the rules that were involved. For each branch, we consider the graph corresponding to the leaf of the branch, and a distance between that graph and the one from the ground truth is computed (*e.g.* a graph edit distance). Depending on this distance, we modify the weights of the rules corresponding to every hypothesis present in the branch. If some rule is involved in many branches, the better the branch, the more its influence is taken into account.

A possible extension would be to determine which hypothesis leads to some graph incompatible with the ground truth in order not to consider the following hypotheses in the reduction string, or to give them higher penalty.

5 Conclusion

The representation we choose to use is the reduction strings. This is because during the learning process, we need to keep track of all rules that were used when the pyramid was built, which means that we need a lot of extra information beside the pyramid itself. Since we found that this extra information is indeed redundant with the pyramid, we choose to keep only it as a representation for the whole pyramid. Reduction string is a simple way to represent it, as they give the contraction kernels that are needed to build any level of the pyramid. This means that we can still build a pyramid from the chains if we need to consider all the graphs, for example after the learning process is finished.

We then showed that the space of all possible reduction string can be represented by a tree, the reduction tree, that can be used as the search space when we do learning. We also found out that it is possible to both extend the expressiveness of the tree and make it more easy to built at the same time, provided that we use the basic rules we proposed to simplify the local exploration. Also, the set of possible basic rules is limited, so it is theoretically possible to consider all of them, and even though their number is too high to really do that, we can still define some process to make the considered rules change dynamically.

The learning process itself consists in finding the best element from the search space (the set of all possible reduction strings, represented by the leaves of the reduction tree). Unfortunately, this need to build the whole reduction tree, whose size grows exponentially. So we need some search algorithm like the one that was described in section 4. It builds a limited size subtree of the whole reduction tree, using the current weights of the rules to choose which branches to look at.

One thing now to consider is the number of rules and symbols (especially temporary symbols needed to emulate the complex rules with the basic ones) that are needed for solving some simple problems, like digit recognition. This is

especially important, as the number of rules is directly linked to the whole complexity of the building process, and to the speed of convergence of the learning process which has to be evaluated. Finally, it would also be interesting to look at how to add or remove rules from the list of considered ones.

References

1. E. Duchesnay. *Agents situés dans l'image organisés en pyramide irrégulière, contribution à la segmentation par une approche d'agrégation coopérative et adaptative.* PhD thesis, Laboratoire de Traitement du Signal et de l'Image, University of Rennes I, France, 2001.
2. E. Hancock and M. Vento, editors. 4^{th} *IAPR International Workshop on Graph-Based Representations in Pattern Recognition*, volume LNCS 2726 of *Lecture Notes in Computer Science*, York, UK, June/July 2003. Springer-Verlag.
3. JM. Jolion and A. Montanvert. The adaptive pyramid : a framework for 2D image analysis. *Computer Vision, Graphics, and Image Processing : Image Understanding*, 55(3):339–248, 1992.
4. W.G. Kropatsch. Building irregular pyramids by dual graph contraction. *IEEE Proc. Vision, Image and Signal Processing*, 142(6):366–374, 1995.
5. P. Meer. Stochastic image pyramids. *Computer Vision, Graphics, and Image Processing*, 45(3):269–294, 1989.
6. M.Melki and J.M. Jolion. Building of symbolic hierarchical graphs for feature extraction. In E. Hancock and M. Vento, editors, 4^{th} *IAPR International Workshop on Graph-Based Representations in Pattern Recognition*, pages 44–58. Springer Verlag, 2003.

Representing and Segmenting 2D Images by Means of Planar Maps with Discrete Embeddings: From Model to Applications

Achille Braquelaire

LaBRI, Laboratoire Bordelais de Recherche en Informatique,
UMR 5800, Université Bordeaux 1, 351,
cours de la Libération, 33405 Talence, France
achille.braquelaire@labri.fr

Abstract. Representing the regions of a segmented image is an important aspect of image segmentation. Several different models have been proposed to represent the regions of a segmented image but most of them are dedicated to a specific method. Among the non hierarchical models, the model of planar maps with discrete embedding is certainly the most versatile one. Maps have the great advantage to provide a continuity of representation from the abstract mathematical model to the concrete implementation. They encode and provide most of topological and geometrical features required by segmentation algorithms and can be efficiently updated. In this paper we give an overview of the use of planar maps with discrete embedding in the context of image segmentation and we show how to design, implement and use a general environment for 2D image segmentation, from the mathematical model up to a real application. The model, data structure, algorithms and API described in this paper are currently implemented in a software which will be available under LGPL in the course of year 2005.

Keywords: Image representation, image segmentation, combinatorial maps, discrete boundaries, feature extraction.

1 Introduction

Representing the regions of a segmented image is an important aspect of image segmentation. On the one hand the representation provides the features used to build the decomposition of the image into homogeneous regions. Thus the description of the decomposition must be powerful enough to allow to extract any required feature. On the other hand segmentation process generally build solutions by refining progressively the decomposition. Thus the model used to described regions must be efficiently updated all along the segmentation steps. The simplest way to get an efficient model is to specialize it according to the minimal set of operations required by a specific method. But such a model is shown to be too restrictive to implement more general methods. Efficiency and

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 92–121, 2005.

versatility are thus two objectives which are hard to conciliate and when many models have been proposed to represent the regions of a segmented image, most of them are dedicated to a specific method.

The most basic method to represent segmented images is the array of labels [38, 76, 65] which consists in associating each pixel with a label such that all the pixels sharing a same label belong to a same region. This structure is very simple to implement but is ill-adapted to region splitting which involves a relabeling of all the pixels of the new sub-regions. Moreover this model does not efficiently provide topological features.

Hierarchical data structures allows to process images at different level of resolution. Several hierarchical models have been proposed such as quadtrees [46, 37, 72, 2], pyramids [77, 11, 82], irregular pyramids [61, 63, 57], linked pyramids [67, 48], dual pyramids [56], and more recently combinatorial pyramids [27]. Such data structures are very efficient to implement top-down region based algorithms. Starting at a coarse level of resolution, the initial image partition can be refined from level to level until reaching the resolution level of the original image. Nevertheless the hierarchical organisation of data restricts the possibilities of merging (and thus of interactive editing) and features like boundary geometry or neighbourhood are not immediate to extract.

Among the non hierarchical models, the model of embedded planar maps, or topological maps with discrete embedding [18, 49, 36, 39], is certainly the most versatile one. Embedded maps encode both the geometry and the topology of the regions of a segmented image and allow free region editing, splitting, and merging. They are more general than region adjacency graphs [47, 74] which have no geometric embeddings, which does not encode the whole topology of the segmented image, and which are ill-adapted to splitting. Two dimensional planar maps have been used for image editing [18, 4, 43, 19, 36], for image segmentation [49, 39, 23, 40, 14, 53] and for video processing [5, 6, 55]. Maps have been generalized in the context of topological based scene modeling and in order to represent n-dimensional spaces [58], and several recent works have addressed the problem of the representation of 3D discrete images with maps for 3D image segmentation [16, 7, 33, 34, 12].

In the context of 2D image analysis, embedded maps provide an efficient framework to implement most of the operations involved by segmentation algorithms [25], such as domain reconstruction (or restoration) [71] which consists in traversing each point of the region domain, region splitting and merging, point inclusion or point membership property [60, 30] which consists in determining if a point is located inside or outside a given region, region localisation which consists in finding the region containing a given point, obtaining of geometric features (such as area, perimeter, boundary shape, etc.) [70, 30, 68] and of topological features (neighbourhood, surroundness [70], regions inside or outside a given boundary, counting holes [64, 69], etc.).

In this paper we give an overview of the use of planar maps with discrete embedding in the context of image segmentation. Maps have the great advantage to provide a continuity of representation from the abstract mathematical model

to the concrete implementation. We would like to illustrate that by showing how to design, implement and use a general environment for 2D image segmentation, from the mathematical model up to a real application. In the following section we recall how to represent the topology of a continuous segmented image with a set of planar maps. In section 3 we show how to associate such a set of maps with discrete boundaries and thus how to use this model to represent and segment discrete images. In sections 4 and 5 we briefly describe data structures and algorithms to use this model in the context of image segmentation. In section 6 we specify a small API that is enough to implement most of split and merge segmentation methods. Finally in section 7 we describe a full example of constrained segmentation method designed with this API to solve a real problem in the context of medical imaging.

2 Representing Regions

A segmented image is a partition of an image into a set of regions, each region being a connected subset of points of the image. In the Euclidean plane a region is *simply connected* when its boundary is a simple closed curve also called Jordan's curve.[1] Remark that a Jordan's curve defines two regions: a bounded region without hole which is called the inside region, and its complement which is an unbounded region with a hole which is called the outside region. When a bounded region is not simply connected, it has some holes and its boundary consists of several Jordan's curves, one of them — the outer boundary — surrounding all the other ones — the inner boundary.

Consider the example developed in Fig 1. The image of Fig 1-a can be segmented into seven regions each one being a homogeneously textured area. The region boundaries have been drawn in white on the image. The wide region located in the bottom of the image, say A, is a simply connected region and its boundary consists of a unique Jordan's curve. On the other hand the background is a bounded region with two holes and thus is not simply connected. Its outer boundary is the boundary of the image and its inner boundary consists of two Jordan's curves, one of which being the outer boundary of the region A and the other one being the largest oval-shaped contour.

In the Euclidean plane the boundaries of the regions of a segmented image can be partitioned in a natural way according to their neighbouring. Consider for instance the both small half-oval shaped regions of the running example. Both these regions share a part of boundary which is an horizontal line. Thus the boundary of these regions can be split into three *segments* of boundary which are respectively the horizontal line shared by the both regions, the part of boundary which is adjacent to the upper region and not to the lower one, and the part which is adjacent to the lower one and not to the upper one. Each segment of boundary is a Jordan's arc (a set of points homeomorphic to the real interval $[0, 1]$). This decomposition induces a graph the edges of which correspond to the

[1] A Jordan's curve is a set of points homeomorphic to the real interval [0, 1].

Jordan's arcs and the vertices to the segment junctions. The graph induced by the boundaries of the segmented image of Fig 1-a is the graph shown in Fig 1-b. This graph has height vertices labeled from a to h and eleven edges. Remark that it is necessary to add an arbitrary vertex to associate with a graph edge the outer boundary of an isolated region. For instance the vertices labelled by a and h have been arbitrarily added on respectively the outer boundary of the image and the outer boundary of the region A.

A partition of the Euclidean plane into simply connected region is called a *topological planar map*, or simply a *planar map*. More formally a planar map [78] is a decomposition of the Euclidean plane into a finite set V of points, a finite set E of disconnected open Jordan's curves, each one having its extremities in V, and a finite set of simply connected regions whose boundaries are unions of elements of V and E. The elements of V and E are respectively the vertices and the edges of the map, and each simply-connected region is called a *face*. The faces corresponding to bounded regions are called the *finite faces* and the face corresponding to the unique unbounded region is called the *infinite face*.

Each connected component of the boundary graph of a segmented image is thus a planar map. For instance on the running example the boundary graph is decomposed into four planar maps which are the subgraphs defined respectively on the set of nodes $\{a\}$, $\{b, c\}$, $\{d, e, f, g\}$, and $\{h\}$. The second one has two vertices, three edges (labeled by 8, 9, and 10) and three faces, two finite faces which are the face surrounded by the sequence of edges $(8, 9)$ and $(8, 10)$, and an infinite face the boundary of which is the sequence $(9, 10)$.

A topological map can be efficiently encoded by a pair $\langle \sigma, \alpha \rangle$ of permutations defined on a set of labels called *darts*. Each dart can be seen as an half-edge of the topological map. Given an orientation of the plane, say counterclockwise, a vertex v of the map is describe by a circular sequence $(d_1, d_2, \ldots, d_n)$ which is the sequence of darts reaching it. This sequence is a cycle of the permutation σ and the notation $(d_1, d_2, \ldots, d_n)$ is a shortcut for $\sigma(d_1) = d_2, \ldots, \sigma(d_{n-1}) = d_n$, and $\sigma(d_n) = d_1$. The permutation σ is the set of all such cycles. The permutation α is an involution without fixed point (each cycle is of length 2). Each cycle $(d, \alpha(d))$ of α encodes an edge of the map by linking two darts. Such a representation is called a *combinatorial map* [32].

If π is a permutation and x is an element which has an image by π, we denote by $\pi^*(x)$ the cycle of π that contains x. According to this notation, $\sigma^*(d)$ (resp. $\alpha^*(d)$) is the vertex (resp. the edge) that contains the dart d.

It may be convenient to encode the darts by positive and negative integers such that $\alpha(d) = -d$ [19]. According to this convention, a representation of the four combinatorial maps of the running example is shown in Fig 1-c. The set of darts is the set $\{-11, \ldots, -1, 0, 1, \ldots, 11\}$. The related permutations are:

- $\sigma_1 = (-1, 1)$ and $\alpha_1 = (-1, 1)$;
- $\sigma_2 = (8, 9, -10)(-8, 10, -9)$ and $\alpha_2 = (-8, 8)(-9, 9)(-10, 10)$;
- $\sigma_3 = (-2, 4, -7)(3, -5, -4)(5, 6, 7)(2, -6, -3)$ and $\alpha_3 = (-2, 2)(-3, 3)(-4, 4)(-5, 5)(-6, 6)(-7, 7)$;
- $\sigma_4 = (11, -11)$ and $\alpha_4 = (-11, 11)$.

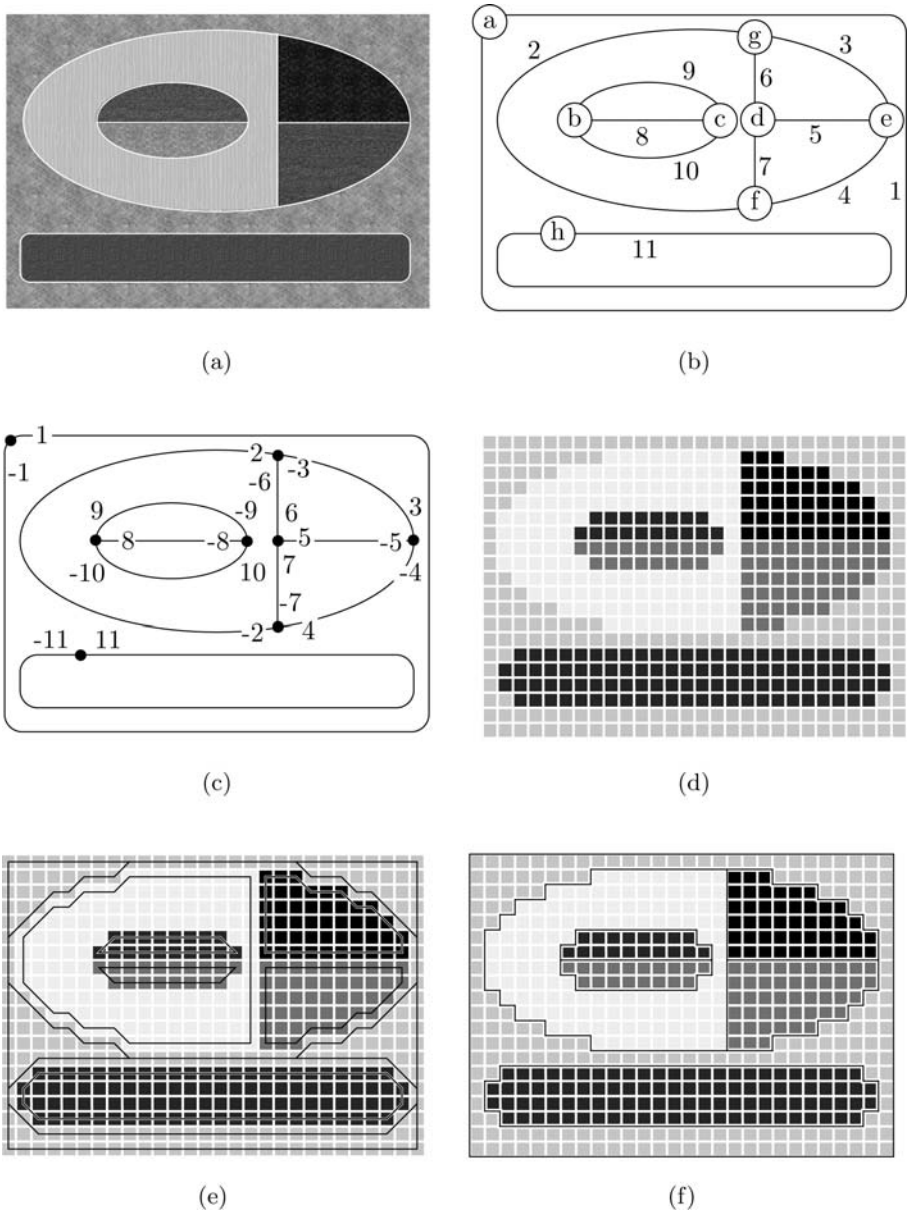


Fig. 1. Representation of the topology and of the geometry of a segmented image: (a) a segmented continuous image with its region boundaries, (b) the associated boundary graph, (c) the representation of this graph by combinatorial maps, (d) a discrete segmented image with the same topology, a representation of the geometry of regions (e) with pixel based contours, and (f) with interpixel contours

A face the map is encoded by the circular sequence of darts encountered when turning around the face, clockwise for a finite face and counterclockwise for the infinite one. For instance the map $\langle \sigma_3, \alpha_3 \rangle$ has four faces which are the cycles (-2, -6, 7), (5-4, -7), (-5, 6, -3) and $(4, 3, 2)$ (see also Fig. 2-a). The infinite face (in this case the last one) represents the unbounded region of the Euclidean plane which is the complement of the union of all the finite regions.

A given combinatorial map $\langle \sigma, \alpha \rangle$ may be associated with several different topological maps. That means that there are different ways to draw a combinatorial map on the plane. In fact there as many topological maps as there are faces in the combinatorial map. To draw a combinatorial map consists thus in first deciding which face is the infinite face and then to organize vertices and edges according to this choice. Once the infinite face is set, all the possible drawings are topologically equivalent.

A remarkable property of combinatorial maps is that the faces of the map $\langle \sigma, \alpha \rangle$ are encoded by the cycles of the permutation $\varphi = \sigma \circ \alpha$. Consider for instance a dart of the map $\langle \sigma_3, \alpha_3 \rangle$, say the dart -2. The cycle of -2 in the permutation φ is the circular sequence $(\sigma \circ \alpha)^*(-2)$. We have $\sigma(\alpha(-2)) = -6$, $\sigma(\alpha(-6)) = 7$, and $\sigma(\alpha(-7)) = -2$. The cycle $\varphi^*(-2)$ is thus the finite face (-2, -6, 7).

The permutations φ and σ are defined on the same set of darts and the tuple $\langle \varphi, \alpha \rangle$ is also a combinatorial map. Moreover the maps $\langle \sigma, \alpha \rangle$ and $\langle \varphi, \alpha \rangle$ are dual. The orientation apart the dual map can be drawn by associating with each face of the primal map a vertex of the dual map, and by intersecting each edge of the primal map by an edge of the dual map, both edges being defined by the same pair of darts. For instance, on the running example, the dual map of $\langle \sigma_3, \alpha_3 \rangle$ is the map $\langle \phi_3, \alpha_3 \rangle$ with $\phi_3 = (-2, -6, -7)\ (5, -4, -7)(-5, 6, -3)(4, 3, 2)$ and $\alpha_3 = (-2, 2)(-3, 3)(-4, 4)(-5, 5)(-6, 6)(-7, 7)$ (see Fig. 2-b). Let us underline that a dual map encodes the adjacency of the regions of a segmented image in a more general way than a region adjacency graph does, since there is in the dual map an edge for each segment of boundary shared by two adjacent regions.

Combinatorial maps are a very simple and elegant formalism to describe both planar maps and operations defined on them. For instance, on the running example, removing the edge $(-5, 5)$ consists in merging the face containing the dart 5 which the one containing the dart -5, i.e. the faces $(-5, 6, -3)$ and $(-7, 5, -4)$. This operation may be defined either on the map $\langle \sigma, \alpha \rangle$, by setting $\sigma'(3) = -4$ and $\sigma'(7) = 6$, or on the map $\langle \varphi, \alpha \rangle$, by replacing both cycles $(-5, 6, -3)$ and (5, -4, -7) by the cycle (6, -3, -4, -7) (see for instance [19, 36, 23, 24, 26] for a formal definition of these operations).

Remark that neither the vertices nor the edges need to be explicitly encoded. Each vertex, edge or face may be represented by any dart of the permutation cycle that represents it. Moreover by taking $\alpha(d) = -d$ the permutation α has not to be stored. So a combinatorial map, and thus a topological map, can be implemented with only an array of integers the size of which is twice the number of edges of the map.

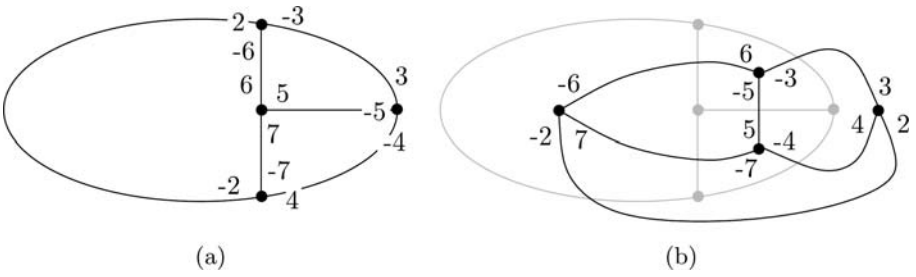


Fig. 2. The map $\langle \varphi, \alpha \rangle$ of Fig. b is obtained by defining a vertex for each face of the map $\langle \sigma, \alpha \rangle$ of Fig. a and by crossing each edge of the map $\langle \sigma, \alpha \rangle$ by an edge defined by the same pair of darts. The resulting graph is a representation of the map $\langle \varphi, \alpha \rangle$ according to an inverse orientation (or of the map $\langle \varphi^{-1}, \alpha \rangle$)

It is noticeable that the same construction provides both a mathematical tool for formal proof and an efficient data structure for implementation purpose. Finally we may chose to implement either the primal representation or the dual one, each of then being obtainable from the other one with a negligible computational overhead.

Since a map encode the topology of only one connected component of the boundary graph, there are as many maps associated with the boundary graph as there are connected components. It is thus necessary to encode how these maps contribute to the representation of the topology of a non simply connected region of the segmented image.

It is of course possible to define an inclusion tree which nodes are the regions of the segmented image [64, 44]. Nevertheless it is enough to encode the inclusion for only the infinite faces. It may be done in a straightforward way by adding to the model an *inclusion relation* which associates the finite face corresponding to the outer boundary of each non simply connected region with the list of the infinite faces corresponding to the outer boundary of the holes this region contains. The finite face is called a *parent face* and the infinite ones the associated *children faces*. For instance the inclusion relation of the four maps of Fig 1-c may be described by the relation: $(1, -11), (1, 4), (-6\ -10)$.

Of course this encoding is not unique since any dart of a face cycle may be used to define the relation. We shall see in the next section that it is convenient to associate a label which each face. The relation of inclusion will then be defined in a unique way according to this labeling. Note that operations on maps may also modify this relation [36, 23, 26].

3 Representing Discrete Regions

In order to use combinatorial maps with discrete images it is necessary to get a representation of a discrete segmented image that can be associated with a set of planar maps. It requires the decomposition of a segmented image into discrete correspondents of vertices, edges and faces.

Discrete boundaries can be defined either in the image domain as pixel based contours [42, 62, 66, 71, 30] or in a discrete space different of the image space as interpixel contours [22, 71, 52, 54, 10, 50, 17]. Pixel based contours have several drawbacks when used to represent the boundaries of a segmented image. For instance if boundary segments are part of image regions a given boundary segment belongs to only one of its neighbouring regions, and if boundary segments are not part of image regions, the set of regions does not define a partition of the image. On the other hand, the interpixel representation provides a consistent topological framework [1] and makes it possible to define in a natural way discrete analogous of the edges of the boundary graph [17, 36]. On the running example of Fig 1, a discrete discrete segmented image is displayed in Fig 1-d. The pixels are represented as colored unit square and the regions are the maximal 4-connected component. The image of Fig 1-e shows an example of boundaries defined with pixel contours and the one of Fig 1-f an example of interpixel boundaries which is compatible with the boundary graph of Fig 1-b.

Intuitively interpixel boundaries are drawn *between* pixels. If the image plane is the discrete space $\mathbb{Z} \times \mathbb{Z}$, the boundary plane is the *half-integer* plane which is obtained by translating the discrete plane $\mathbb{Z} \times \mathbb{Z}$ by $(-\frac{1}{2}, -\frac{1}{2})$ [17]. A point $p = (x_p, y_p)$ of the image plane and a point $p' = (x'_p, y'_p)$ of the boundary plane are *half-neighbours* if $|x_p - x'_p| = |y_p - y'_p| = \frac{1}{2}$. Each point of the image plane has four half-neighbours in the boundary plane and each point of the boundary plane has four half-neighbours in the image plane. Finally two adjacent points of the boundary plane share exactly two half-neighbours in the image plane.

Each point of the boundary plane having two or more half-neighbouring points belonging to different regions of a segmented image is a *boundary point*. Two adjacent boundary points are *linked* if their common half-neighbours belong to different regions, and the *rank* of a boundary point is the number of boundary points linked to it. Two boundary points b and b' are *connected* if there is a boundary path (or *contour*) linking them [20].

The boundary ∂r of a region r is the set of boundary points adjacent to both a point inside r and a point outside r. It consists of four-connected closed paths of boundary points. Each path is a sequence of boundary points $b_1, b_2, \ldots, b_n$ with $n > 1$ and such that b_i is linked to $b_i + 1 \quad \forall i$ with $1 \leq i < n$, and $b_i \neq b_j$, $\forall i, j$ with $i \neq j$. It can be shown that such boundaries are discrete Jordan's curves by embedding both the boundary and the image planes into the Khalimsky's plane [50, 51].

According to these definitions the boundary of a discrete segmented image can be decomposed into *segments* and *nodes* in the same way than the boundary graph of a continuous segmented image can be decomposed into edges and vertices. The nodes are either *natural nodes* or *arbitrary nodes*. Natural nodes are the boundary points of rank greater than two. Arbitrary nodes are points arbitrary selected on each boundary component consisting of a unique closed contour (one arbitrary node selected for each closed contour). A *segment* is then a maximal contour without node. On the example of Fig. 3, the boundary points are displayed with disks. The nodes are boundary points displayed with grey disks.

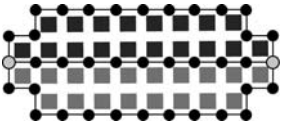


Fig. 3. Example of interpixel boundary defined in the half-integer plane

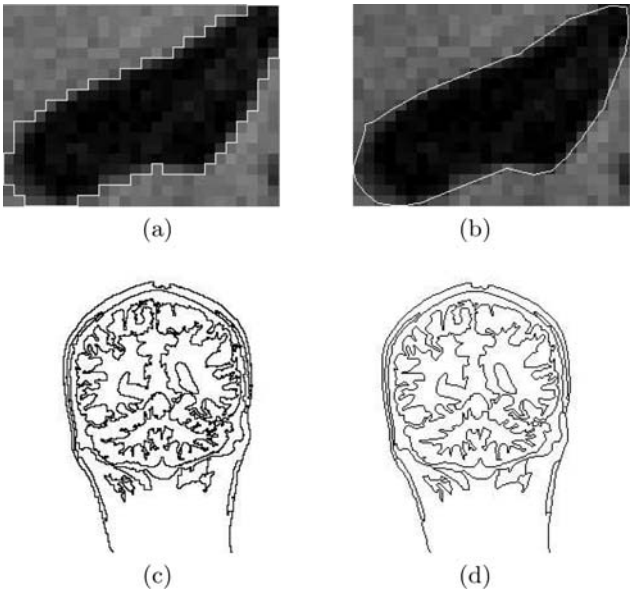


Fig. 4. Examples of geometric smoothing of interpixel boundary

There are two nodes, each one of rank 3, and three segments. On this example both nodes are natural nodes.

Now a segmented image associated with a boundary plane can be partitioned into a set of nodes, a set of discrete Jordan's curves joining two nodes, and a set of 4-connected regions. If all regions are simply connected, such a structure is the discrete analogous of a topological map and is called a topological map with discrete embedding, or a discrete map. When some regions are not simply connected, there are several discrete maps associated by an inclusion relation as there are several topological maps in the Euclidean case.

Like topological maps, discrete maps can be described by pairs of permutations. The analogous of a dart of a combinatorial map is a *geometrical dart*. Each geometrical dart is associated with an end of segment. Is p is a boundary point at the end of a segment and n the node linked to p the associated geometrical dart is the pair (p, δ) where δ is the elementary direction from n to p. By this way a discrete map induces a combinatorial map defined as follows:

- each geometrical dart of the discrete map is associated with a dart of the combinatorial map;
- each pair of dart associated with the two geometrical darts of a same segment forms a cycle of the permutation α;
- each sequence of darts associated with a sequence of geometrical darts sharing a same node forms a cycle of the permutation σ, according to the order of the geometrical darts around the node [20].

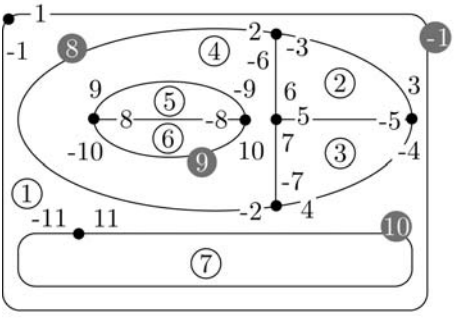
It is thus possible to represent discrete segmented images by associating a usual topological representation with a discrete geometrical one. The main drawback of this model is the poor visual aspect of the boundaries which are defined as 4-connected path. This problem has been solved by introducing a reversible geometric smoothing called Euclidean paths [81, 79, 21] in which each point of the path is moved inside its unit cell by projecting it onto a local discrete tangent. The resulting path is a smoothed path the sampling of which gives back exactly the starting discrete path. Enlargements of a discrete interpixel boundary and of its associated Euclidean path are displayed in Fig. 4-a and Fig. 4-b. It is possible to smooth by this way all region inter-pixel boundaries of a segmented image and it can be shown that the topology of the boundary is not altered by this smoothing [15]. The image of Fig. 4-d is an example, displayed at scale 1, of reversible geometric smoothing of the 4-connected interpixel boundaries displayed in Fig. 4-c.

4 From Model to Data Structure

The next step to get an image segmentation environment is now to define a minimal data structure both to encode the models previously described and to perform feature extraction and representation updating involved by segmentation algorithms.

Topological Data Structure. As stressed above the data structure used to represent a map is simply an array of integers indexed by darts. This array may encode either σ or ϕ. We have chosen to encode the second one. Moreover, all the maps involved in the representation of the topology of a segmented image are disjointed in the sense that each map $\langle \varphi_i, \alpha_i \rangle$ is defined on a set of darts D_i disjointed from the other ones. We define the permutation φ by $\varphi = \cup \varphi_i$ and the permutation α by $\alpha = \cup \alpha_i$. We have $\varphi_{|D_i} = \varphi_i$ and $\alpha_{|D_i} = \alpha_i$, and we may thus simply denote by $\langle \varphi, \alpha \rangle$ the set of maps $\{\langle \varphi_i, \alpha_i \rangle\}$. In the same way σ denotes the union of all the σ_i permutations. The permutation φ is encoded by an array `phi` of integers, and for any dart d, if D_i is the set of darts that contains d, we have `phi[d]` $= \varphi_i(d)$, and `phi[-d]` $= \sigma_i(d)$.

In order both to address regions and to define the inclusion relation it is convenient to define a *region labeling*. Therefore each dart is labelled with a labeling function λ such that two darts have the same label by λ if and only if they belongs to the same cycle of φ. Thus for each pair of darts (d, d') we have: $\lambda(d) = \lambda(d') \Leftrightarrow \varphi^*(d) = \varphi^*(d')$.



d	$\lambda(d)$	d	$\lambda(d)$
-11	10	1	1
-10	9	2	8
-9	9	3	8
-8	5	4	8
-7	3	5	3
-6	4	6	2
-5	2	7	4
-4	3	8	6
-3	2	9	5
-2	4	10	6
-1	-1	11	7

f	$\beta(f)$
-1	-1
1	1
2	-3
3	5
4	7
5	-8
6	8
7	11
8	4
9	-9
10	-11

Fig. 5. A face labeling of the running example. The label in gray are labels of infinite faces

Conversely, a function β associates each face label f with a dart of the corresponding face cycle such that $\lambda \circ \beta(f) = f$, and for each dart d, $\beta \circ \lambda(d) \in \varphi^*(d)$. This dart, called the *canonical dart* of f, is an arbitrary entry point in the dart face cycle. The face of label f is thus the cycle $\varphi^*(\beta(f))$, and for each face label f we have: $\forall d \in \varphi^*(\beta(f)),\ \lambda(d) = f$. Both the function λ and the function β are encoded by an array of integers (respectively the array `lambda` and the array `beta`). A labeling of the maps of the running example is given in Fig 5.

For each face f associated with a holed region, the relation ***children*** gives the list of infinite faces associated with the holes. Conversely the relation ***parent*** gives for each infinite face f the finite face f' such that $f \in$ ***children***(f'). The relations ***parent*** and ***children*** are the inclusion relations. Remark that each region is associated which only one finite face and possibly several infinite faces. Each region may thus be labeled in a unique way by the label of its associated finite face. For instance the background region of the segmented image of the running example is labeled by 1 which is also the label of its associated finite face. The other faces associated with this region are the two infinite faces labeled respectively by 8 and 10. The inclusion relations of the running example are given in Fig 6. Two regions r and r' may be neighbouring regions because their associated faces are adjacent in one of the dual maps: $\exists d \in \varphi^*(\beta(r))$, and $\exists d' \in \varphi^*(\beta(r'))$ such that $\alpha(d) = d'$. But two regions may also be neighbouring regions according to the ***parent*** (or ***children***) relation. That leads us to consider three different neighbouring modes on which we shall come back in section 6.

f	***parent***(f)
8	1
9	4
10	1

f	***children***(f)
1	$\{8, 10\}$
4	$\{9\}$

Fig. 6. The inclusion relation of the running example

Geometrical Data Structure. We have now to define a data structure to represent the geometrical embedding of maps, i.e. the geometry of interpixel boundaries. It is possible to associate each segment with a local embedding. This embedding can be defined independently for each segment, either explicitly by a sequence of elementary steps or implicitly in a procedural way. Another solution consists in defining a global embedding of the whole boundaries by encoding the whole part of the boundary plane corresponding to the image domain. The main advantage of the first solution is to provide simplest updating, especially for segment removing. However the local encoding is ill-adapted to the splitting of a region and to geometrical editing of regions. The solution retained here is thus to encode explicitly the boundary plane by using a global data structure called boundary image.

The boundary plane and the image plane are isomorphic. Thus an image and its boundary image have about the same number of elements (in fact when an image is of size $N \times M$, its boundary image is of size $(N+1) \times (M+1)$ in order to encode the image outer boundary). The boundary image encodes both boundary points and link. A boundary point may have from two to four links. An element of the boundary image without links cannot be a boundary point and conversely an element of the boundary image with links is a boundary point. Thus it is only necessary to explicitly encode links. Moreover the linking relation is a symmetrical relation. Thus it is only necessary to encode links along two of the four possible directions. For instance if we chose to encode upward and rightward links it is possible to know if a boundary point p' is linked downward to a boundary point p by checking if its downward neighbour p is linked upward to it. Since there is no way to recognize an arbitrary node it is also necessary to mark nodes in the boundary image. Thus only three bits are needed to store each entry of the boundary image.

To sum up, if the domain of an image is the set of points $\{(i,j) \in \mathbb{Z}^2,\ 0 \leq i < H,\ 0 \leq j < W\}$ the associated boundary domain is the set $\{(i-\frac{1}{2},\ j-\frac{1}{2}),\ (i,j) \in \mathbb{Z}^2,\ 0 \leq i \leq H,\ 0 \leq j \leq W\}$. The boundary domain is encoded by an array `B` called *boundary image* where the entry `B[i][j]` encodes three boolean informations: whether the boundary point p of coordinates $(i-\frac{1}{2}, j-\frac{1}{2})$ is a node, whether p is linked upward to either another boundary point or to a node, and whether p is linked rightward to either another boundary point or to a node. One can retrieve the geometry of a segment by following links from one of its geometrical darts to the other one.

Correspondence Between Topology and Geometry. The correspondence between topological and geometrical data structures is encoded by a associating combinatorial and geometrical darts. A geometrical dart is encoded by a point (which is a pair of coordinates) and a direction (upward, leftward, downward or rightward). The relation between geometrical and topological darts is stored in an array of geometrical darts indexed by topological darts. A hash table with pairs of node coordinates as keys is used to avoid to traverse this array when looking for a geometrical dart and get an efficient access to the topological representation from the geometrical one.

5 Overview of Algorithms

We have described in the previous section the whole data structure used to implement a topological and geometrical representation of a segmented image. Let us now give a short overview of related algorithms.

It is possible to find the region that contains a point by scanning the boundary image from this point until reaching a segment. If the boundary image is scanned horizontally (for instance rightward) it is enough to look for the first encountered vertical link. Once the segment is reached the traversal continue by following this segment until reaching a geometrical dart. The associated topological dart d identify a face which is an infinite face if the scanning has reached the outer boundary of a connected component and a finite face in the other case. The region is then given by the label $\lambda(d)$ in the case of a finite face and by the label $\boldsymbol{parent}(\lambda(d))$ in the case of an infinite face.

The boundary of a region f is obtained by traversing the cycle $\phi^*(\beta(f))$ and the cycles $\phi^*(\beta(f'_i))$ of the faces f'_i of $\boldsymbol{children}(f)$. The geometry of this boundary is obtained by traversing the segments corresponding to the geometrical darts associated with the topological darts of $\phi^*(\beta(f)) \cup \bigcup_i \{\phi^*(\beta(f'_i))\}$.

According to the orientation of the plane the finite faces are traversed clockwise and the infinite ones counterclockwise. The domain of a region can thus be reconstructed by building the list of all image points that are on the left of an upward link or on the right of a downward link (see Fig. 7) and by sorting this list relatively to lines and then to columns. The resulting list is exactly the list of horizontal lines covering the region domain.

Fig. 7. Reconstruction of the domain of a region. The dark disks represent boundary points and the white squares image points

The construction of the representation of a segmented region r can be done with a complexity $K \times |r|$ where $|r|$ is the number of points of the region, and where K is a constant equal to 7 in the worst case. The topological updates involved by split and merge can be expressed by mean of elementary operations on the permutations and on the inclusion relations. The cost of geometrical updating involved by splitting and merging is $O(\sum_{s \in S} |s|)$ where S is the set of inserted or removed segments. All these algorithms have been described in detail in previous works [19, 36, 23, 24, 20, 26].

6 Designing a Toger API

In order to validate the interest of the model of planar maps with discrete embedding in the contex of image segmentation, we describe in this section how to

interact with such an environment, that we call *toger kernel* in the following of this paper. We first give a short description of the types of objects manipulated by such an API and then of the main functions of the interface.

Types. It is convenient to provide both a high level and a low level of interaction. At the high level (or region level) the objects returned by the functions of the API are regions or lists of regions, *paths* or lists of paths which encode the geometry of region boundaries, and *domains* which encode the geometry of regions. A region is a label encoded by an integer. A path is a sequence of adjacent points that can be encoded by an array or in a more suitable way by a generator, i.e. a set of functions { `first`, `last`, `next`, `previous`, `length` }, that is a usual interface to traverse the elements of a list. The domain of a region r is encoded by a sequence of pairs of points. A point is simply a pair of integer coordinates. Each pair defines a *span* of the region r which is an horizontal maximal line (P_{2i}, P_{2i+1}) belonging to r. The point P_{2i} (resp. P_{2i+1}) is thus located on the right (resp. the left) of a vertical interpixel boundary element of r. The list of pairs of points is the list of all the spans of the domain of r. Finally we have seen that two adjacent regions can be related according to three different neighbouring modes. The type `neighbouring_mode` is used to denote these modes. The high level of interaction does not require any knowledge of the internal representation. Conversely the low level (or map level) provides interactions directly with the topological maps. The data types used at map level are the darts, the face labels and the geometrical darts. As seen above, both darts and face labels are elementary types encoded by integers. We have seen that a region label is the label of its finite face. Thus region labels are a subset of face labels. The geometrical darts are pairs consisting of a point and a direction. Finally, the type `toger` is used to refer to the whole representation. All these types are summarized in Table 1.

Table 1. Types of the toger API

dart	set of integers
face	set of integers
region	set of integers
neighbouring_mode	{ *DIRECT*, *INNER*, *OUTER*, *ANY* }
point	pair of coordinates
direction	{ *UPRIGHT*, *LEFTWARD*, *DOWNWARD*, *RIGHTWARD* }
geometrical_dart	point and direction
path	generator of points
domain	list of pairs of points
toger	topological and geometrical representation

Side Effect Functions. Among the side effect functions (i.e. functions that modify the representation they receive in parameter) we need a function that builds the representation of a segmented region: *split_region* : *toger* × *region* × (*point* × *point* → *boolean*) → *list of regions*. The sub-regions of the segmented region are implicitly described by a partitioning function f : *point*×*point*→*boolean* which

receives two neighbouring points in parameter and returns *true* if these points belong to a same region and *false* if not. The function *split_region* updates the topological and the geometrical representation of a region (second parameter) according to a representation (first parameter) and to a partitioning function (third parameter); the result is the list of labels of the new sub-regions.

In order to remove a boundary shared by two adjacent regions we need a function that merges these regions: *merge_regions* : *toger* × *region* × *region* → ∅. This function modifies a representation (first parameter) by merging two regions (second and third parameters). The label of the region resulting from the merge is the same as the one of the first region. Note that the removed part of boundary is not necessarily connected and that this operation may disconnect to components in the boundary graph. This function is the only low level deleting function that can be defined in such an API because other basic delete operations (like segment removing for instance) does not guarantee that the consistency of the representation is preserved. Higher level deleting functions can be considered like for instance the removing of all the region being inside a closed contour.

It may also be convenient to modify the representation by inserting a new contour. It can be done by a function like *insert_contour* : *toger* × *path* → *list of regions*. In order to preserve the consistency of the representation, an inserted contour must be either a closed contour or a contour joining two nodes [19]. Thus the contour to insert (second parameter) must be preprocessed by the function *insert_contour* before being inserted, in order to satisfy one of these conditions. All these functions modify both the geometry and the topology of the representation. It is also possible to consider functions that modify only the geometry, which can be for instance useful to locally smooth a boundary.

Point Inclusion and Region Localisation. We also need function to search for a region. We have shortly described how to retrieve the region containing a given point. This functionality can be provided by a function like *find_region* : *toger* × *point* → *region* which returns the label of the region containing a point (second parameter) according to a representation parameter). It is also possible to define a function *belongs_to* : *toger* × *point* × *region* → *boolean* that checks if a point parameter) belongs to a region (third parameter) according to the representation (first parameter),

Geometrical Features. The main geometrical features are the domain of a region which can be obtained by a function like `region_domain` : *toger* × *region* → *domain*, and the geometry of its boundary which can be obtained by a function like *region_boundary* : *toger* × *region* → *list of closed_path*. Since a boundary may consist of several closed path, it its convenient to get the outer boundary as the first element of the returned list. It may also be convenient to get only the outer boundary: *region_outer_boundary* : *toger* × *region* → *closed_path* and the part of boundary shared by two adjacent regions by using the function *regions_common_boundary* : *toger* × *region* × *region* → *list of paths*. Geometrical features of segment may also be useful. That leads us to consider functions like *segment* : *toger* × *dart* → *path*, that returns a path which describes the geometry of the segment associated with a dart, and *segment_length* : *toger* × *dart* →

integer, that returns the segment length. Several different length estimators can be used, such as the number of elementary steps or the length of the associated Euclidean path [80].

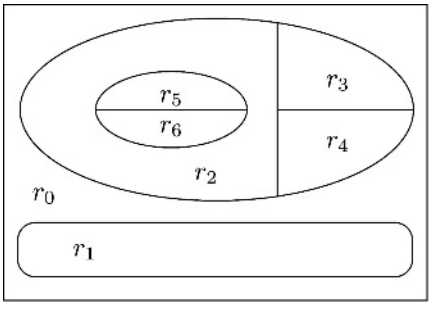
Low Level Topological Features. Defining the interaction with the low topological level is straightforward. Two functions *lambda* : *toger* × *dart* × *face* and *beta* : *toger* × *face* → *dart* implement the functions λ and β associating darts with labels. The functions *alpha* : *toger* × *dart* → *dart*, *sigma* : *toger* × *dart* → *dart*, and *phi* : *toger* × *dart* → *dart* implements the permutations α, σ, and φ used to traverse the combinatorial representation of maps. Finally the connected components can be traversed with function *parent* : *toger* × *face* → *face* that returns for a face f the face ***parent***(f) if f is infinite or a void value if f is finite, and with the both functions *first_child* : *toger* × *face* → *face* and *next_child* : *toger* × *face* → *face* that implement the relation ***children***.

Topological Marking. It is sometimes necessary to traverse the graph of regions with respect to some marking. By associating marks to dart it is possible to maintain the consistency of the marking when doing a side effect operation. When an edge (d_1, d_2) is split into two adjacent edges (d_1, d_2) and (d'_1, d'_2), the dart d'_i receives a copy of the marks of the dart d_i. By this way, when a region is split into subregions, it is possible without overhead to preserve the consistency of marks on faces and more generally on any contour. It is also possible to preserve the consistency of contour marking through region merging. When a region r' is merged with a region r, and if the common boundary of r and r' is a unique edge e, the marking can be preserved by marking the face associated with r before removing e. If there are several edges $e_1, ..., e_n$, the removing of these edges disconnects from one to $n-1$ components of the map. In this case it is also necessary to unmark some edges in these components. It is possible to mark an oriented edge, an non-oriented edge (by marking a cycle of α), a node (by marking a cycle of σ), or a face boundary (by marking a cycle of φ). The number of marks depends on the space allocated for each mark. By allocating one byte per dart we get height marks per dart which is enough for most of traversal algorithms and which the memory cost is reasonable with modern computers. The type *set of flags* is a boolean combination of flags used to manipulate the marks.

The interface of marking operations can be defined like the one of geometrical feature functions. The functions *mark_region_boundary* : *toger* × *region* × *set of flags* → ∅ and *unmark_region_boundary* : *toger* × *region* × *set of flags* → ∅ respectively marks and unmarks the boundary of a region. Similarly, the functions *mark_outer_boundary* : *toger* × *region* × *set of flags* → ∅ and *unmark_outer_boundary* : *toger* × *region* × *set of flags* → ∅ respectively marks and unmarks only the outer boundary of a region, and *mark_common_boundary* : *toger* × *region* × *region* × *set of flags* → ∅ and *unmark_common_boundary* : *toger* × *region* × *region* × *set of flags* → ∅ the part of boundary shared by two regions. Finally we need functions like *mark_all_darts* : *toger* × *set of flags* → ∅ and *unmark_all_darts* : *toger* × *set of flags* → ∅ to set and clear marks on all the darts, and a function

dart_is_marked : *toger* × *dart* × *set of flags* → *boolean* to check if a dart is marked. Each time, a parameter of type *set of flags* is used to specify which is the mark (or are the marks) to modify.

High Level Topological Functions. The neighbourhood of a region may be considered according to the three neighbouring modes. Both of the neighbouring modes, the *inner* and the *outer* ones, are oriented relations. The third one is a symmetric relation. A region r' is a direct neighbour of a region r if the intersection of their outer boundaries is not empty. A region r' is an inner neighbour of a region r if the outer boundary of r' intersects the internal boundary of r. In that case r is an outer neighbour of r'. In other terms a region r' is a direct neighbour of a region r if $\exists d \in \varphi^*(\beta(r))$, and $\exists d' \in \varphi^*(\beta(r'))$ such that $\alpha(d) = d'$. A region r' is an inner neighbour of a region r (or r is an outer neighbour of r') if $\exists d \in \varphi^*(\beta(r'))$ such that $\boldsymbol{parent}(\lambda(\alpha(d))) = r$. The neighbouring modes of the regions of the running example are given in Fig. 8.



	r_1	r_2	r_3	r_4	r_5	r_6	r_7
r_1		I	I	I			I
r_2	O		D	D			
r_3	O	D		D			
r_4	O	D	D		I	I	
r_5				O		D	
r_6				O	D		
r_7	O						

Fig. 8. Example of region neighbouring relations. The letters D, I and O denote respectively the direct, inner and outer neighbouring. For instance the second line of the array means that region r_1 has one outer neighbour (the region r_0), two direct neighbours (the regions r_2 and r_3), and two inner neighbours (the regions r_4 and r_5)

The neighbourhood of a region may also be considered according to the topological marking. The neighbouring relation is thus restricted such that two regions are not considered as neighbouring regions if their common boundary is marked, according to a given set of marks. That leads us to define the interface of the function *region_neighbourhood* that gives the neighbourhood of regions as a function of signature: *toger* × *region* × *neighbouring_mode* × *set of flags* → *list of regions*. It is also useful to have a function that gives any neighbouring region of a given one, for instance when looking for a neighbouring region to merge with. So the function *any_region_neighbour* : *toger* × *region* × *neighbouring_mode* × *set of flags* → *region* returns a neighbouring region of a region according to a neighbouring mode and a set of marks. The function *are_neighbours* : *toger* × *region* × *region* × *neighbouring_mode* → *boolean* checks if two regions are neighbouring regions. Finally more general functions may be easily defined, such as a the function *inner_regions* : *toger* × *region* → *list of regions* which gives the list

of regions located inside the outer boundary of a given region, or the function *all_regions* : *toger* → *list of regions* which gives the list of all the currently defined regions.

Updating the Region Attributes. When implementing a segmentation method it is generally necessary to attach attributes to regions. These attributes have to been initialized and/or updated when regions are split or merged. It may raise a problem of software engineering when the split or the merge is activated by a program module which is not the one which has in charge the update of attributes. In that case the activation of attribute update should be done automatically by the kernel. A classic solution to this problem consists in attaching processings to each side effect function. For instance, we can attach to the *region_split* action a function f which the signature is *region* × *region* × *parameter* → ∅, where *parameter* is a type of generic parameter (like `Object` in Java or `void*` in C and C++). Each time the *region_split* function is executed, the function f is automatically called by the kernel with the regions to be merged as parameters. It may be also possible to attach a list of functions, or functions being called either before or after the side effect. This mechanism is implemented for each side effect function.

For instance let us suppose we want to attach descriptive statistical moments to each region. These moments are be used to compute features like mean or variance. Since these moments are additive, when two regions are merged, the moments can be updated by simply adding themselves the moments of the same order of each region and storing the result in the remaining region [23]. This can easily be done by the way of the mechanism described above.

7 A Full Example of Constrained Segmentation

In this section we describe a full example of segmentation application using the functionalities of the toger kernel. This application was developed in the context of a medical imaging project of examination of temporal sequences of cardiac MRI. Each sequence corresponds to a set of grey level slices acquired in a given plane and varying with time over the cardiac cycle. The goal of the project was to design and implement an algorithm of segmentation of the left ventricle along the whole cycle.

Characteristics of the Problem. Each slice must be split into three regions which are the ventricle cavity (denoted by R_c), the muscle (or myocardial wall) denoted by R_m, and the "exterior" denoted by R_e. The regions boundaries of the segmented image consist of two closed contours, the contour shared by R_c and R_m, called endocardial contour, and the contour shared by R_m and R_e, called epicardial contour (see Fig. 9-a).

The ventricle cavity R_c is a heterogeneous region in which the turbulence of the blood flow induces a large intensity distribution. The region R_m of the muscle is made of muscular fibers and is homogeneous. The exterior R_e is composed of

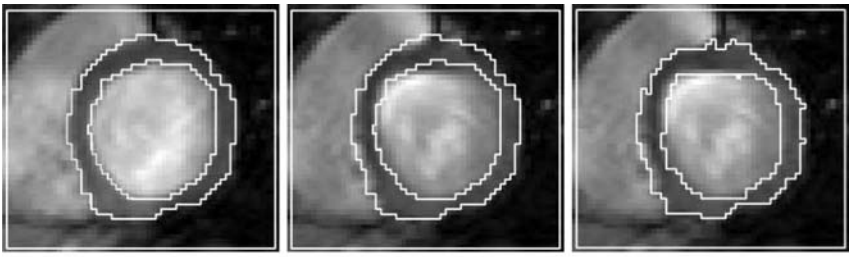


Fig. 9. Endocardial and epicardial contours computed on the $(i-1)^{th}$ slice (a), insertion of the contours of the $(i-1)^{th}$ slice in the $(i)^{th}$ slice (b), and computed contours (c) of the $(i)^{th}$ slice

the right ventricle, lungs and air so it is a heterogeneous region. In slices, the blood flow appears bright and the muscle has mid-gray intensities. The first image of the sequence (telediastolic phase) has be chosen because of a good contrast between the cavity and the muscle. Along the cardiac cycle, the cavity undergoes a phase of contraction followed by a phase of dilation. During the contraction, the quality of contrast between the cavity and the muscle decreases due to signal loss from turbulent blood flow. After telesystole, the extension of the cavity causes contrast loss and noise. The last slice of the sequence is similar to the first.

This problem is related to the quantitative analysis of the cardiovascular system. Several approaches have been used to solve it, based on active contour [3, 31], graph searching [41], deformable templates [59], fuzzy logic [45], and region growing [75]. The methods based on active contours models provide interesting results but these methods are quite sensitive to the initialization process [73] and the turbulence of the blood pool creates two very contrasted zones in the cavity and the active contours converge to this artificial separation. To deal with the problems raised by the turbulence of the blood flows, the artifacts and the noise inherent to cardiac examinations, it turns out to be necessary to take in account the cardiac properties. In this context, a constrained split and merge method based on the toger kernel has been developed. The constraints were defined in order to take in account the topological and geometrical characteristics of the left ventricle over time. This method is totally dedicated to the segmentation of the left ventricle [28, 29].

Principle of the Method. The endocardial and epicardial contours in the first image of the sequence are adjusted manually by editing the result of an automatic split and merge segmentation. Both contours are then propagated from each slice to the next one (see Fig. 9-b). Each time, the regions delimited by these contours are modified by a constrained split and merge algorithm in order to compute the new contours (see Fig. 9-c). This approach allowed us to design a method which is robust despite noise, artifacts and time varying contrast.

Let us denote by R_c^i, R_m^i and R_e^i the three regions of the i^{th} slice, and let us suppose that the regions R_c^{i-1}, R_m^{i-1} and R_e^{i-1} of the $(i-1)^{th}$ slice are known. First the segmentation of the i^{th} slice is initialized with the three regions R_c^{i-1}, R_m^{i-1} and R_e^{i-1} (see Fig. 10-a). Then the three regions are split independently into homogeneous sub-regions (see Fig. 10-b). Each region split is based on a multi-thresholding computed from the region histogram. A first classification of the obtained sub-regions is achieved which gives a set of *stable* sub-regions which are set of pixels that remain in the same cavity from the $(i-1)^{th}$ slice to the i^{th} one. Each stable sub-region is thus removed by merging it with a suitable neighbour (see Fig. 10-c).

The other sub-regions are the *unstable* sub-regions. Each unstable sub-region is a region which is adjacent to the boundary of a cavity of the $(i-1)^{th}$ slice and which satisfies some topological and geometrical constraints. In the last step of the method, such sub-regions are assigned to one of both their neighbouring cavities. The cavities contour evolves each time the assignment of an unstable sub-region changes. At the end of the stage all the unstable sub-regions are merged with a suitable neighbour according to their assignment (see Fig. 10-d).

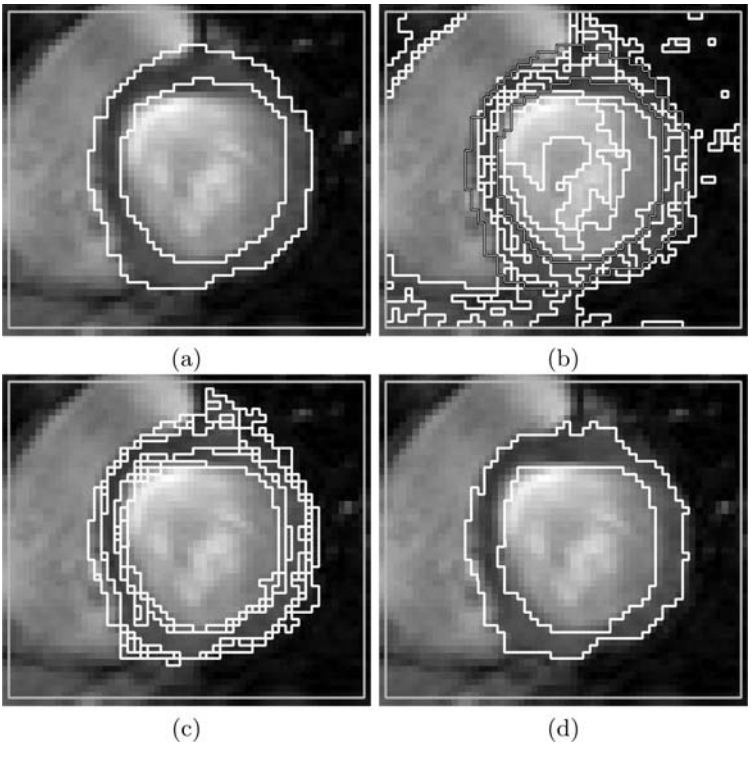


Fig. 10. The regions R_c^{i-1}, R_m^{i-1} and R_e^{i-1} are projected on the i^{th} slice (a), each region is split independently (b), the stable sub-regions are determined and removed (c), and the unstable sub-regions are finally merged with one of their neighbouring cavities (d)

Constraints. Two conditions have been retained to determine stable sub-regions: *(i)* the initial topology of the first slice must be preserved, and *(ii)* the geometrical deformations on the epicardial and the endocardial contours must be small (in terms of surface and perimeter the cavity and the muscle are quite constant from slice to slice). According to these conditions, four constraints have been defined to select stable sub-regions. The first and second ones are topological constraints which goal is to satisfy condition *(i)*. The first one identifies sub-regions that does not intersect a boundary of at least one of the regions R_c^{i-1}, R_m^{i-1} and R_e^{i-1} and the second one the regions of the muscle that share a frontier with both the cavity and the exterior, making a bridge between the cavity and the exterior.

The last two constraints are geometrical constraints which goal is to satisfy condition *(ii)*. The third constraint identifies sub-regions which represent a significant ratio of the region R_α^{i-1} in which they are included. If the assignment of such a sub-region changes, then the region R_α^i undergoes an important surface modification (in comparison with R_α^{i-1}). Considering that these sub-regions are stable prevents the surface of the regions R_c, R_m and R_e from significantly increasing or decreasing from one slice to the next one. The endocardial contour undergoes more deformations than the epicardial one. Thus the ratio associated with the surface of the sub-regions adjacent to the endocardial contour is greater than the ratio associated with the surface of the sub-region adjacent to the epicardial one. The fourth constraint is dedicated to the perimeter variation. A significant increase in perimeter without an important change of surface can be interpreted as the presence of an asperity in the endocardial or the epicardial contours. In order to avoid these defects of contours, a sub-region the reassignment of which would induce a significant change of the length of the endocardial or epicardial contour is selected as stable.

These constraints can be summarized as follows:

1. Let r be a sub-region previously assigned to R_α^{i-1}. If $\partial r \cap \partial R_\alpha^{i-1} = \varnothing$ then r is stable, and is assigned to R_α^i.
2. Let r be a sub-region previously assigned to R_m^{i-1}. If $\partial r \cap \partial R_c^{i-1} \neq \varnothing$ and $\partial r \cap \partial R_e^{i-1} \neq \varnothing$ then r is stable and is assigned to R_m^i.
3. Let us consider a sub-region r that shares the endocardial contour or the epicardial contour and is included in $R_\alpha^{i-1} (\alpha \in \{c, m, e\})$. If $\frac{|r|}{|R_m^{i-1}|} \geq \varepsilon_1^\gamma$ then r is stable and is assigned to R_α^i (ε_1^γ is a constant that depends on the contour γ).
4. Let us consider a sub-region r previously assigned to R_α^{i-1}. If $\frac{|\partial r \cap \partial R_\alpha^{i-1}|}{|\partial r|} \leq \varepsilon_2$ then r is stable and is assigned to R_α^i

Classification of Unstable Sub-regions. The unassigned sub-regions share the contours of the region R_m^{i-1} and they have an "insignificant" area. They can be assigned to anyone of their neighbouring sub-regions while respecting the topological and geometrical constraints. For the sub-regions sharing the endocardial contour this assignment is done according to an heuristic function F. In this case, the sub-regions have to been assigned either to the cavity or to the

muscle. The function $F(r, R)$ measures the deformation induced by the merging of the sub-region r with the region R, R being R_c^i or R_m^i. This function, which is similar to a criterion by Beveridge et al [9] is defined as follows[2]:

$$F(r, R_l) = \alpha F_{sim}(r, R_l) + \beta F_{hom}(r, R_l) + \gamma F_{com}(r, R_l)$$

where μ_r denotes the mean of r and

$$F_{sim}(r, R_l) = \frac{|\mu_r - \mu_l|}{max(\mu_r, \mu_c, \mu_m)}$$
$$F_{hom}(r, R_l) = \frac{\boldsymbol{QE}(r \cup R_l) - max(\boldsymbol{QE}(r), \boldsymbol{QE}(R_l))}{\boldsymbol{QE}(r \cup R_l)}$$
$$F_{com}(r, R_l) = 1 - \frac{|\partial r \cap \partial R_l|}{|\partial r|}$$

and with $l \in \{c, m\}$, $\alpha \geq 0$, $\beta \geq 0$, $\gamma \geq 0$, and

$$\boldsymbol{QE}(r) = \sum_{p \in r} i_p^2 - \mu_r * \sum_{p \in r} i_p$$

The function F_{sim} measures of the similarity of two regions, the function F_{hom} measures the perturbation of the grey-level distribution resulting from merging two regions and the function F_{com} measures the common perimeter of two regions. Each of them varies from zero to one and for each one the merge is favored when the function is close to zero. We consider that the geometrical function and the combination of the colorimetric and the homogeneity functions have the same contribution. Then, the weighting parameters are set to $\alpha = \beta = 0.25$ and $\gamma = 0.5$.

The sub-regions sharing the epicardial contour must be assigned either to R_m or to R_e. Since the exterior R_e is very heterogeneous the assignment function can only take into account the considered sub-region and the muscle region R_m. The distribution of grey level of the muscle is assumed to respect a Gaussian distribution. A sub-region is assigned to the muscle if its own grey level distribution is similar to the distribution of grey level of the muscle. This condition can be expressed as follows:

$$\frac{\mu - \mu_m}{\sigma_m} \leq t \Rightarrow r \in R_m$$

Implementation. Let us now describe how to implement this method with the API of the toger kernel. First, there is nothing to do to initialize the segmentation of the i^{th} slice with the segmentation of the $(i-1)^{th}$ one. We only keep the current representation and switch to the new slice. Let t be this current representation.

[2] The main difference between this function and the Beveridge et al. criterion is that function F selects a region according to an evaluation whereas the Beveridge's criterion decides whether the merge between two regions is possible.

Since the label of the infinite region is always *-1* the region R_e is the only element of the list returned by the expression *region_neighbourhood(t, -1, INNER)*. Likewise the region R_m can be obtained by the expression *region_neighbourhood(t,* R_e*, INNER)*, and the region R_c by *region_neighbourhood(t,* R_m*, INNER)*.

The three regions may be now split into sub-regions. But we will need to be able to determine if a new sub-region is or not adjacent to an old contour. Thus, before splitting the regions, we mark each contour with a different flag. It can be done by calling *mark_outer_boundary(t,* R_m*, FLAG_1)* and *mark_outer_boundary(t,* R_c*, FLAG_2)*.

If S denote the current slice and $S(p)$ the gray level of the point p in this current slice the histogram H_r of a region r can be computed in the following way:

```
D = region_domain(t, r)
set to 0 all the entries of H_r
for each pair (p_2i, p_2i+1) of D
    for p varying from p_2i to p_2i+1 do
        increment H_r[S(p)]
```

For each region R a multi-thresholding fonction T_R is then computed from the histogram H_R. From this function we can easily define the function $f_R(p_1, p_2)$ required by the function *split_region* as a function which returns the result of the comparison $T_R(S(p_1)) = T_R(S(p_2))$. Thus we can split each of the three regions R_e, R_m and R_c and get the related lists of sub-regions L_e, L_m and L_c in the following way:

```
L_e = split_region(t, R_e, f_e)
L_m = split_region(t, R_m, f_m)
L_c = split_region(t, R_c, f_c)
```

Remark that since the consistency of marks is preserved both the epicardial and the endocardial of the previous segmentation remain available.

Let us now consider the classification of sub-regions into stable and unstable sub-regions. To compute the first constraint we have to check if $\partial r \cap \partial R_\alpha^{i-1}$ is empty. It is equivalent to check if no edge of the boundary of the face associated with r is marked, which can easily be done in the following way:

```
d_0 = beta(t, r)
d = d_0
repeat
    if (dart_is_marked(t, d, ANY) or dart_is_marked(t, -d, ANY))
        return false
    d = phi(t, d)
until d = d_0
return true
```

The flag *ANY* is the union of all possible flags.

The second constraint concerns the sub-regions previously assigned to R_m^{i-1}, which is the list L_m. For each sub-region of L_m we have to check if $\partial r \cap \partial R_c^{i-1} \neq \varnothing$ and $\partial r \cap \partial R_e^{i-1} \neq \varnothing$. It is equivalent to check if the boundary of r is marked

at least once for the epicardial contour (*FLAG_1*) and once for the endocardial contour (*FLAG_2*). It can be done by modifying the previous algorithm in a straightforward way.

To check the third constraint we have to compute the size of a region r. It can be done from the domain of the region returned by the function *region_domain*. If the set of spans of r is $\{(p_0, p_1), ..., (p_{2k}, p_{2k+1})\}$ then the size of r is given by $\Sigma_{i=0}^{k}(P_{2k+1} - P_{2k} + 1)$. The size $|R_m^{i-1}|$ of the region of the muscle of the previous segmentation can be computed before the splitting step. The test $\frac{|r|}{|R_m^{i-1}|} \geq \varepsilon_1^{\gamma}$ is computed for each remaining sub-region of L_c and of L_c and L_m. The constant ε_1 associated with the endocardial contour is used for all the sub-regions of L_c. For the sub-regions of L_m either the constant associated with the endocardial contour or with the epicardial contour is retained according to the mark (*FLAG_1* or *FLAG_2*) set on at least one edge of the boundary of r. Note that the unmarked sub-regions have been removed by the first constraint and that the sub-regions of R_m marked with two different flags have been removed by the second constraint.

To compute the last constraint $\frac{|\partial r \cap \partial R_\alpha^{i-1}|}{|\partial r|} \leq \varepsilon_2$ it is necessary to compute for each sub-region both the length of its boundary and the length the part of its boundary which intersects a contour, i.e. which is marked. Since unconnected sub-regions (in other terms hole) have been removed by the first constraint each remaining sub-region is simply connected. It is thus enough to consider only the outer boundary which is given by the cycle of $\beta(r)$ in the permutation ϕ. Let us denote by lr and li the length of respectively the outer boundary of r and the common boundary of r and the epicardial or endocardial contour. We can get these features in the following way:

```
d0 = beta(r)
d = d0
lr = 0
li = 0
repeat
    l = segment_length(t, d)
    lr += l
    if (is_marked(t, d))
        li += l
    d = phi(d)
until (d == d0)
```

We have thus $\frac{|\partial r \cap \partial R_\alpha^{i-1}|}{|\partial r|} = \frac{li}{lr}$.

At this stage all the stable sub-regions have been identified. Before to process the unstable sub-regions we have to remove stable sub-regions by iterating merging. It can be done in a straightforward way by traversing the list of stable sub-regions and by merging each stable sub-region r with any neighbouring sub-region r' being in the same region, i.e. such that the common boundary of r and r' is unmarked:

```
for each stable sub-region r do
    r' = region_any_neighbour(t, r, ANY, NOMARK)
    region_merge(t, r, r')
```

We can now consider the processing of unstable sub-regions and thus the computation of the function $F(r, R_l)$. The region R_l for $l \in \{e, m, c\}$ is the last sub-region merged when processing the stable sub-regions of the list L_l. As described above it is straightforward to initialize the descriptive moments $M_i(r) = \sum_{p \in r}(S(p))^i$ when regions are split and to update them when sub-regions are merged. The three first descriptive moments M_0, M_1 and M_2 are attached to each region of the representation which allows to get $\mu_r = \frac{M_1(r)}{M_0(r)}$, and $\boldsymbol{QE}(r) = M_2(r) - \frac{M_1^2(r)}{M_0(r)}$. To have $F(r, R_l)$ we also need to know $|\partial r|$ and $|\partial r \cap \partial R_l|$ the computation of which has been described above. Each sub-region of L_c (resp. L_e) may be merged either with R_c or R_m (resp. R_m or R_e) according to F. Each sub-region of R_m falls the first case if it is adjacent to the endocardial contour and in the second case otherwise, which can be determined from the mark. When the three lists have been processed the three remaining regions are the new regions R_c, R_m and R_e and the marks are removed by using the function *unmark_all_darts*.

Results. This algorithm has been tested on 33 temporal sequences of short axes MR images (see Fig. 11 for an example of result). Each of these sequences has been acquired from a different patient. The validation has been done for medium-plane sequences. The results have been qualified by hospital practitioners: 12

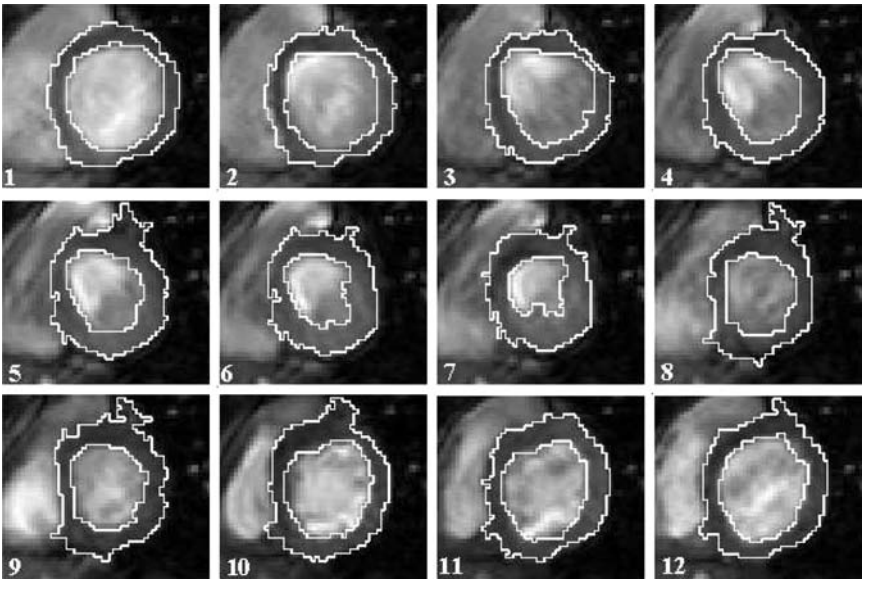


Fig. 11. An example of automatic segmentation of the left ventricle along twelve slices of a temporal MRI sequence

excellent, 8 good, 8 medium, 4 insufficient and 1 mediocre segmentations of an entire sequence. The study of inter- and intra-observer reproducibility made for these sequences, shows that our algorithm corresponds to the expectations of the hospital practitioners; that is a decrease of the inter and intra variability.

8 Conclusion and Perspectives

In this paper we have given an overview of the use of planar maps with discrete embedding in the context of image segmentation. We have described how to represent the topology of a segmented image with a set of planar maps and how to associate with planar maps the boundaries of a discrete segmented image. We have show how this model can be used to implement most of operations involved in the design of segmentation algorithms, such as reconstruction of the geometry of the boundary and of the domain of a region, point inclusion, region localisation, obtaining of geometric and topological features, and updating involved by region splitting and merging. We have described the data structures and the algorithms that permits to use this model in the context of image segmentation.

This model has been used with success to implement various segmentation applications. We have thus decided to capitalize on this experiment to develop a portable general image representation library implementing the model and the algorithms summarized in this paper. A first version of the API of this library have been presented in section 6. Finally we have illustrated this approach by describing a full example of constrained segmentation method designed with this API. The General Image Representation Library will be available under LGPL in the course of year 2005.

In parallel we are working on the extension on this model to represent and segment three-dimensional discrete images. Two different models were proposed, one of them by Braquelaire, Desbarats, Domenger and Wütrich [16, 34, 13, 35] and the other one by Bertrand, Damiand and Fiorio [7, 33, 8]. We are currently collaborating to mix both models and design an implementation of a 3D general image representation library [12].

Acknowledgement. This paper summarizes works made in the context of a LaBRI research project to which several people have contributed: Jean-Philippe Domenger and Luc Brun concerning the image representation and segmentation sides, Christelle Cassen, Jean-Philippe Domenger and Jean-Louis Barat concerning the segmentation of cardiac MRI, and Anne Vialard concerning the geometrical smoothing of contours. I specially thank Jean-Philippe Domenger for his help in the preparation on this paper.

References

1. E. Ahronovitz, J.P. Aubert, and C. Fiorio. The star-topology: a topology for image analysis. In *5th DGCI Proceedings*, pages 107–116, 1995.

2. C. Ang, H. Samet, and C.A. Shaffer. A new region expansion for quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(7):682–686, 1990.
3. C. Baldy, P. Douek, P. Croisille, I.E. Magnin, D.Revel, and M. Amiel. Automated myocardial edge detection from breath-hold cine-mr images : Evaluation of left ventricular volumes and mass. *Magnetic Resonance Imaging*, 12(4):589–598, 1994.
4. P. Baudelaire and M. Gangnet. Planar maps: an interaction paradigm for graphic design. In *Proc. of CHI'89*, pages 313–318. Addison-Wesley, 1989.
5. J. Benois-Pineau, A. Braquelaire, and A. Ali-Mhammad. Interactive fine object-based segmentation of generic video scenes for object-based indexing. In Ebroul Izquierdo, editor, *proc. of WIAMIS'2003*, pages 200–203, 2003.
6. J. Benois-Pineau, G. Peretie, and A. Braquelaire. Adaptive video pre-processing for bit-rate reduction in object-based predictive coding schemes. In *proc. of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, pages 27–30, Orlando, Florida, July 2003.
7. Y. Bertrand, G. Damiand, and C. Fiorio. Topological encoding of 3d segmented images. In *Discrete Geometry for Computer Imagery*, number 1953 in Lect. Note in Comp. Science, pages 311–324, Uppsala, Sweden, december 2000.
8. Y. Bertrand, G. Damiand, and C. Fiorio. Topological map : minimal encoding of 3D segmented images. In *In proc. of GBR 2001, ISBN 887146579-2*, pages 63–73, 2001.
9. J.R. Beveridge, J. Griffith, R. Kohler, A.R. Hanson, and E.M. Riseman. Segmenting images using localised histograms and region merging. *International Journal of Computer Vision*, 2:311–347, 1989.
10. H. Bieri. Hyperimages – an alternative to the conventional digital images. In *Eurographics'90 proceedings*, pages 341–352, 1990.
11. M. Bister, J. Cornelis, and A. Rosenfeld. A critical view of pyramid segmentation algorithms. *Pattern Recognit Letter.*, 11(9):605–617, 1990.
12. A. Braquelaire, G. Damiand, J.P. Domenger, and F. Vidil. Comparaison and convergence of two topological models for 3D image segmentation. In *proc. of GBR 2003, ISBN 887146579-2*, pages 32–43, 2003.
13. A. Braquelaire, P. Desbarats, and J.P. Domenger. 3D split and merge with 3-maps. In *In proc. of GBR 2001, ISBN 887146579-2*, pages 32–43, 2001.
14. J.P. Braquelaire and L. Brun. Image segmentation with topological maps and inter-pixel representation. *Journal on Visual Communication and Image Representation*, 9(1):62–79, 1998.
15. J.P. Braquelaire, L. Brun, and A. Vialard. Inter-pixel euclidean paths for image analysis. In *Lecture Notes in Computer Science*, volume 1176, pages 193–204. Discrete Geometry for Computer Imagery, Springer-Verlag, 1996.
16. J.P. Braquelaire, P. Desbarats, J.P. Domenger, and C. Wütrich. A topological structuring for aggregates of 3D discrete objects. In *Proc of GBR'99, Osterreichische Computer Gesellschaft, ISBN 3-8580-126-2*, pages 193–202, 1999.
17. J.P. Braquelaire and J.P. Domenger. Intersection of discrete contours. In *Proc. of Compugraphics'91*, pages 434–444, 1991.
18. J.P. Braquelaire and P. Guitton. A model for image structuration. In *Proc. of Computer Graphics International'88*, pages 426–435. Springer, 1988.
19. J.P. Braquelaire and P. Guitton. $2\frac{1}{2}$ scene update by insertion of contour. *Computer and Graphics*, 15(1):41–48, 1991.
20. J.P. Braquelaire and J.P.Domenger. Representation of segmented images with discrete geometric maps. *Image and vision Computing*, 17:715–735, 1999.

21. J.P. Braquelaire and A. Vialard. Euclidean paths : a new representation of boundary of discrete regions. *Graphical Models and Images Processing*, 61:16–43, 1999.
22. R. Brice and C.L. Fennema. Scene analysis using regions. *Artificial intelligence*, 1:205–226, 1970.
23. L. Brun. *Segmentation d'images couleur à base topologique.* PhD thesis, Université Bordeaux 1, december 1996.
24. L. Brun and J.P. Domenger. A new split and merge algorithm based on discrete map. In *Proc.of WSCG'97*, pages 21–30, 1997.
25. L. Brun, J.P. Domenger, and J.P. Braquelaire. Discrete maps: a framework for region segmentation algorithms. In J.M Jolion and W Kropatsch, editors, *Proc. of Graph based Representations, GbR'97*, pages 83–92. Springer-Verlag, 1998.
26. L. Brun, J.P. Domenger, and M. Mokhtari. Incremental modifications on segmented image defined by discrete maps. *Journal of visual commmunication and Image representation*, 14:251–290, 2003.
27. L. Brun and W. Kropatsch. Introduction to combinatorial pyramids. In G. Bertrand and A. Imiya, editors, *Digital and image geometry*, volume 2243 of *LNCS*, pages 108–127. Springer, 2001.
28. C. Cassen. *Développement d'une méthode de segmentation applicable aux images d'IRM cardiaque.* PhD thesis, Université Bordeaux 1, 2001.
29. C. Cassen, J.P Domenger, A. Braquelaire, and J.L. Barat. Left ventricular segmentation in mri images. In *proc of ISPA 2001, 2nd IEEE Region 8 Eurasip symposium image and signal processing and image analysis*, 2001.
30. L.W. Chang and K.L. Leu. A fast algorithm for the restoration of images based on chain codes descriptions and its applications. *Computer Vision, Graphics and Image Processing : Image Understanding*, 50:296–307, 1990.
31. L.D. Cohen. On active contour models and ballons. *Computer Vision, Graphics and Image Processing : Image Understanding*, 53(2):211–218, 1991.
32. R. Cori. Un code pour les graphes planaires et ses applications. Thèse d'état de l'université Paris VII, and Astŕisque 27, 1973 and 1975.
33. G. Damiand. *Définition et étude d'un modèle topologique minimal de représentation d'images 2D et 3D.* Thèse de doctorat, Université Montpellier II, décembre 2001.
34. P. Desbarats. *Strucuration d'images segmentées 3D discrtes.* Thèse de doctorat, Université Bordeaux 1, décembre 2001.
35. P. Desbarats and J.-P. Domenger. Retrieving and using topological characteristics from 3D discrete images. In *Preoceedings of the 7th Computer Vision Winter Workshop*, pages 130–139. PRIP-TR-72, 2002.
36. J.-P. Domenger. *Conception et implémentation du noyau graphique d'un environnement 2D$\frac{1}{2}$ d'édition d'images discrètes.* PhD thesis, Univ. Bordeaux 1, avril 1992.
37. R.C. Dyer, A Rosenfeld, and S Hanan. Region representation: Boundary codes from quadtrees. *ACM: Graphics and Image Processing*, 23(3):171–179, March 1980.
38. F. Ferri and E. Vidal. Colour image segmentation and labeling through multiedit-condensing. *PRL*, 13:561–568, 1992.
39. C. Fiorio. *Approche interpixel en analyse d'images : une topologie et des algorithmes de segmentation.* Thèse de doctorat, Université Montpellier II, novembre 1995.
40. C. Fiorio. A topologically consistent representaion for image analysis: the topological graph of frontiers. In S. Miguet, A. Montavert, and S. Ubéda, editors, *Lectures Notes in Computer Sciences*, volume 1176, pages 151–162, 1996.

41. S.R. Fleagle, D.R. Thedens, W. Stanford, B.H. Thompson, J.M.Weston, P.P. Patel, and D.J. Skorton. Automated myocardial edge detection on mr images : Accuracy in consecutive dubjects. *Journal of Magnetic Resonance Imaging*, 3:738–741, 1993.
42. H. Freeman. On the encoding of arbitrary geometric configurations. *IRE Trans. Electr. Compu.*, 10:260–268, June 1961.
43. M. Gangnet, J.C. Hervé, T. Pudet, and J.M. VanThong. Incremental computation of planar maps. In *Proc. of SIGGRAPH'89*, 1989.
44. M. Gangnet and J.M. Van Thong. Robust boolean operations on 2D paths. In *Proc. of COMPUGRAPHICS'91*, pages 434–444, 1991.
45. I. Gath and D. Hoory. Fuzzy clustering of elliptic ring-shaped clusters. *Pattern Recognition Letters*, 16:721–741, 1995.
46. G.M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Trans. On Pattern Analysis and Machine Intelligence*, 1(2):145–153, April 1979.
47. P. Jasiobedzki. Adaptive adjacency graphs. In *Proceedings, SPIE Geometric methods in Computer Vision*, San Diego, CA, 1993.
48. J.M. Jolion and A. Montanvert. The adaptative pyramid : A framework for 2D image analysis. *Computer Vision, Graphics and Image Processing : Image Understanding*, 55(3):339–348, May 1992.
49. K.C. Keeler. *Map Representations and Optimal Encoding for Image Segmentation.* Phd thesis, Harvard University, 1990.
50. E. Khalimsky, R. Kopperman, and P.R. Meyer. Boundaries in digital planes. *Journal of applied Math. and Stocastic Analysis*, 3:27–55, 1990.
51. E. Khalimsky, R. Kopperman, and P.R. Meyer. Computer graphics and connected topologies on finite ordered sets. *Topology and its Applications*, 36:1–17, 1990.
52. T.Y. Kong and A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics and Image Processing : Image Understanding*, 48:357–39, 1989.
53. U. Köthe. Xpmaps and topological segmentation – a unified approach to finite topologies in plane. In A. Braquelaire, J.-O. Lachaud, and A. Vialard, editors, *Proc of DGCI'02*, volume 2310 of *LNCS*, pages 22–33. Springer, 2002.
54. V.A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics and Image Processing : Image Understanding*, 46:141–161, 1989.
55. P. Krämer, J. Benois-Pineau, and J.P. Domenger. Scene similarity measure for video content segmentation in the framework of rough indexing paradigme. In *Proc of AMR'2004*, pages 144–155, 2004.
56. W. Kropatsch. Preserving contours in dual pyramids. pages 563–565, 1988.
57. W. Kropatsch. Building irregular pyramids by dual graph contraction. *IEEE Proc. Vision, Image and Signal Processing*, 142(6):366–374, 1995.
58. P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifold. *Int. Journ. of Comp. Geom. and Appl.*, pages 275–324, 1994.
59. P. Lipson, A.L. Yuille, D.O. Keefe, J. Cavanaugh, J. Taaffe, and D. Rosenthal. Deformable templates for feature extraction from medical images. In First European Conference on Computer Vision, editor, *Lecture Notes in Computer Science Computer Vision-ECCV90*, volume 427, pages 413–417, Antibes, France, April 1990. Proccedings Springer-Verlag.
60. R.G. Loomis. Boundary networks. *Communications of the ACM*, 8(1):44–48, 1965.
61. P. Meer. Stochastic image pyramids. *Computer Vision Graphics Image Processing*, 45:269–294, 1989.
62. R.D. Merrill. Representation of contours and regions for efficient computer search. *Communications of the ACM*, 16(2):69–82, 1973.
63. A. Montanvert, P. Meer, and A. Rosenfeld. Hierarchical image analysis using irregular tesselations. *PAMI*, 13(4):307–316, 1991.

64. P. Morse. Concepts of use in contour map processing. *Communications of the ACM*, 12(3):147–152, 1969.
65. C. J. Nicol. A systolic approach for real time connected component labeling. *Computeer vision and Image understanding*, 61(1):17–31, January 1995.
66. T. Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Sci., Washington, 1982.
67. M. Pietikainen, A. Rosenfeld, and I. Walter. Split and link algorithms for image segmentation. *Pattern Recognition*, 15(4):287–298, 1982.
68. S.V. Raman, S. Sarkar, and K.L. Boyer. Hypothesizing structures in edge-focused cerebral magnetic resonance images using graph-theoretic cycle enumeration. *CVGIP: Image Understanding*, 57(1):81–98, 1993.
69. A. Rosenfeld. Digital topology. *Amer. math. monthly*, 86:621–630, 1979.
70. A. Rosenfeld. *Picture Languages*. Academic Press, 1979.
71. A. Rosenfeld and A. Kak. *Digital Picture Processing*, volume 2. Academic Press, 1982.
72. H. Samet. Region representation: Quadtrees from boundary codes. *ACM : Graphics and Image Processing*, 23(3):163–170, March 1980.
73. A. Sebbahi, A. Herment, A. de Cesare, and E. Mousseaux. Multimodality cardiovascular image segmentation using a deformable contour model. *Computerized Medical Imaging and Graphics*, 21(2):79–89, 1997.
74. L.G. Shapiro. Connected component labeling and adjacency graph construction. In Kong and Rosenfeld, editors, *Topological Algorithms for Digital Image Processing (Machine Intelligence and Pattern Recognition, Volume 19)*. Elsevier, 1996.
75. K.R. Singleton and G.M. Pohost. Automatic cardiac mr image segmentation using edge detection by tissue classification in pixel neigborhoods. *Magnetic Resonance in Medecine*, 37:418–424, 1997.
76. M. Suk. A new segmentation technique based on partition mode test. *PaRe*, 16(5):469–480, 1983.
77. S.L. Tanimoto. Image data structures. In S. L. Tanimoto and A. Klinger, editors, *Structured Computer Vision*, pages 31–56. Academic Press, New York, 1980.
78. W.T. Tutte. A census of planar maps. Canad.J.Math., 15:249–271, 1963.
79. A. Vialard. *Chemins euclidiens : Un modèle de représentation des contours discrets*. Phd thesis, Université Bordeaux 1, 1996.
80. A. Vialard. Geometrical parameters extraction from discrete paths. *Lecture Notes in Computer Science*, 1176:24–35, 1996. Discrete Geometry for Computer Imagery'96.
81. A. Vialard and J.P. Braquelaire. Transformations géométriques de courbes discrètes 2D. In *3èmes journées de l'AFIG (Marseille)*, 1995.
82. D. Willersinn and W.G. Kropatsch. Dual graph contraction for irregular pyramids. In *International Conference on Pattern Recogntion D: Parallel Computing*, pages 251–256, Jerusalem, Israel, 1994. International Association for Pattern Recognition.

Inside and Outside Within Combinatorial Pyramids

Luc Brun[†] and Walter Kropatsch[‡,⋆]

[†] GreYC -CNRS UMR 6072,
ENSICAEN,
6 Boulevard du Maréchal Juin,
14045 Caen(France)
krw@prip.tuwien.ac.at
[‡] Institute for Computer-aided Automation,
Pattern Recognition and Image Processing Group,
Vienna Univ. of Technology- Austria
luc.brun@greyc.ismra.fr

Abstract. Irregular pyramids are made of a stack of successively reduced graphs embedded in the plane. Such pyramids are often used within the segmentation and the connected component analysis frameworks to detect meaningful objects together with their spatial and topological relationships. The graphs reduced in the pyramid may be region adjacency graphs, dual graphs or combinatorial maps. Using any of these graphs each vertex of a reduced graph encodes a region of the image. Using simple graphs one edge between two vertices encodes the existence of a common boundary between two regions. Using dual graphs and combinatorial maps, each connected boundary segment between two regions is associated to one edge. Moreover, special edges called loops may be used to differentiate a special type of adjacency where one region surrounds the other. We show in this article that the loop information does not allow to distinguish inside and outside of the loop by local computations. We provide a method based on the combinatorial pyramid framework which uses the orientation explicitly encoded by combinatorial maps to determine inside and outside with local calculus.

1 Introduction

An irregular pyramid [7] is defined as a stack of successively reduced graphs. The hierarchical representation provided by such pyramids allows to reduce the computational cost of many graph algorithms using reduced versions of the initial graph. This representation also provides a nice framework for graph algorithms based on a divide and conquer strategy. Finally, the irregular pyramids provide

⋆ WK was supported by the Austrian Science Foundation under grants P14662-INF and FSP-S9103-N04.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 122–131, 2005.

a global representation which may be used to add further constraints on many graph algorithms.

Irregular pyramids have been widely used to encode partitions within the segmentation and the connected component analysis frameworks [7, 8]. The dual graph Pyramids introduced by Kropatsch [6] are defined as a stack of dual graphs successively reduced. Within such pyramids the mapping of a non surviving vertex to a surviving one is performed by the contraction of their common edge. The contraction of a graph reduces the number of vertices while maintaining the connections to other vertices. As a consequence self loops or multiple edges may occur, some of them being redundant in that they do not surround any part of the graph. Such edges surround thus "empty inside" and are called empty self loops. These redundant edges may be locally characterized in the dual of the graph and suppressed by a removal step.

One particular type of adjacency between two regions is called the *includes* relationship. The includes relationship relates two regions, one is placed 'outside', the other is 'inside' and is surrounded by the outside region in the embedding (see e.g. the arrows in Fig. 3). A self-loop incident to the vertex encoding the outside surrounds the inside (Fig. 2(a)). The edge corresponding to the self-loop in the dual graph is a bridge connecting inside and outside region. Without orientation the exchange of inside and outside does not change the topology of the graph, the two graphs are indistinguishable. Such loops which are not present at the base level are created by the successive contraction and removal operations applied to build the pyramid.

A Combinatorial Pyramid is defined as a stack of successively reduced combinatorial maps. The reduction scheme used within the combinatorial pyramid framework is similar to the one used within dual graph pyramid. However, the formalisms of the combinatorial and dual graph pyramids are quite different. One of the main specific property of combinatorial maps is the explicit encoding of the orientation of the plane. The method presented in this paper uses the explicit encoding of the orientation by combinatorial maps to differentiate usual adjacency relationships from the includes ones.

The rest of this paper is structured as follows: We first present in section 2 the main properties of the combinatorial maps. Then we present in section 3 the combinatorial pyramid framework together with the main concepts used to compute inside relationships. Finally, we present the problem of the determination of the includes relationships in section 4 together with one method to determines the set of regions inside a given one. We conclude this last section with an experiment illustrating the usefulness of includes relationship.

2 Orientation in Combinatorial Maps

A combinatorial map $G = (\mathcal{D}, \sigma, \alpha)$ encodes a partition on an orientable surface without boundary. Combinatorial maps are used within the image processing and analysis framework to encode image's partitions. Using $2D$ images, combinatorial maps may be understood as a particular encoding of a planar graph where

each edge is split into two half-edges called darts. Since each edge connects two vertices, each dart belongs to only one vertex. A $2D$ combinatorial map is formally defined by the triplet $G = (\mathcal{D}, \sigma, \alpha)$ where $\mathcal{D}$ represents the set of darts and σ is a permutation on $\mathcal{D}$ whose cycles correspond to the sequence of darts encountered when turning counter-clockwise around each vertex. Finally α is an involution on $\mathcal{D}$ which maps each of the two darts of one edge to the other one. Given a combinatorial map $G = (\mathcal{D}, \sigma, \alpha)$, its dual is defined by $\overline{G} = (\mathcal{D}, \varphi, \alpha)$ with $\varphi = \sigma \circ \alpha$. The cycles of permutation φ encode the faces of the combinatorial map and may be interpreted as the sequence of darts encountered when turning clockwise around a face. In what follows, the cycles of α, σ and φ containing a dart d will be respectively denoted by $\alpha^*(d)$, $\sigma^*(d)$ and $\varphi^*(d)$. An introduction to combinatorial maps and combinatorial pyramids may be found in [4, 2].

Fig. 1(a) describes a dual combinatorial map $\overline{G} = (\{\blacksquare\}, \{\blacksquare - \blacksquare\})$ encoding a 3×3 4-connected planar sampling grid. Using this encoding the φ, σ and α cycles of each dart may be respectively understood as elements of dimensions 0, 1 and 2 and formally associated to a 2D cellular complex [4]. More precisely, each α cycle may be associated to a crack between two pixels. Each of the two darts of an α cycle corresponds to an orientation along the crack. For example, the cycle $\alpha^*(1) = (1, -1)$ is associated to the crack encoding the right border of the top left pixel of the 3×3 grid (Fig. 1(a)). The darts 1 and -1 define respectively a bottom to top and top to bottom orientation along the crack.

These results may be extended to any reduced combinatorial map encoding an image partition. In this case, each dart of the map should be interpreted as a sequence of oriented cracks encoding an oriented and connected boundary segment between two regions. Such a sequence is simply called a *segment*. The relationships between segments and cracks are as follows: Each oriented crack belongs to at most one segment. Moreover, if one oriented crack belongs to a segment associated to a dart d, the same crack with an opposite orientation belongs to the segment associated to $\alpha(d)$. For example, the dart 16 in Fig. 1 (c) is

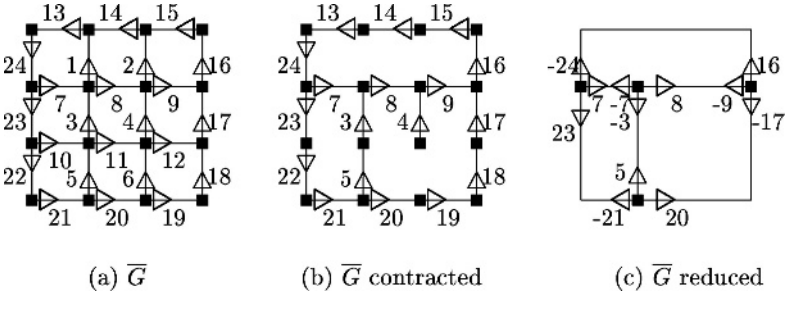


Fig. 1. A dual of a combinatorial map (a) encoding a 3×3 grid with the contracted combinatorial map (b) obtained by the contraction of $K_1 = \alpha^*(1, 2, 10, 11, 12, 6)$. The reduced combinatorial map (c) is obtained by the removal of the empty self loops defined by $K_2 = \alpha^*(4)$ and the double edges defined by $K_3 = \alpha^*(13, 14, 15, 19, 18, 22) \cup \{24, -16, 17, -20, 21, -23, 3, -5\}$

associated to the sequence of oriented cracks encoded by the darts 16.15.14.13.24 (Fig. 1(b)) while the dart -24 is associated to $-24. - 13. - 14. - 15. - 16.$

3 Connected Boundary Segments and Orientation Within the Combinatorial Pyramid Framework

As in the dual graph pyramid scheme [6] (Section 1) a combinatorial pyramid is defined by an initial combinatorial map successively reduced by a sequence of contraction or removal operations. Contraction operations are encoded by contraction kernels. These kernels defined as a forest of the current combinatorial map may create redundant edges such as empty-self loops and double edges(Fig. 1(b)). Empty self loops (edge $\alpha^*(4)$ in Fig. 1(b)) may be interpreted as region's inner boundaries and are removed by an empty self loops removal kernel after the contraction step. The remaining redundant edges called double edges, belong to degree 2 vertices in $\overline{G}$ (e.g. $\varphi^*(13)$, $\varphi^*(14)$, $\varphi^*(15)$) in Fig. 1(b)) and are removed using a double edge removal kernel which contains all darts incident to a degree 2 dual vertex. Note that, any combinatorial map deduced from the application of a contraction kernel followed by the two removal kernels cannot contain empty self loops. No dart d of such a combinatorial map may thus satisfy the relationship : $\sigma(d) = \alpha(d)$. Further details about the construction scheme of a pyramid may be found in [2, 3].

As mentioned in Section 2, each dart of a reduced combinatorial map may be associated to a sequence of oriented cracks called a segment. Since each oriented crack is encoded by one dart in the base level combinatorial map G_0(Section 2), a segment may be equivalently defined as a sequence of darts belonging to G_0. Let us consider a combinatorial map $G_i = (\mathcal{D}_i, \sigma_i, \alpha_i)$ defined at level i such that G_i does not contain any empty self loop. Given a dart d in $\mathcal{D}_i$ the sequence $d_1 \dots d_n$ encoding the segment associated to d is defined by [2]:

$$d_1 = d \, , d_{j+1} = \varphi_0^m(\alpha_0(d_j)) \text{ and } \alpha_0(d_p) = \alpha_i(d). \tag{1}$$

where $\overline{G_0} = (\mathcal{D}_0, \varphi_0, \alpha_0)$ is the dual of the initial combinatorial map and m is the minimal integer such that $\varphi_0^q(\alpha_0(d_j))$ survives at level i or belongs to a double edge kernel. This last condition is tested in constant time using the implicit encoding of combinatorial pyramids [2].

Note that, if G_0 encodes the 4-connected planar sampling grid, each φ_0 cycle is composed of at most 4 darts (Fig. 1(b)). Therefore, the computation of d_{j+1} from d_j requires at most 4 iterations and the determination of the whole sequence of cracks composing a boundary between two regions is performed in a time proportional to the length of this boundary.

Each oriented crack associated to an initial dart d_j may be encoded by the position of its starting point and one move. Using a 4-connected sampling grid these moves belong to $\{right, up, left, down\}$. Given a dart d of G_i, let us denote respectively by $Fm(d)$ and $Lm(d)$ the moves of the first and last oriented cracks of the segment associated to d. If $d_1 \dots d_p$ denotes the sequence of initial darts associated to d, we have $d_1 = d$ and $d_p = \alpha_0(\alpha_i(d))$ (equation 1). The

two darts d_1 and d_p may thus be retrieved in constant time from d. Moreover, $Fm(d)$ and $Lm(d)$ are equal to the move of the oriented cracks respectively associated to d_1 and d_p. This correspondence between the oriented cracks and the initial darts may be defined using any implicit numbering of the initial darts (see e.g. Fig. 1(a)). The values of $Fm(d)$ and $Lm(d)$ may thus be retrieved without additional memory requirement and in constant time using an appropriate numbering of the initial darts. For example, the first and last moves of the dart 16 in Fig. 1(c) are associated to the moves of the darts 16 and $\alpha_0(-24) = 24$ in $\overline{G_0}$ (Fig. 1(a)) and are respectively equal to *up* and *down*.

Given a dart d in G_i, and the sequence of darts $d_1 \dots d_p$ in G_0 encoding its segment, the properties of the segments (Section 2) together with the properties of the combinatorial pyramids [2] induce the two following properties:

$$\forall j \in \{1, \dots, p-1\} \quad move(d_j)^{-1} \neq move(d_{j+1}) \tag{2}$$

$$Lm(d) \neq Fm(\sigma_i(d))^{-1} \tag{3}$$

where $move(d_j)$ denotes the move of the oriented crack associated to d_j and $move(d_j)^{-1}$ is the opposite of the move of d_j (e.g. $right^{-1} = left$).

Equation 2 states that two successive moves within a segment cannot be opposite. This property is induced by the fact that one segment cannot contain twice a same crack with two orientations. Equation 3 states that the first move of the σ_i successor of a dart d cannot be the opposite of the last move of d. Otherwise, the dart d would be an empty self loop of G_i which is refused by hypothesis.

Given a dart d_1 in G_i, let us consider a sub-sequence $d_1 \dots . d_q$ of $\sigma_i^*(d_1)$ such that $d_q \neq \alpha_i(d_1)$. Let us also consider the sequence S of oriented cracks defined as the concatenation of the segments associated to $d_1 \dots . d_q$. The orientation of the sequence $d_1 \dots . d_q$ is then defined as the overall number of clockwise turns between the successive cracks along S. In order to measure such an orientation we define the angle between two successive oriented cracks as :

- +1 if the two oriented cracks define a clockwise 90° turns,
- -1 if the two oriented crack define a counter-clockwise 90° turns,
- 0 if the two oriented crack correspond to a same move,
- undefined if the two oriented cracks correspond to opposite moves.

Such angles may be easily encoded using a basic 4×4 array indexed by the Freeman's codes of the moves: right,up, left and down are numbered from 0 to 3. The angle between two moves $m1$ and $m2$ is denoted by $(m1, m2)\widehat{}$. We have for example, $(right, right)\widehat{} = 0$, $(right, up)\widehat{} = -1$,$(right, down)\widehat{} = +1$ and $(right, left)\widehat{} = undefined$.

Given the angle between two successive oriented cracks we define the orientation of a dart as the sum of the angles between the oriented cracks along its associated segment. Given a dart d in G_i the orientation of d is thus defined by:

$$or(d) = \sum_{j=1}^{n-1} (move(d_j), move(d_{j+1}))\widehat{} \tag{4}$$

where $d_1 \dots d_n$ is the the sequence of initial darts encoding the segment associated to d. Note that $(move(d_j), move(d_{j+1})^\frown$ cannot be undefined for any $j \in \{1, \dots, n-1\}$ (equation 2).

The orientation of a dart may be computed on demand using equation 4 or may be attached to each dart and updated during the construction of the pyramid. Indeed, let us consider two successive double darts d_1 and d_2 at one level of the pyramid. If d_1 survives at the above level its orientation may be updated by [1]:

$$or(d_1) = or(d_1) + or(d_2) + (Lm(d_1), Fm(d_2))^\frown \tag{5}$$

Note that this last formula may be extended to the removal of a sequence of successive double edge.

The dart's orientation may thus be computed by fixing the orientation of all initial darts to 0 and updating the dart's orientation using equation 5 during the removal of each double edge kernel.

Let us consider a sequence $d_1 \dots d_q$ in G_i such that $d_{j+1} = \sigma_i(d_j)$ for all j in $\{1, \dots, p-1\}$ and $d_q \neq \alpha_i(d_1)$. Its orientation is defined by:

$$or(d_1 \dots d_q) = \left(\sum_{j=1}^{q-1} or(d_j) + (Lm(d_j), Fm(d_{j+1}))^\frown \right) + or(d_q) \tag{6}$$

The quantity $(Lm(d_q), Fm(d_1))^\frown$ has to be added to $or(d_1 \dots d_q)$ if the sequence defines a closed boundary. Note that $(Lm(d_j), Fm(d_{j+1}))^\frown$cannot be undefined for any $j \in \{1, \dots, q-1\}$ (equation 3). Moreover, one can show that if the sequence defines a closed boundary and if $Lm(d_q) = Fm(d_1)^{-1}$, then we should have $\alpha_i(d_q) = d_1$, which is refused by hypothesis.

Using the same notations and hypothesis than equation 6, one important result shown by Braquelaire and Domenger [1] states that the orientation of a sequence $d_1 \dots, d_q$ defining a closed boundary is equal to 4 if it is traversed clockwise and -4 otherwise. Moreover, this sequence corresponds to:

- a finite face of $\overline{G_i}$ and thus a region if its orientation is equal to -4,
- a set of faces of $\overline{G_i}$ connected by bridges and included in one face if the orientation is equal to 4. Such a set of faces is called an infinite face [1].

By construction each combinatorial map G_i of a combinatorial pyramid is connected and all faces of G_i but one define a finite face. The infinite face of a combinatorial map encodes the background of the image (denoted by E in Fig. 2). The above property is used in Section 4 to compute inside and outside adjacency relationships with local calculus.

4 Computing Inside Relationships

As in the dual graph pyramid framework, the inclusion relationships are encoded within the combinatorial pyramid framework by self-loops: One vertex v adjacent

to a vertex w and surrounded by one loop of w encodes a region included in the region associated to w.

This property is illustrated in Fig. 2 where the image is partitioned into 4 regions. The region encoded by the vertex A includes the two regions B and C and is adjacent to the region D. The vertex E encodes the background of the image. These inclusion relationships are encoded in the combinatorial map G by the loop $(1, -1)$ which surrounds vertex B and the two nested loops $(1, -1)$ and $(3, -3)$ which surround vertex C. Note that each loop corresponds to a bridge in $\overline{G}$ (Fig. 2(b)).

Let us denote the combinatorial map represented in Fig. 2 by $G = (\mathcal{D}, \sigma, \alpha)$. The σ cycle of vertex A is equal to $\sigma^*(1) = (1, 2, 3, 4, -3, 5, -1, 6, 7, 8)$. If we suppose that the loop $(1, -1)$ does not surround the vertices B and C incident to the edges $\alpha^*(2, 3, 4, -3, 5)$ but the vertices D and E incident to $\alpha^*(6, 7, 8)$ we obtain the same σ orbit $\sigma^*(1)$. This last remark shows that inside relationships cannot be decided locally without additional information. Note that this problem is not specific to the combinatorial pyramid framework. Indeed the same example may be built within the dual graph pyramid framework leading to the same drawback.

Let us consider a combinatorial map $G_i = (\mathcal{D}, \sigma_i, \alpha_i)$ defined at the level i of a combinatorial pyramid and one loop $\alpha_i^*(d)$ of G_i such that $\sigma_i^*(d) = (d, d_2, \ldots, d_{k-1}, \alpha_i(d), d_{k+1}, \ldots, d_p)$. Let us additionally consider the two sequences of darts $C_1 = (d_2 \ldots, d_{k-1})$ and $C_2 = (d_{k+1}, \ldots, d_p)$ such that $\sigma_i^*(d) = (d, C_1, \alpha_i(d), C_2)$. The loop $\alpha_i^*(d)$ surrounds either the vertices incident to $\alpha_i^*(C_1)$ or $\alpha_i^*(C_2)$. In order to differentiate these two configurations we say that d is the starting dart of the loop in the former case and the ending dart in the later one. Note that since $\alpha_i^*(d)$ defines a bridge in $\overline{G_i}$ both C_1 and C_2 define closed boundaries. Moreover, since G_i does not contain redundant edges, we have $d_2 \neq \alpha_i(d_{k-1})$ and $d_{k+1} \neq \alpha_i(d_p)$. Using equation 6, we obtain after some calculus:

$$or(\sigma_i^*(d)) = or(C_1) + or(C_2) - 4 \Rightarrow or(C_1) = -or(C_2) \tag{7}$$

Fig. 2. *One partition composed of 4 regions with two included ones*

```
1   list starting_dart(combi_map G_i, dart d_1) {
2       list L=∅
3       stack P
4       for each dart d_k in σ_i^*(d) = (d_1, ..., d_p){
5           if(d_k is a loop) {
6               if(P is empty or α_i(d_k) is not on the top of the stack P)
7                   push d_k and or_k in P
8               else {// α_i(d_k) on top of the stack P
9                   let C_1 be the sequence of darts between α_i(d_k) and d_k
10                  computes or(C_1) using equation 8
11                  if(or(C_1) == 4)  L = L ∪ {α_i(d_k)} else  L = L ∪ {d_k}
12              }
13      }
14      return L
15  }
```

Algorithm 1: *Determination of the starting darts of the loops*

where $or(\sigma_i^*(d))$ denotes the orientation of the whole sequence of darts $(d, d_2, \ldots, d_p)$. Since this sequence defines a counter clockwise traversal of the face its orientation is equal to -4 (Section 3).

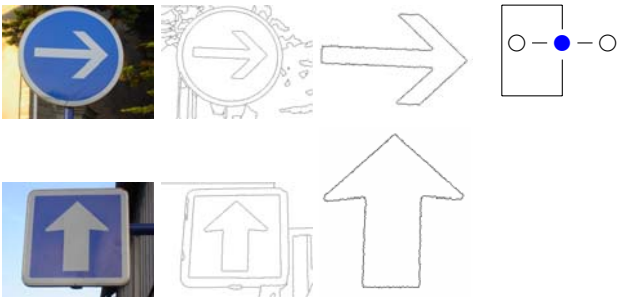
Equation 7 may be interpreted as follows: The loop $\alpha_i^*(d)$ corresponds to a bridge in $\overline{G_i}$ the removal of which splits the combinatorial map into two connected components. The component encoding the including face is traversed counter-clockwise and have thus an orientation equals to -4. On the other hand the remaining component corresponds to the included regions and has an opposed orientation equals to 4. Therefore, given the orientations of C_1 and C_2, d is the starting dart of the loop if the orientation of C_1 is equal to 4. Otherwise $\alpha_i(d)$ is the starting dart of the loop and d the ending one.

This last result is the basis of Algorithm 1 which traverses the σ_i cycle of a given vertex $\sigma_i^*(d_1) = (d_1, \ldots, d_p)$ and computes at each step the orientation of the sequence $d_1 \ldots, d_k$ denoted by or_k(equation 6).

Let us consider a dart d_k in $\sigma^*(d_1)$ such that $\alpha_i^*(d_k)$ corresponds to a loop and $\alpha_i(d_k) = d_j$ has been previously encountered ($j < k$). Then since the loops are nested d_j should be on the top of the stack. Using equation 6, if we denote by C_1 the sequence of darts between d_j and d_k, its orientation may be retrieved from or_k and or_j by the following formula:

$$or(C_1) = or_k - or_j - (Lm(d_j), Fm(d_{j+1}))\hat{} + (Lm(d_{k-1}), Fm(d_{j+1}))\hat{} \quad (8)$$

The darts d_j, d_{j+1} and d_{k-1} may be retrieved from the current dart d_k by: $d_j = \alpha_i(d_k)$; $d_{j+1} = \sigma_i(d_j)$ and $d_{k-1} = \sigma_i^{-1}(d_k)$. Given equation 8, the algorithm determines from the sign of $or(C_1)$ the starting dart of the loop between d_j and d_k (line 11). This starting dart is added to a list returned by the algorithm. Note that equation 8 is evaluated in constant time since or_k is the current orientation and or_j is retrieved from the stack.



(a) (b) (c) (d)

Fig. 3. Extraction of symbols within roadsigns using inside/outside information

Given the list of starting darts determined by Algorithm 1, the set of vertices included in $\sigma_i^*(d_1)$ is retrieved by traversing, the sequence $\sigma_i^*(d_1)$ from each starting dart to the corresponding ending one. By construction all darts encountered between the starting and ending darts of the loop encode adjacency relationships to included vertices. Note that in case of nested loops some loops may be traversed several times. Given a starting dart d, this last drawback may be avoided by replacing any encountered starting dart by its α_i successor during the traversal from d to $\alpha_i(d)$.

Our algorithm, is thus local to each vertex and the method may be applied in parallel to all the vertices of the combinatorial map G_i. Given a vertex $\sigma_i^*(d_1)$, the determination of its starting darts requires to traverse once $\sigma_i^*(d_1)$. Moreover, the determination of the inside relationships from the list of starting darts requires to traverse each dart of $\sigma^*(d_1)$ at most once. The worse complexity of our algorithm is thus equal to $\mathcal{O}(2|\sigma_i^*(d_1)|)$.

Fig. 3 illustrates one application of the inside/outside information to image analysis. The roadsign represented in Fig. 3(a) are composed of only two colors with one symbol inside a uniform background, the background itself being surrounded by one border with a same color than the symbol. In our example, the two roadsigns have a uniform blue background which includes one symbol representing a white arrow. The blue background is surrounded by a white border. In this application we wish to extract the sign of the roadsign using only topological and color information (and thus independently of the shapes of the symbol and the roadsign). Using only adjacency and color information, the symbol cannot be distinguished from the border of the roadsign since the border and the symbol have a same color and are both adjacent to the background of the roadsign (Fig. 3(d)). However, using inside/outside information, the symbol and the border may be distinguished since the background of the roadsign is adjacent to the border but includes the symbol. Our algorithm first builds a combinatorial pyramid using a hierarchical watershed algorithm [5]. Fig. 3(b) represents the top level of the hierarchies obtained from the two roadsigns. Using the top level combinatorial map of each pyramid our algorithm selects the k most blueish regions of the partition (k is fixed to five in our experiment). This last

step defines a set of canditate regions for the background of the roadsign. This background is then determined as the region whose included regions have the closest mean color from the color's symbol (equal to white in this experiment). Note that this step removes from the k selected candidates any regions which do not include another region. We thus explicit the a priori knowledge that the background of the roadsign should includes at least one region. The symbol is then determined as the set of regions included in the selected region (Fig. 3(c)). The symbol of the roadsign may thus be over segmented or composed of several disconnected regions. Finally, let us note that the includes information needs to be computed only on the k selected candidates for the roadsign's background. Within this experiment a global algorithm computing the includes information for all vertices would require useless calculus.

5 Conclusion

The method presented in this paper allows to get the set of regions inside in a given one. This method uses the orientation of the plane explicitly encoded by combinatorial maps and is particularly suited for algorithms using occasionally the inside information. Its worse complexity is equal to twice the number of edges incident to the vertex for which the inclusion relationships are computed. In our future work we plan to design a more global algorithm getting the inclusion information for all vertices.

References

1. J. P Braquelaire and J.P Domenger. Geometrical, topological, and hierarchical structuring of overlapping 2-d discrete objects. *Computers & Graphics*, 21(5):587–597, September 1997.
2. Luc Brun. *Traitement d'images couleur et pyramides combinatoires*. Habilitation à diriger des recherches, Université de Reims, 2002.
3. Luc Brun and Walter Kropatsch. Combinatorial pyramids. In Suvisoft, editor, *IEEE International conference on Image Processing (ICIP)*, volume II, pages 33–37, Barcelona, September 2003. IEEE.
4. Luc Brun and Walter Kropatsch. Receptive fields within the combinatorial pyramid framework. *Graphical Models*, 65:23–42, 2003.
5. Luc Brun, Myriam Mokhtari, and Fernand Meyer. Hierarchical watersheds within the combinatorial pyramid framework. In *Proc. of DGCI 2005*. IAPR-TC18, LNCS, 2005. to be published.
6. Walter G. Kropatsch. Building Irregular Pyramids by Dual Graph Contraction. *IEE-Proc. Vision, Image and Signal Processing*, Vol. 142(No. 6):pp. 366–374, December 1995.
7. P. Meer. Stochastic image pyramids. *Computer Vision Graphics Image Processing*, 45:269–294, 1989.
8. Annick Montanvert, Peter Meer, and Azriel Rosenfeld. Hierarchical image analysis using irregular tessellations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):307–316, APRIL 1991.

The GeoMap: A Unified Representation for Topology and Geometry

Hans Meine and Ullrich Köthe

Cognitive Systems Laboratory, University of Hamburg ,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{meine, koethe}@informatik.uni-hamburg.de

Abstract. We propose the GeoMap abstract data type as a unified representation for image segmentation purposes. It manages both topology (based on XPMaps) and pixel-based information, and its interface is carefully designed to support a variety of automatic and interactive segmentation methods. We have successfully used the abstract concept of a GeoMap as a foundation for the implementation of well-known segmentation methods.

1 Introduction

The goal of image segmentation is to identify regions that are conceptually coherent and serve as a basis for further analysis steps. Segmentation methods rely on local information on both direct properties of pixels and regions and the neighborhood. Today, computer vision researchers agree that correct handling of topology is needed when dealing with regions and boundaries, in order to avoid problems like the connectivity paradox.

Information on neighborhood-relations is conveniently stored in graph structures like the well-known *region adjacency graphs* (RAG [1]). These structures differ in expressiveness; some have problems with representing certain configurations occurring in image analysis (separate contours / holes, see e.g. [2]). Thus, a number of advanced formalisms for finite topology have been proposed for solving these problems [3, 2, 4, 5]. Another problem related to these graph structures is that usually, the geometry of the regions is stored separately (in so-called label images, edgel lists, or the like), and an algorithm has to modify both the graph and the external data when for example regions are merged. This puts the burden of preventing inconsistencies between the graph representation and the pixel geometry on the user (i.e. developer of the algorithm).

Furthermore, there are several possible definitions of regions and boundaries in discrete images - like crack edges, 8-connected boundaries between 4-connected regions or vice versa, or working with a hexagonal grid (some examples follow in Sect. 2, see Fig. 1 on page 135) - but they cannot be used interchangeably, since algorithms usually work directly on the pixel layer. We can generalize algorithms by formulating them on a higher abstraction level, and managing all relations between the topology and geometry on the pixel level in one abstract data type.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 132–141, 2005.

The GeoMap we introduce here will i) allow to work on a *natural abstraction level* with faces, edges, and vertices as basic entities (resulting in more concise, readable and reusable code), while ii) offering access to *both their neighborhoods and their associated pixels* at any time. This leads to *considerable advantages*: Having a common, unified representation for different automatic and interactive segmentation algorithms makes it possible to use them not only alternatively, but also together on one image. Furthermore, it facilitates the separation of the basic segmentation approach i) from the definition of topology on the pixel layer, but also from e.g. ii) cost definitions driving an optimization process, and thus allows to recombine parts from different publications.

2 The GeoMap Concept

As mentioned above, the GeoMap builds upon the XPMap formalism [5], and extends it by integrating the required geometrical information. We will now formally introduce the concept of a GeoMap, then carefully design an application interface suitable to exploit the advantages of our unified representation in Sect. 2.1, and finally propose a possible internal representation for our abstract data type (ADT) in Sect. 2.2. First, we need to define combinatorial maps.

Definition 1. *A* combinatorial map *is a triple* (D, σ, α) *where* D *is a set of* darts *(half-edges), and* σ, α *are permutations defined on* D *such that all* α *orbits have length* 2 *and the map is* connected*, i.e. there exists a* σ*-*α*-path between any two darts:*

$$\forall d_1, d_2 \in D\colon\ \exists \pi \in \left\{ \prod_{0 \le i \le k} \tau_i \,\middle|\, \tau_i \in \{\sigma, \alpha\},\ k \in \mathbb{N} \right\}\colon\ \pi(d_1) = d_2$$

The orbits of σ*,* α*, and the composed permutation* $\varphi = \sigma^{-1} \circ \alpha$ *are called* vertices*,* edges*, and* faces *respectively.*

A combinatorial map is *planar*, if and only if its number of vertices, edges, and faces fulfills Euler's equation ($|\alpha|$ denotes the number of orbits in α):

$$|\sigma| - |\alpha| + |\varphi| = 2 \tag{1}$$

An obstacle when trying to use planar combinatorial maps for image segmentation is that they cannot represent multiple boundary sets, which occur if we have regions with holes.

A common solution is to introduce auxiliary bridges which connect the contours (cf. [2]), but this complicates further handling, since i) algorithms working with edges have to explicitly check for these, and ii) there is no naturally defined place where these bridges should be attached to the contours. The latter becomes even more bothersome when we add geometrical information to the combinatorial structure. Then the auxilliary bridges also need geometric representations, which is unnecessary and may even be impossible if the geometry is defined with finite resolution as in our pixel-based approaches below. Furthermore, such

bridges are undesirable if they have to be distinguished from "real" bridges that represent incomplete boundaries information.

We avoid auxilliary bridges by means of the XPMap formalism [5]:

Definition 2. *We call a tuple* $(C, c_0, \mathsf{exterior}, \mathsf{contains})$ extended planar map (XPMap) *where* C *is a set of non-trivial planar combinatorial maps (the components of the XPMap),* c_0 *is a trivial map that represents the infinite face of the XPMap,* exterior *is a relation that labels one* φ*-orbit of each component in* C *as the exterior orbit, and* contains *is a relation that assigns each exterior orbit to exactly one non-exterior* φ*-orbit or to the infinite face of* c_0*.*

Note that an XPMap naturally defines permutations σ, α, and φ, which are simply the compositions of all permutations of the combinatorial maps in C.

XPMaps are a powerful representation for finite topology and suitable for image segmentation; however, segmentation algorithms are normally not entirely topology-based, but in general need to access the geometry and other (pixel-) properties of the boundaries and regions, such as brightness and gradient. Due to this important observation, we will now introduce the GeoMap.

Consider a complete partitioning of the plane into a set $\mathcal{P}$ of open regions that we call *basis cells* (which normally correspond to pixels or Khalimsky cells [6]). Furthermore, consider a relation $\mathsf{dim}\colon \mathcal{P} \to \{0, 1, 2\}$ that assigns a *dimension* to each basis cell. We then group connected basis cells of the same dimension into *block cells* according to the following rules (where $\mathcal{P}_d := \{p \in \mathcal{P} | \mathsf{dim}\,(p) = d\}$):

$$V := CC\left[\bigcup_{\mathcal{P}_0} p^c\right],\ E := CC\left[\left(\bigcup_{\mathcal{P}_1} p^c\right) \setminus \bigcup V\right],\ F := CC\left[\left(\bigcup_{\mathcal{P}_2} p^c\right) \setminus \left(\bigcup V \cup \bigcup E\right)\right]$$

where p^c denotes the closure of p and $CC\,[\ldots]$ is the set of connected components. These three types of block cells are called *vertices*, *edges*, and *faces* respectively. Fig. 1 shows some example $(\mathcal{P}, \mathsf{dim})$ pairs; these variants will be discussed in Sect. 2.2.

The *neighborhood* of a block cell c is defined as $N(c) := \{c_i \,|\, c \cup c_i \text{ is connected}\}$ where $c, c_i \in V \cup E \cup F$. Note that $N(c)$ will never contain cells $c_i \neq c$ of the same type as c, since the basis cells would have identical types and thus be combined into one connected component.

If all vertices and edges are simply connected (i.e. have no holes), and $\forall e \in E : ((|N\,(e)| \geq 3) \wedge (N\,(e) \cap V \leq 2))$ holds, we can represent the discrete topology of the block cells with an XPMap [7], and use this to build a GeoMap.

Definition 3. *A* GeoMap *is a tuple* $(\mathcal{P}, X, g)$ *where* $\mathcal{P}$ *is a set of basis cells,* X *is an XPMap that represents their induced topology, and* $g : V \cup E \cup F \to 2^{\mathcal{P}}$ *is a relation that assigns the set of contained basis cells to each block cell.*

The handling of both basis- and block cells is simplified by introducing labels: We require each basis cell p to have a unique label $b = \mathsf{label}\,(p)$ (usually, this will be the pixel coordinate) and assign unique labels l to the block cells. Note that we do not require the block cell labels to be continuous, since this leads to difficulties later with modifications which remove cells.

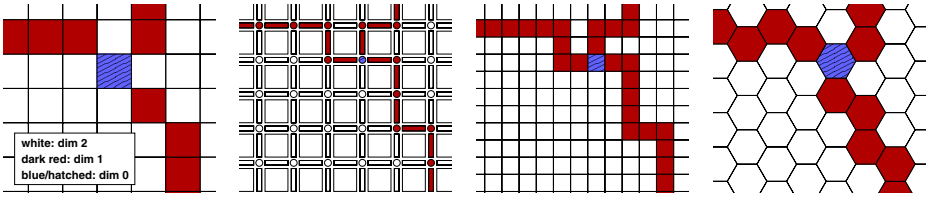


Fig. 1. Example GeoMap cells (left to right: 8-connected pixel boundaries, inter-pixel interpretation, explicit crack edges, example boundary on hexagonal pixel grid)

2.1 GeoMap Interface Design

In order to make the GeoMap a useful representation in practice, it is important that we define an abstract interface that reflects all requirements of segmentation algorithms. In [8] we systematically examined these needs of several algorithms; the results will be summarized in the following.

Topology Queries. There are several topology-related tasks that must be supported: Testing whether two regions are adjacent, querying all boundary components of a face, and listing all adjacent faces.

Cell Geometry Queries. Segmentation algorithms frequently need to know the shape of block cells, for example for collecting statistics on their basis cells' properties (i.e. boundary strength, mean region color).

Inverse Geometry Queries. Interactive segmentation requires a mapping from a specific basis cell (e.g. position obtained with a pointing device) to the block cell at that position.

Transformations. Considering segmentation as the transformation of an initial partitioning of the image plane into the desired result, we need operations like removing single edges or completely merging faces.

Application-Specific Data. Algorithms will rely on application-specific properties of the cells, for example to decide about which regions to merge. Thus, it must be possible to store and update this information in such a way that it is kept consistent with the current segmentation.

The last requirement is satisfied through the association of labels with each block cell, which can be used to index arrays with application-specific data; it is important however that these labels do not change in undefined ways. We will now explain solutions to the other tasks in detail, and then illustrate our implementation in Sect. 2.2.

Topology Queries: The DartTraverser Concept. Since the GeoMap is based on XPMaps, the basic entities used to encode its topology are *darts*. In order to make inspection of the topology as easy as possible, we introduce the DartTraverser concept, which uses a dart d to represent the current position during navigation within the GeoMap.

A GeoMap defines the permutations σ, α, and φ on its darts. The current position of a DartTraverser can be changed by moving to the successor or

predecessor of the current dart in σ (which corresponds to turning around the vertex), α (jumping to the opposite side of the edge), or the composed permutation $\varphi = \sigma^{-1} \circ \alpha$ (following the contour of the face to the left):

$$\begin{array}{lll} \text{nextSigma: } d := \sigma(d) & \text{nextAlpha: } d := \alpha(d) & \text{nextPhi: } d := \phi(d) \\ \text{prevSigma: } d := \sigma^{-1}(d) & \text{prevAlpha: } d := \alpha^{-1}(d) & \text{prevPhi: } d := \phi^{-1}(d) \end{array}$$

Now we do not only want to navigate on the darts, but we also want to access any information associated with the vertices, edges, or faces, so the DartTraverser interface also allows to query the identifying labels of the vertex which the current dart is attached to (represented by the orbit $\sigma^*(d)$), the edge it belongs to ($\alpha^*(d)$) and the face to the left ($\varphi^*(d)$).

$$\begin{array}{rl} \text{startNodeLabel: } d \mapsto \mathsf{label}(\sigma^*(d)) & \text{endNodeLabel: } d \mapsto \mathsf{label}(\sigma^*(\alpha(d))) \\ \text{edgeLabel: } d \mapsto \mathsf{label}(\alpha^*(d)) & \\ \text{leftFaceLabel: } d \mapsto \mathsf{label}(\phi^*(d)) & \text{rightFaceLabel: } d \mapsto \mathsf{label}(\phi^*(\alpha(d))) \end{array}$$

Geometry Queries. As stated above, there need to be means to i) get the block cell associated with a given basis cell or to ii) query all basis cells belonging to one block cell. In order to answer the first question, the GeoMap offers

$$\text{cellAt: } b \mapsto l$$

which returns the label l of the block cell for which $g(l)$ contains the basis cell labelled b. The second task - finding all basis cells belonging to a cell - is usually closely related to collecting properties of these basis cells (e.g. finding the mean color, inspecting the gradient, calculating the center of mass). This can be accomplished by querying the GeoMap for a CellScanIterator:

$$\text{cellScanIterator: } l \mapsto \text{CellScanIterator}(\{\mathsf{label}(p) \,|\, p \in g(l)\})$$

This CellScanIterator is then used to iterate over the basis cell labels, which are needed in order to look up the properties for the corresponding cells.

Transformations. Since image segmentation is a dynamic process, the GeoMap would be useless without means for modification. An important design decision for this part of the interface is that the GeoMap should offer a small set of simple transformations, which makes formal correctness proofs possible and ensures that the representation stays in a consistent state. Non-admissible transformations can be rejected by checking the preconditions of each operation.

Köthe [5] proposes a set of Euler Operators [9] for image segmentation; these are operators that leave Euler's equation on the number of cells and connected boundary components $|C|$ in a planar XPMap valid: $|V| - |E| + |F| - |C| = 1$. We define the following operations on the GeoMap G, which all take the form $G, d \mapsto G'$ and return a $G' = (\mathcal{P}, X', g')$ with X' and g' created from X and g as follows:

merge edges merge the two edges $\alpha^*(d)$ and $\alpha^*(\sigma(d))$ and the vertex $\sigma^*(d)$ (must have degree 2) into one single edge $(|V'| = |V| - 1,\ |E'| = |E| - 1)$
remove bridge merge the edge $\alpha^*(d)$ (which must be a bridge) into the surrounding face $\varphi^*(d)$ $(|E'| = |E| - 1,\ |C'| = |C| + 1)$
remove isolated vertex merge an isolated vertex represented by the empty orbit $\alpha^*(d)$ into the surrounding face $\varphi^*(d)$ $(|V'| = |V| - 1,\ |C'| = |C| - 1)$
merge faces merge the two faces $\varphi^*(d)$ and $\varphi^*(\sigma(d))$ (must not be identical) and their common edge $\alpha^*(d)$ into one face $(|E'| = |E| - 1,\ |F'| = |F| - 1)$

The relation g' is derived from g by assigning the basis cells of all cells being merged to the resulting cell.

Note that each of the above operations is a reduction (reducing the number of cells). Conceptually, they all have inverse operations that could be used to e.g. split block cells, effectively creating new ones from the same basis cells. Additionally, split operations could be applied on basis cells, which would change $\mathcal{P}$. However, adding geometric information introduces an asymmetry between split and merge operations (the former are no longer parametrizable with a single dart), which is why split operations are beyond the scope of this paper.

Note that the GeoMap handles both updating the geometry and the topology, but the application-specific information on the cells has to be updated by the application. Usually, it is possible to combine the statistics of the cells being merged in order to get the statistics of the resulting cell, and a GeoMap implementation can provide hooks for callbacks to ensure that this happens.

Building GeoMap Pyramids. We consider segmentation as the transformation of an initial partitioning of the image plane into the desired result. Usually, the initial tessellation is an oversegmentation as resulting from a watershed transform or optimal cut [10], or the trivial one where every pixel is a separate region (i.e. the first segmentation step is to look for *any* boundary evidence). In this setting, further segmentation stages can be computed by using the above reduction operators, and one can arrange the results over time in a pyramid, where each level contains less cells than the one below. This corresponds to the approach of irregular pyramids [3, 2, 4], which can be used to create more coarse, abstracting segmentations without losing the ability to represent important detail.

2.2 CellImage Realization

So far, we have concentrated on the abstract properties of a GeoMap; now we focus on a possible implementation.

We propose a straight-forward extension of the common label images as internal representation for a GeoMap: The geometry information is stored in a CellImage, the pixels of which are the basis cells and each carry a dimension (specifying their type - vertex, edge, or face pixel) and the label of the block cell they belong to. The complete topological information is derived from this internal representation (see Fig. 2) according to Definition 3, which offers consistent views on the same segmentation from both perspectives.

The relation dim (page 134) is crucial for the correct derivation of topology from the basis cells (i.e. pixels). In the past, researchers have concentrated on

crack edge-based interpretation of region images (inter-pixel boundaries, [11, 4, 1]), but it has also been shown that a topological representation can be derived from "thin", 8-connected boundaries (resulting from a watershed segmentation for example) [7]. Note that inter-pixel contours are commonly made explicit by doubling the image size and inserting boundary pixels, but the resulting 4-connected contours are visually much less appealing than 8-connected contours due to strong staircase effects. On the plus side, inter-pixel nodes always consist of one basis cell and have limited degree, which makes crack edge contours much easier to use. Definition 3 allows for both inter-pixel contours and explicitly represented ones on square or hexagonal pixel grids. We will concentrate on the interesting case of 8-connected boundaries in the following, since it has not yet received as much attention as the inter-pixel approaches.

Moving in the Orbits. Lets have a look at how navigation through the topology works in some examples. Our internal representation of a DartTraverser is a pixel position and a direction, pointing to one of the pixel's neighbors (indicated by the arrows in Fig. 2).

α-orbit: Finding the opposite half-edge is accomplished by simply following the edge pixel-wise to the next vertex pixel and turning around. Both crack-edges and the boundary pixel classification by Köthe [7] guarantee by definition that each edge pixel has a unique successor and predecessor.

σ-orbit: Finding the next dart in the σ-orbit of a Khalimsky vertex is straightforward, since its degree is limited to the number of four direct neighbors. In the case of 8-connected boundaries it involves a more complex procedure: Here, vertices can consist of more than one pixel, which means that their contour has to be followed in order to find the σ successor (cf. Fig. 2 right).

Giving Access to the Geometry. Section 2.1 introduced the geometry-related part of the GeoMap ADT, notably the CellScanIterator, which allows to iterate over the labels of all basis cells (pixels) of a given block cell (cf. g in Def. Definition 3). It can be efficiently realized by scanning the internal CellImage (restricted to the cached bounding box of the block cell) and stopping

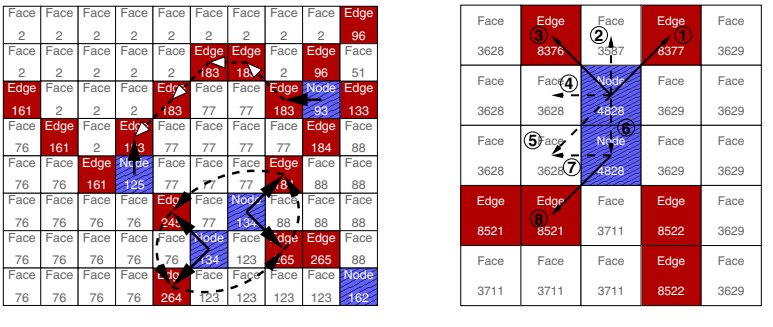


Fig. 2. *Left*: Two DartTraversers cycling through their α- and σ-orbits, respectively. *Right*: Detailed series of intermediate states for finding two σ successors

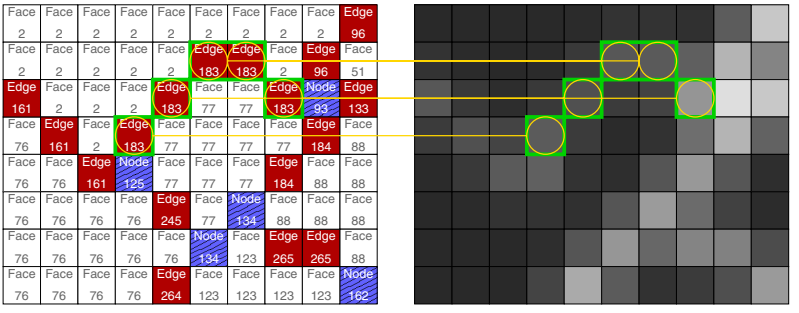


Fig. 3. CellScanIterator scanning edge 183 in a gradient magnitude image

at pixels belonging to the cell being queried. At each step, the iterator returns the basis cell's label (i.e. its position), which is then used to look up properties in any application-specific image, for example to find the mean color of a region in the original image, or the gradient estimates on a given edge (cf. Fig. 3).[1]

3 Application

The GeoMap is designed to serve as a versatile representation for many segmentation algorithms. We have employed it to implement a variety of approaches, some of which we will describe here.

Canny Hysteresis. Canny's segmentation approach is undoubtedly the most well-known one, and its steps are still representative for its state-of-the-art descendants: After collecting initial evidence for edges (the initial oversegmentation in our case), the candidate set is filtered to get the final result. We implemented this hysteresis thresholding on the basis of the edges in our GeoMap. However, we are not limited to assess the edges based on gradient information, but also implemented measures based on the adjacent faces (e.g. difference of their mean colors, T-test, ...).

Contraction Kernels. The GeoMap allowed us to implement irregular pyramids as in [2] by grouping a set of Euler Operations into complex contractions. Furthermore, the CellScanIterator made it easy to implement (e.g. color-based) salience measures to define the contraction kernels.

Active Paintbrush. In contrast to the above, this tool relies entirely on human interaction; is allows to "paint over" region boundaries to initiate region merge operations [12]. It is very useful to interactively mark fine structure in low-contrast images (e.g. angiography). Since our framework is based on one common representation, it is also possible to use the paintbrush to correct errors made

[1] A more efficient implementation directly scans the target image in parallel, making the indirection via the position unnecessary.

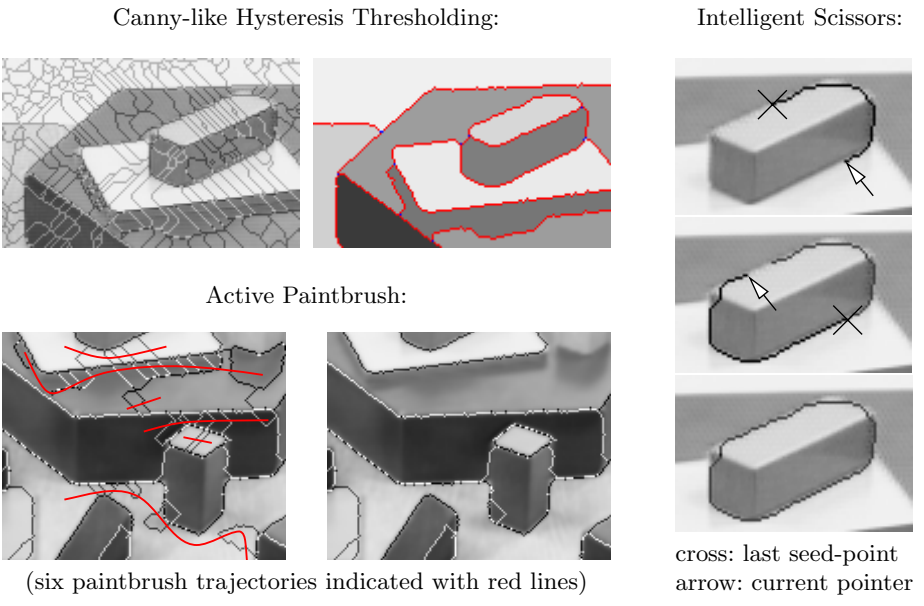


Fig. 4. Example screenshots showing the tools in action

by the other, automatic tools. To facilitate this, we augmented it with a means to protect the boundary of single regions from being changed.

Intelligent Scissors. After the selection of an initial seed point on a contour, this semi-interactive tool highlights the optimal path to the current pointer position in real-time with a *live-wire* [13]. A complete contour can be delineated with only a few additional selections. In order to define the optimal path, we measure and combine the significance of single edges - another example where the abstraction level of the GeoMap formalism led to directly reusable components, namely the cost measures from the hysteresis tool.

Implementing these algorithms based on the GeoMap formalism means to abstract from the boundary definition. We have used all these algorithms with both a crack edge representation and 8-connected thin boundaries [7] in an irregular pyramid, whose level 0 contained a watershed oversegmentation. This conforms to the recent approach of starting with *superpixels* [10], not pixels directly.

Note that our experimental results prove that it is possible to achieve this level of abstraction not only without losing flexibility, but that generic programming techniques allow for very efficient realizations of the formalism. For example, the GeoMap of a 640×480 image initially segmented into 11.161 vertices, 17.930 edges, and 6.775 faces can automatically be reduced (based on face statistics) into a final result with about 30 regions in 3.6 seconds on a Pentium III notebook with 800 MHz.

4 Conclusion

The GeoMap formalism demonstrates that the integration of topology and geometry in one unified representation leads to a versatile basis for image segmentation. Adapting algorithms to this framework leads to concise, comprehensible code and does not sacrifice speed. At the same time, the GeoMap introduces a level of abstraction that facilitates the decomposition of published segmentation methods and (re-)combination of approaches, i.e. interchange edge salience definitions or apply several algorithms on the same image.

In the future, we want to extend the GeoMap formalism to work with other (subpixel- or 3D) boundary definitions and add split operations and means to refine the contours retroactively. On the application side, we are currently working on the integration of learning methods and more sophisticated edge salience measures based on boundary continuity.

References

1. Pavlidis, T.: Structural Pattern Recognition. Springer (1977)
2. Kropatsch, W.G.: Building irregulars pyramids by dual graph contraction. IEEE-Proc. Vision, Image and Signal Processing **142** (1995) 366–374
3. Montanvert, A., Meer, P., Rosenfeld, A.: Hierarchical image analysis using irregular tessellations. In: T-PAMI. Volume 13., IEEE (1991) 307–316
4. Brun, L., Kropatsch, W.: Introduction to combinatorial pyramids. In G. Bertrand, A. Imiya, R.K., ed.: Digital and Image Geometry. Volume 2243 of LNCS. Springer Verlag (2001) 108–127
5. Köthe, U.: XPMaps and topological segmentation - a unified approach to finite topologies in the plane. In Braquelaire, A., Lachaud, J.O., Vialard, A., eds.: 10th Intl. Conference on Discrete Geometry for Computer Imagery (DGCI 2002). Volume 2310 of LNCS., Berlin, Springer (2002) 22–33
6. Khalimsky, E., Kopperman, R., Meyer, P.R.: Computer graphics and connected topologies on finite ordered sets. Topology and its Applications **36** (1990) 1–17
7. Köthe, U.: Deriving topological representations from edge images. In Asano, T., Klette, R., Ronse, C., eds.: Geometry, Morphology, and Computational Imaging. Volume 2616 of LNCS., Berlin, Springer (2003) 320–334
8. Meine, H.: XPMap-based irregular pyramids for image segmentation. Diploma thesis, Dept. of CS, University of Hamburg (2003)
9. Mäntylä, M.: An Introduction to Solid Modeling. Computer Science Press (1988)
10. Ren, X., Malik, J.: Learning a classification model for segmentation. In: Proc. of the Tenth Intl. Conf. On Computer Vision (ICCV-03). Vol. 1., IEEE (2003) 10–16
11. Braquelaire, J.P., Domenger, J.P.: Representation of region segmented images with discrete maps. Technical Report 1127-96, Université Bordeaux, Laboratoire Bordelais de Recherche en Informatique (1996)
12. Maes, F.: Segmentation and Registration of Multimodal Images: From Theory, Implementation and Validation to a Useful Tool in Clinical Practice. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium (1998)
13. Mortensen, E.N., Barrett, W.A.: Toboggan-based intelligent scissors with a four parameter edge model. In: Computer Vision and Pattern Recognition. Vol. 2., IEEE (1999) 452–458

Pyramids of n-Dimensional Generalized Maps

Carine Grasset-Simon, Guillaume Damiand, and Pascal Lienhardt

SIC, FRE-CNRS 2731 - Université de Poitiers,
bât. SP2MI, Bvd M. et P. Curie,
BP 30179, 86962 Futuroscope Chasseneuil Cedex - France
{simon, damiand, lienhardt}@sic.univ-poitiers.fr

Abstract. Graph pyramids are often used for representing irregular pyramids. Combinatorial pyramids have been recently defined for this purpose. We define here pyramids of n-dimensional generalized maps. This is the main contribution of this work: a generic definition in any dimension which extend and generalize the previous works. Moreover, such pyramids explicitly represent more topological information than graph pyramids. A pyramid can be implemented in several ways, and three representations are discussed in this paper.

Keywords: Hierarchical data structure, irregular pyramids, generalized map, n-G-map pyramids, multi-resolution.

1 Introduction

Hierarchical structures are often used for representing a same object with different resolutions. The first level of such a structure generally represents the object with a very fine precision, then the precision decreases as the level increases. For image processing, pyramids are used for representing different segmentation levels of an image. It is thus possible to get immediately the segmentation level suited for a particular process. It is also possible to modify a pyramid level, for instance when changing the segmentation criteria.

Irregular image pyramids have been studied by many authors, and used for several applications ([1, 2, 3, 4, 5, 6, 7, 8]). An irregular pyramid is defined as a stack of reduce graphs where each graph is built from the precedent level by a sampling or decimation process. Brun and Kropatsch have extended this notion by defining pyramids in which each level is a 2-dimensional combinatorial map ([9, 10, 11]). Combinatorial maps represent the topology of any subdivisions of any orientable surfaces without boundary.

We extend these works for any dimension by defining pyramids of n-dimensional generalized maps (or n-G-maps). This is the main result of this paper. Generalized maps represent the topology of any subdivisions of any orientable or not orientable n-dimensional manifolds with or without boundary (see [12, 13]). So, pyramids of n-G-maps can be used in order to process images in 2D, 3D, as well as 4D (for instance for tracking objects in 3D image sequences). An

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 142–152, 2005.

other interest of generalized maps is that their definition is homogeneous for any dimensions. This facilitates the definition of generic operations and algorithms.

Our work follows that of Damiand and Lienhardt who have defined for n-G-maps a general operation for removing and contracting cells of any dimensions [14]. More precisely, each level of a pyramid is a simplification of the previous one, obtained by applying this operation.

We recall in section 2 the notion of generalized map as well as the operation of cell contraction and removal. Generalized map pyramids are defined in section 3. We study in section 4 three different representations of generalized map pyramids. We discuss in section 5 the construction of a pyramid in the context of image segmentation. Generalized map pyramids and graph pyramids are compared in section 6. Further issues are discussed in section 7.

2 Recalls: n-G-Map, Cell Removal and Contraction

n-dimensional generalized maps (or n-G-maps) represent the topology of n-dimensional subdivided objects, and more precisely the topology of quasi-manifolds (see [12, 15, 13]). An n-G-map is a set of abstract elements (darts), together with applications defined on these darts. Cells are implicitly represented as sets of darts. (see [13] for more details).

Definition 1 (n-G-map and i-Cell (cf. Figure 1)). *Let $n \geq 0$.*
An n-dimensional generalized map G is defined by $G = (D, \alpha_0, \ldots, \alpha_n)$ where:

1. *D is a finite set of darts;*
2. *$\forall k,\ 0 \leq k \leq n$, α_k is an* involution[1] *on D;*
3. *$\forall k, j,\ 0 \leq k < k+2 \leq j \leq n$, $\alpha_k \alpha_j$ is an involution.*

Let $d \in D$, $N = \{0, 1, \ldots, n\}$ and let i be such that $0 \leq i \leq n$.
The i-*cell incident to d is the orbit* [2]

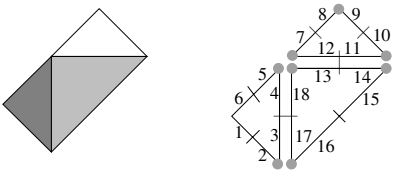
$$<>_{N-\{i\}}(d) =< \alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n > (d).$$

In order to define n-G-map pyramids, Damiand and Lienhardt have proposed an operation for simultaneously removing and contracting cells of any dimensions [14](cf. figure 2). These cells have to satisfy two preconditions: they are disjoint two by two[3], and their degree is two.

[1] An involution f on a finite set S is a one to one mapping from S onto S such that $f = f^{-1}$.

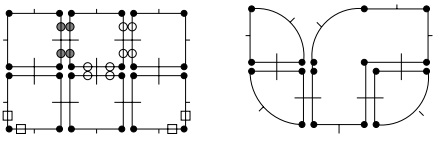
[2] Let $\{\Pi_0, \ldots, \Pi_n\}$ be a set of permutations on D. The orbit of an element d related to this set of permutations is $< \Pi_0, \ldots, \Pi_n > (d) = \{\Phi(d), \Phi \in < \Pi_0, \ldots, \Pi_n >\}$, where $< \Pi_0, \ldots, \Pi_n >$ denotes the group of permutations generated by $\Pi_0, \ldots, \Pi_n$.

[3] The set of i-cells is a partition of the set of darts of the n-G-map, for each i between 0 and n. Two cells are disjoint if their intersection is empty, i.e. when no dart is shared by the cells.



(a) (b)

Fig. 1. (a) A subdivision of a surface. (b) The corresponding 2-G-map. Darts are represented by numbered black segments. Two darts in relation by α_0 share a little vertical segment (ex. darts 1 and 2). Two darts in relation by α_1 share a same point (ex. darts 2 and 3). Two distinct darts in relation by α_2 are parallel and close to each other (ex. darts 3 and 17); otherwise, the dart is its own image by α_2 (ex. dart 2). The vertex incident to dart 2 is $< \alpha_1, \alpha_2 > (2) = \{2, 3, 16, 17\}$, the edge incident to dart 3 is $< \alpha_0, \alpha_2 > (3) = \{3, 4, 17, 18\}$, and the face incident to dart 9 is $< \alpha_0, \alpha_1 > (9) = \{7, 8, 9, 10, 11, 12\}$



a b

Fig. 2. An example of simultaneous removal and contraction of cells of different dimensions. (a) A 2-G-map where the darts of removed 0-cells, removed 1-cells and contracted 1-cells are respectively marked by empty squares, circles and gray disks. (b) The resulting 2-G-map

For $i \in N$, R_i is the set of removed i-cells, and C_i is the set of contracted i-cells. Note that $R_n = \emptyset$ and $C_0 = \emptyset$ since it is not possible to remove n-cells nor to contract 0-cells (see [14] for more details).

3 Definition of n-G-Map Pyramids

An n-G-map pyramid is a hierarchical structure. Each level is an n-G-map: the first level represents the initial data; the other levels represent successive simplifications. A possible bottom-up construction is the following one. Level 0 is an n-G-map which represents the initial data. According to the application, a process will define two sets for any dimension i ($i \in N$): R_i^0 (resp. C_i^0) is the set of i-cells which will be removed (resp. contracted). The cells of these sets have to satisfy the preconditions of the removal and contraction operation. The application of this operation gives the level 1 map. This process is reapplied in order to get the other levels of the pyramid.

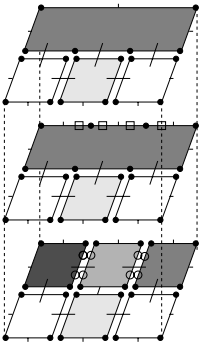


Fig. 3. A 2-G-map pyramid composed of three levels. The darts of removed 0-cells (resp. 1-cells) are marked by empty squares (resp. circles)

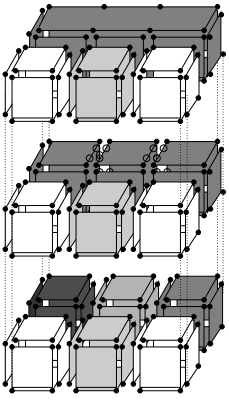


Fig. 4. A 3-G-map pyramid composed of three levels. The second level is obtained by removing 2 faces, and the third level is obtained by removing 4 edges (the two upper edges and the two edges in the background which are adjacent to the first ones)

The formal definition of an n-G-map pyramid is the following:

Definition 2 (n-G-map Pyramid). *Let $n,\ m \geq 0$. A $m+1$ level pyramid $\mathcal{P}$ of n-dimensional generalized maps is the set $\mathcal{P} = \{G^k\}_{0 \leq k \leq m}$ where:*

1. *$\forall k,\ 0 \leq k \leq m$, G^k is the n-G-map $(D^k, \alpha_0^k, \ldots, \alpha_n^k)$,*
2. *For each k, $0 \leq k \leq m$, for each i, $0 \leq i < n$, let R_i^k (resp. C_i^k) be sets of i-cells such that: cells are disjoint two by two and the degree of each cell is equal to 2, i.e.:*
   - *$\forall C,\ C' \in \bigcup_{i=0}^{n}(R_i^k \cup C_i^k)$, $C \cap C' = \emptyset$,*
   - *$\forall i,\ 0 \leq i \leq n-2$, $\forall d \in R_i^k$, $d\alpha_{i+1}^k\alpha_{i+2}^k = d\alpha_{i+2}^k\alpha_{i+1}^k$,*
   - *$\forall i,\ 2 \leq i \leq n$, $\forall d \in C_i^k$, $d\alpha_{i-1}^k\alpha_{i-2}^k = d\alpha_{i-2}^k\alpha_{i-1}^k$,*
3. *$\forall k,\ 0 < k \leq m$, G^k is obtained from G^{k-1} by removing the cells of $\cup_{i=0}^{n} R_i^{k-1}$ and contracting the cells of $\cup_{i=0}^{n} C_i^{k-1}$.*

Examples of 2D and 3D pyramids are provided in figures 3 and 4.
Two major properties of n-G-map pyramids are:

- each dart which belongs to a removed or a contracted cell of level k does not belong to another removed or contracted cell in the same level or in another level.
- Let k, $0 \leq k < m$. Note that a one to one mapping φ^k exists between the surviving darts of G^k (i.e. the darts which are not removed nor contracted),

and the darts of G^{k+1} ($\varphi^k : D^k - \bigcup_{i=0}^{n}(R_i^k \cup C_i^k) \longrightarrow D^{k+1}$). In order to simplify, we will denote a dart of G^k and its image in G^{k+1} by the same name. So, we have: $D^{k+1} = D^k - \bigcup_{i=0}^{n}(R_i^k \cup C_i^k)$.

More formally:

Proposition 1.

1. $\forall i, j \in N$, $\forall k, l \in [0..m-1]$ *we have:* $\begin{cases} R_i^k \cap C_j^l = \emptyset, \\ R_i^k \cap R_j^l = \emptyset, \ \textit{with } i \neq j \ \textit{or } k \neq l, \\ C_i^k \cap C_j^l = \emptyset, \ \textit{with } i \neq j \ \textit{or } k \neq l. \end{cases}$
2. $\forall k$, $0 \leq k < m$, $D^{k+1} \subseteq D^k$.

These properties can be easily deduced from the definition of the removing and contracting operation [14], and from the definition of n-G-map pyramids. Moreover these properties are useful for the definition of representations of n-G-map pyramids, and more precisely for the *implicit* and *hierarchical* representations.

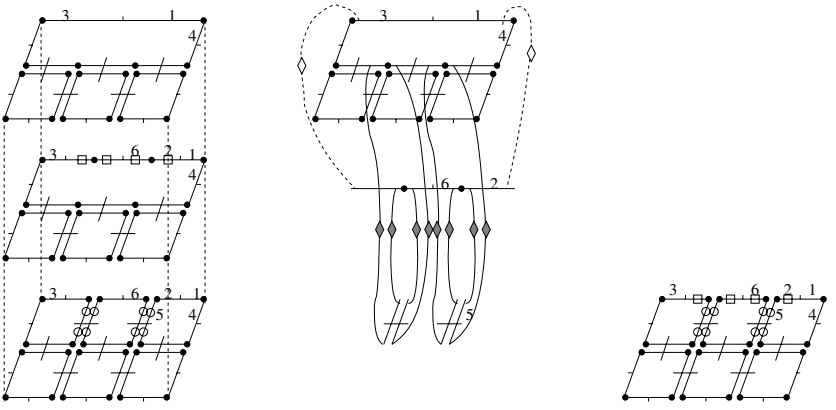
4 Different Representations of n-G-Map Pyramids

An n-G-map pyramid can be represented more or less explicitly according to the performance expected in space and in time. We present in this section three possible representations: *explicit*, *hierarchical* and *implicit* (see figure 5). Note that each representation has to satisfy the constraints of the definition of n-G-map pyramids, which can be easily expressed for each representation. Note also that these three representations contain the same information, but each of them has different advantages and drawbacks (explain in the following).

Explicit n-G-Map Pyramid: each level k (i.e. G^k) and each bijection φ^k are explicitly represented (see figure 5-a). The representation contains thus $m+1$ n-G-maps, and each dart is linked with its predecessor (except the darts of level 0) and with its successor[4] (except the darts of the last level and the darts which belong to a removed or contracted cell). Moreover, for each level, a dart which belongs to a removed or contracted cell in this level is marked by the type of the operation (contraction or removal) and the dimension of the cell.

Hierarchical n-G-Map Pyramid: $D^k - \bigcup_{i=0}^{n}(R_i^k \cup C_i^k)$ is identified with D^{k+1} for each k ($0 \leq k < m$) but each involution α_i is explicitly represented ($\forall i$, $0 \leq i \leq n$). This representation (see figure 5-b) contains a single set of darts, and for each dart one table which represents involutions α for all levels (the size of this table is the product of the number of involutions by the number of levels for each dart). If a dart disappears at level k, its images by involutions α are only defined from level 0 to level k. A possible optimization is the following: when $d\alpha_i^k = d\alpha_i^{k+1}$, the image of d is represented only once. More precisely, given i

[4] The predecessor relation corresponds to $(\varphi^{k-1})^{-1}$ and the successor relation φ^k.



		α_0	α_1	α_2
level 0	1	2	4	1
	2	1	5	2
level 1	1	2	4	1
	2	1	6	2
level 2	1	3	4	1

a

	level 0			*level* 1			*level* 2		
	α_0	α_1	α_2	α_0	α_1	α_2	α_0	α_1	α_2
1	2	4	1	2	4	1	3	4	1
2	1	5	2	1	6	2	–	–	–

b

	α_0	α_1	α_2
1	2	4	1
2	1	5	2

c

Fig. 5. Three representations of a same pyramid. For each representation, the corresponding array shows images by involutions α for the two darts 1 and 2. (a) *Explicit*. Removed 0-cells (resp. removed 1-cells) are marked by empty squares (resp. circles). (b) *Hierarchical*. Each dart is drawn in the last pyramid level where it exists. When two darts linked by α_i are drawn in the same level, their link α_i is drawn in the usual way. Otherwise, the links α_0^1 and α_0^0 between two darts of two different levels are represented by lines with empty lozenge and the link α_1^0 by line with filled lozenge. (c) *Implicit*. Removed 0-cells of level 0 (resp. removed 1-cells of level 1) are marked by empty squares (resp. circles)

$(0 \leq i \leq n)$ and a dart d, we only memorize distinct images of d by α_i, and for each dart, the last level at which it belongs.

Implicit n-G-Map Pyramid: this representation contains only the first level (i.e. G^0) and three marks are associated with each dart. The first one corresponds to the type of operation (removal or contraction) which suppresses the dart (if the case arises). The second one corresponds to the dimension of the removed or contracted incident cell. The third one indicates the level at which the cell disappears.

The choice of one representation depends on the particular needs of the application, since these representations offer different advantages and drawbacks.

The *explicit* representation is characterized by an important redundancy of information, and thus an heavy cost in memory space. On the other hand, each pyramid level can be directly accessed. So, the extraction of a particular level or the modification of the pyramid can be easily achieved.

The *hierarchical* representation is less costly in memory space, since there are less redundant informations. Any level can be directly extracted by using involutions of this level. Moreover, given a cell, one can directly access to the set of cells of a lower level which are "merged" into this cell. This can be useful for example for the representation of different levels of details (see [16] where a data structure, based upon a similar principle, is proposed in order to model complex architectural environments). On the other hand, it is usually difficult to modify the pyramid, since all levels are not explicitly represented. More precisely, it is difficult to propagate modifications between the pyramid levels.

The *implicit* representation has an optimal complexity in memory space since there is no redundant information. Moreover, the pyramid can be directly modified, avoiding the problem related to the information propagation. A similar representation is proposed by Brun and Kropatsch [17] for representing a 2D combinatorial pyramid. The main drawback of this representation is that a pyramid level can not be directly accessed: it is necessary to compute it when required.

5 Construction of a Pyramid for Image Segmentation

For image processing, pyramids are used in particular in order to keep in memory different segmentations of a same image. In order to construct an n-G-map pyramid associated with a multi-level segmented image, level 0 is associated with the initial image. A new level is constructed through two steps: first, the cells of the "previous segmentation" corresponding to homogeneous regions are "merged" into one cell; second, the resulting n-G-map is simplified.

An example is the following. We want to segment a 3D gray level image by using a simple gray level distance as homogeneity criterion. The initialization consists in associating a 3-G-map with the initial image (cubic volumes are associated with voxels). A label representing the gray level of the corresponding voxel is associated with each volume. The voxels corresponding to a homogeneous region are merged. This is achieved by removing the faces which are between the corresponding volumes. More precisely, a process based upon the homogeneity criterion marks level 0 faces, then the 3-G-map is duplicated and the removal and contraction operation is applied, producing level 1 map. The same principle is applied in order to compute the following levels. It is possible to modify the marking process in order to control the topology of the 3-G-maps, for instance for avoiding disconnections. In order to reduce the required memory space, the representation of the boundary between two regions can be simplified, by merging the boundary faces into one face. This simplification can be achieved by removing

the degree 2 edges and then the degree 2 vertices. Other simplifications can be made in order to obtain a minimal representation (see [18, 19]). A new level is constructed by the same two phases: first marking, duplication and removal of faces; then simplifications by removing degree 2 edges and vertices.

For 2D images, pixels which make a homogeneous region (i.e. faces of the 2-G-map) are merged by removing the edges between these faces. The 2-G-map is then simplified by removing the degree 2 vertices. For 4D images, 4D cells are merged by removing the volumes which lie between them. The boundary between two regions can be simplified by removing the degree 2 faces, then the degree 2 edges and then the degree 2 vertices. More simplifications can be done depending on the particular needs for each application.

6 Comparison with Graph Pyramids

Irregular image pyramids are often represented by adjacency graph pyramids. A vertex of such an adjacency graph corresponds to a region of the image and an edge symbolizes the adjacency relation between the two regions associated with the extremity vertices. Figure 6 shows different representations of a segmented image. Note that one region is included into an other one, and that two regions are adjacent several times to each other. An adjacency graph represents all types of adjacency in the same way (figure 6-b), leading to a loss of information.

Dual graph pyramids [3] are an extension of adjacency graph pyramids, designed in order to take into account multi-adjacency and inclusion (see figure 6-c). A dual graph is defined as a multi-graph together with its dual graph (these two graphs have to be connected). However, dual graphs as well as adjacency graphs can not represent the topological order information for all cases (i.e. for instance the order of faces around a vertex or the order of volumes around an edge or a vertex). For example, figure 7 shows a gray level image representing a clover. It is composed by six parts: the background, the stem, three leaves and the roots (see figure 7-a). The corresponding dual graph is shown in figure 7-b. Due to the self-loop, the edge orientation around vertices is loss and it is not possible to know which leaf lies between the two others. On the other hand, n-G-maps, and so n-G-map pyramids, represent inclusion, multi-adjacency and order, for any dimension (cf. figure 6-d and figure 7-c).

An other drawback of adjacency graphs is the fact that all cells are not represented. For instance in 2D (resp. 3D), vertices (resp. edges and vertices) are not represented. More generally, adjacency graphs only represent n-cells and $n-1$-cells. n-G-maps represent all cells of any dimension, and all incidence and adjacency relations between these cells. Figure 8 shows the example of a 3D pipe, which extremities share only one edge. The corresponding adjacency graph contains only two vertices (the pipe and the surrounding region) linked by one edge. So, it is not possible to retrieve the information about the two extremities of the pipe. This can be done using a 3-G-map.

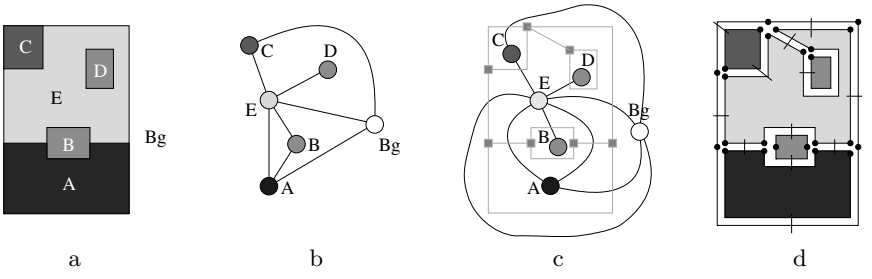


Fig. 6. Representation of a segmented image. (a) An image. (b) The corresponding adjacency graph. (c) The corresponding dual graphs (primal graph in black and its dual in gray). (d) The corresponding 2-G-map

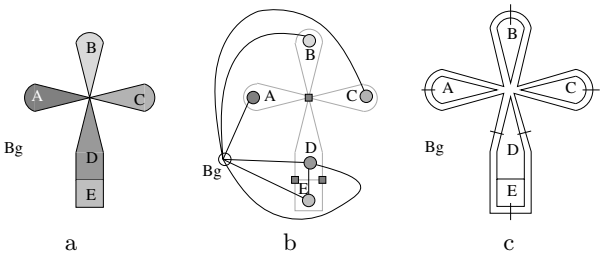


Fig. 7. A 2D image of a clover. (a) The image. (b) The corresponding dual graphs. (c) The corresponding 2-G-map

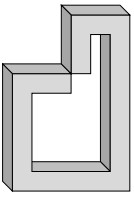


Fig. 8. A 3D image representing a pipe

7 Conclusion and Perspectives

Pyramids of n-dimensional generalized maps are here defined as stacks of reduce n-G-maps where each n-G-map is built from the previous level by contracting or removing cells. n-G-maps unambiguously represent the topology of subdivided n-dimensional objects (for instance n-dimensional images). So, n-G-map pyramids can be used in order to process 2D, 3D and 4D images. n-G-map pyramids can be very useful for applications in which it is necessary to check or to control the (evolution of the) topology of an object, for example when tracking objects in video sequences.

n-G-map pyramids can be represented in different ways. We have discussed three generic representations: *explicit*, *hierarchical* and *implicit*, and their advantages and drawbacks which are important for choosing an efficient representation according to the needs of the application (complexity in memory space and/or in time). But we need to compare more precisely these three representations in practical applications to list advantages, drawbacks and complexities.

n-G-map pyramids have several advantages, compared with adjacency graph pyramids. Mainly, n-G-maps pyramids represent the whole topological informa-

tion about n-dimensional multi-level subdivided objects. It is important since the reduction between levels (achieved by the applications of removal and contraction operation) leads to particular cases (for instance multi-adjacency of cells) which are usually not well handled by graphs.

We intend now to define and generalize the notion of receptive field for n-G-map pyramids. This classical notion, defined for graph pyramids and $2D$ combinatorial map pyramids, establishes a relation between a cell at a given level and the set of cells of lower levels which are contracted or removed into this cell.

It is also necessary to conceive operations for handling this structure: for instance, for modifying any level of the pyramid by contracting or removing a cell, by adding a new cell, etc. An important problem is the fact that it is necessary to efficiently propagate these modifications for the other levels.

Acknowledgements

The authors wish to thank Luc Brun and Walter Kropatsch for their encouragements and help, and David Fradin and Daniel Meneveaux for the many useful discussions.

References

1. Montanvert, A., Meer, P., Rosenfeld, A.: Hierarchical image analysis using irregular tessellations. PAMI **13** (1991) 307–316.
2. Jolion, J., Montanvert, A.: The adaptive pyramid : a framework for 2d image analysis. Computer Vision, Graphics and Image Processing **55** (1992) 339–348.
3. Kropatsch, W.: Building irregular pyramids by dual-graph contraction. Vision, Image and Signal Processing **142** (1995) 366–374.
4. Kropatsch, W., Macho, H.: Finding the structure of connected components using dual irregular pyramids. In: Cinquime Colloque DGCI, Université d'Auvergne (1995) 147–158
5. Kropatsch, W.: Abstraction pyramids on discrete representations. In: Discrete Geometry for Computer Imagery. Number 2301 in LNCS, Bordeaux, France (2002) 1–21.
6. Bertolino, P., Ribas, S.: Image sequence segmentation by a single evolutionary graph pyramid. In: Graph-Based Representations in Pattern Recognition, Springer Verlag (1998) 93–100.
7. Dombre, J.: Système de représentation multi-échelle pour l'indexation et la restauration d'archives médiévales couleur. Phd thesis, Université Poitiers (2002)
8. Marfil, R., Rodriguez, J., Bandera, A., Sandoval, F.: Bounded irregular pyramid: a new structure for color image segmentation. PR **37** (2004) 623–626.
9. Brun, L., Kropatsch, W.: Introduction to combinatorial pyramids. In G. Bertrand, A. Imiya, R.K., ed.: Digital and Image Geometry. Volume 2243 of LNCS. Springer Verlag (2001) 108–127.
10. Brun, L., Kropatsch, W.: Combinatorial pyramids. In Suvisoft, ed.: IEEE International conference on Image Processing (ICIP). Volume II., Barcelona, Spain, IEEE (2003) 33–37.

11. Brun, L., Kropatsch, W.: Contraction kernels and combinatorial maps. PRL **24** (2003) 1051–1057.
12. Lienhardt, P.: Subdivisions of n-dimensional spaces and n-dimensional generalized maps. In: Proceedings of the fifth annual symposium on Computational geometry, Saarbruchen, Germany, ACM Press (1989) 228–236.
13. Lienhardt, P.: N-dimensional generalized combinatorial maps and cellular quasi-manifolds. In: International Journal of Computational Geometry and Applications, Hamburg, Germany (1994) 275–324.
14. Damiand, G., Lienhardt, P.: Removal and contraction for n-dimensional generalized maps. In: Discrete Geometry for Computer Imagery. Number 2886 in Lecture Notes in Computer Science, Naples, Italy (2003) 408–419.
15. Lienhardt, P.: Subdivisions of surfaces and generalized maps. In: Proceedings of Eurographics, Hamburg, Germany (1989) 439–452.
16. Fradin, D.: Modélisation et simulation d'éclairage à base topologique: application aux environnements architecturaux complexes. PhD thesis, Université Poitiers (2004)
17. Brun, L., Kropatsch, W.: Implicit encoding of combinatorial pyramids. In Drbohlav, O., ed.: Proceedings of the Computer Vision Winter Workshop, Valtice, Czech Reublic (2003) 49–54
18. Damiand, G.: Définition et étude d'un modèle topologique minimal de représentation d'images 2d et 3d. Phd thesis, Université Montpellier II (2001)
19. Bertrand, Y., Damiand, G., Fiorio, C.: Topological map: minimal encoding of 3d segmented images. In: Workshop on Graph-Based Representations in Pattern Recognition, Ischia, Italy, IAPR-TC15 (2001) 64–73.

Towards Unitary Representations for Graph Matching

David Emms[†], Simone Severini[†], Richard C. Wilson[‡], and Edwin R. Hancock[‡]

[†] Departments of Computer Science and Mathematics,
[‡] Department of Computer Science,
University of York, York Y010 5DD, UK

Abstract. In this paper we explore how a spectral technique suggested by quantum walks can be used to distinguish non-isomorphic cospectral graphs. Reviewing ideas from the field of quantum computing we recall the definition of the unitary matrices inducing quantum walks. We show how the spectra of these matrices are related to the spectra of the transition matrices of classical walks. Despite this relationship the behaviour of quantum walks is vastly different from classical walks. We show how this leads us to define a new matrix whose spectrum can be used to distinguish between graphs that are otherwise indistinguishable by standard spectral methods.

1 Introduction

Random walks are useful tools in the analysis of the structure of graphs. The steady state random walk on a graph is given by the leading eigenvector of the transition probability matrix, and this in turn is related to the eigenstructure of the graph Laplacian. Hence, the study of random walks has been the focus of sustained research activity in spectral graph theory. For instance, Lovász has written a useful review of the subject [1], and spectral bounds have been placed on the properties of random walks including the mixing times and hitting times [2].

From a practical perspective, there have been a number of useful applications of random walks. One of the most important of these is the analysis of routing problems in network and circuit theory. Of more recent interest is the use of ideas from random walks to define the page-rank index for internet search engines such as Googlebot [3]. In the pattern recognition community there have been several attempts to use random walks for graph matching. These include the work of Robles-Kelly and Hancock [4, 5] which has used both the standard spectral method [4] and a more sophisticated one based on ideas from graph seriation [5] to convert graphs to strings, so that string matching methods may be used. Gori, Maggini and Sarti [6] on the other hand, have used ideas borrowed from page-rank to associated a spectral index with graph nodes and have then used standard subgraph isomorphism methods for matching the resulting attributed graphs.

One of the problems that limits the use of random walks, and indeed any spectral method, is that of co-spectrality. This is the situation in which struc-

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 153–161, 2005.

turally distinct graphs present the same pattern of eigenvalues. Classic examples are strongly regular graphs [7] and certain trees [8, 9].

Recently, quantum walks have been introduced as quantum counterparts of random walks [10, 11]. Their behaviour is governed by unitary rather than stochastic matrices. Quantum walks posses a number of interesting properties not exhibited by classical random walks. For instance, because the evolution of the quantum walk is unitary and therefore reversible, the walks are non-ergodic and what is more they do not have a limiting distribution. The present applications of quantum walks are fast quantum algorithms for database searching [12], graph traversal [13, 14], and the problem of element distinctness [15].

Although the analysis of quantum walks may seem detached from the practical problems listed above, they may offer a way of lifting the problems of co-spectrality. In this paper, we aim to explore how unitary matrices can be used to characterise graphs, and to study the walks they give rise to for strongly regular graphs. To convert the adjacency matrix of the graph into a unitary form we borrow ideas from quantum walks and combine the state space of the walk with a 'coin space' which dictates the quantum amplitudes of the various paths. Our main conclusion is that by making use of a unitary representation of the adjacency structure problems of co-spectrality can be lifted, and random walks can be used to distinguish otherwise indistinguishable graph structures.

2 Random Walks

2.1 The Classical Walk

Consider the graph $G = (V, E)$ with vertex set V and set of undirected edges $E = \{\{u, v\} : u, v \in V, u \neq v\}$. The adjacency matrix, A, for the graph is given by

$$A_{ij} = \begin{cases} 1 & \text{if } \{i, j\} \in E; \\ 0 & \text{otherwise.} \end{cases}$$

The elements of the transition matrix, T, are then

$$T_{ij} = A_{ij}/d_i$$

where $d_i = \sum_{j \in V} A_{ij}$ is the degree of the i^{th} vertex.

The classical (discrete) random walk on a graph is described by a Markov chain, $\{X_t\}$, with state space V. Transitions take place at discrete time intervals between adjacent vertices in the graph with probabilities $P(X_t = j \mid x_{t-1} = i) = T_{ij}$.

A fundamental result for the random walk, provided it is irreducible and aperiodic, is that there exists a unique probability distribution, π_i, satisfying the equations

$$\pi_i = \sum_{j \in V} T_{ij} \pi_j \quad \forall j.$$

This distribution plays an important role in characterizing the walk. For any starting distribution we have

$$P(X_t = v) \to \pi_v \quad \text{as } t \to \infty \quad \forall v.$$

Thus all walks approach a unique limiting distribution that is independent of the initial state.

2.2 The Quantum Walk

Quantum walks on graphs are quantum analogues of classical walks on graphs. There are two models, the discrete [16] and the continuous [13] quantum walk, which require the use of different methods to carry out the quantization. Both models were developed as a result of the interest in the possibilities of quantum computing and both exhibit many interesting properties that distinguish them from their classical counterparts. Although the behaviour of discrete and continuous versions are similar, the exact relationship is not yet fully understood. In this paper we will concentrate on the discrete quantum walk. Of particular interest to us is the possibility of using the quantum walk model to probe graphs using classical computation.

Let $\{|j\rangle : j \in V\}$ be the basis states of a $|V|$-dimensional Hilbert space, $\mathbf{H}$. A vector in this space is of the form

$$|\psi\rangle = \sum_{j \in V} a_j |j\rangle \quad a_j \in \mathbf{C}.$$

The inner product between two vectors $|\psi\rangle$ and $|\psi'\rangle$ is given by

$$\langle \psi | \psi' \rangle = \sum_{j \in V} a_j (a'_j)^*.$$

The state space for the quantum walk is the space of all vectors in $\mathbf{H}$ normalized according to the above inner product. This 'superpostion' means that any normalized complex linear combination of allowed quantum mechanical states is itself an allowed state. This allows the quantum walk to trace *all* possible paths simultaneously and interfere constructively or destructively according to their complex amplitudes. However, when a measurement is made of a quantum mechanical state according to a given basis just one of the basis states is observed with probability

$$P\big(|\psi_t\rangle = |j\rangle\big) = |\langle j | \psi_t \rangle|^2 = a_j a_j^*$$

and the wavefunction is said to 'collapse' to this state. All other information about the complex amplitudes of the various basis states prior to the measurement is lost.

Since measuring the walk causes the wavefunction to collapse to just one of the basis states, it is necessary that the coin used to determine which edge to traverse (more correctly the amplitudes given to traversing each of the possible

edges) is also part of the quantum system, otherwise the result is simply the classical walk. For a k-regular graph this is straightforward, the state space is supplemented by a k-dimensional coin space and the edges labelled from 1 to k such that each of the k edges associated with a particular vertex carries a different label. The basis states of the walk are thus $\{|v\rangle \otimes |c\rangle \ : \ v \in V, c = 1, \ldots, k\}$. Each step of the walk corresponds to one application of the unitary operator, $U = T(I_V \otimes C)$, where C is the coin operator on the coin space, I_V the identity on the vertex space and T the transition operator on the composite space [10]. The transition operator takes the state $|u\rangle|c\rangle \rightarrow |v\rangle|c\rangle$ where the edge $\{u, v\}$ is labelled c. The coin operator is such that

$$C|c\rangle = \sum_{j=1}^{k} a_k|k\rangle \quad \forall c \in \{1, \ldots, k\}, \quad \text{where } \sum_j a_j a_j^* = 1,$$

so that if the walk enters a vertex along one of the edges incident on it, it will leave along all k edges incident upon it with amplitudes dependent on the form the coin takes. The coin operator must be permutation invariant if the walk is to be independent of the labelling of the edges but can be varied so as to affect the mixing properties of the walk. The 'Grover coin' is typically chosen as the coin operator as it is the unitary operator furthest from the identity that is permutation invariant, and hence should provided the fastest mixing times

$$G_{ij}^{(k)} = \begin{cases} \frac{2}{k} - 1 & \text{if } i = j; \\ \frac{2}{k} & \text{otherwise.} \end{cases}$$

When the graph is not regular then coin operators of different dimensions are needed for different vertices, consequently the states for the walk and the operator U can no longer be written in the simple product form given above. Nevertheless, we can still write a general expression for the $2|E| \times 2|E|$ matrix U, whether or not the graph is regular:

$$U_{ij,kl} = \begin{cases} \frac{2}{d_j} - \delta_{il} & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases} \quad \forall\, (i,j),\ (j,i) \text{ where } \{i,j\} \in E.$$

The difference between classical and quantum walks and how the properties of quantum walks can be utilized is a topic of much research [14] [12] [17]. A major difference between a classical and a quantum walk is that a quantum walk does not tend towards a limiting distribution: since the evolution is unitary the magnitude of the quantity $|\psi_t\rangle - |\psi_{t-1}\rangle$ always remains the same. For a number of graphs hitting times of quantum walks have been found to be exponentially faster [14] [13], but in general quantum walks mix quadratically faster [16] than their classical analogues.

3 Analysis of the Unitary Representation

In this paper we wish to concentrate on the how the idea of quantum walks can be used classically. The spectrum of the unitary matrix governing the evolution

of a quantum walk turns out to be related to the spectrum of the transition matrix for the analogous classical walk. The unitary matrix can be written as

$$U_{ij,kl} = A_{ik}A_{jl}\delta_{jk}\left(\frac{2}{d_k} - \delta il\right).$$

Let $\mathbf{v}$ be an eigenvector of U with eigenvalue λ, then

$$\begin{aligned}\lambda v_{ij} &= \sum_{kl} U_{ij,kl}v_{kl} \\ &= \sum_{kl} \frac{2}{d_k}A_{ik}A_{jl}\delta_{jk}v_{kl} - \sum_{kl} A_{ik}A_{jl}\delta_{jk}\delta_{il}v_{kl} \\ &= \frac{2A_{ij}}{d_j}\sum_{l} A_{jl}v_{jl} - A_{ij}v_{ji}.\end{aligned}$$

Let $\mathbf{e}$ be an eigenvector of T with eigenvalue μ, we show that

$$v_{ij} = \frac{A_{ij}}{d_j} - \lambda^* \frac{A_{ij}e_i}{d_i}$$

satisfies this equation and hence is an eigenvector of U. In fact

$$\begin{aligned}\lambda v_{ij} &= 2\frac{A_{ij}}{d_j}\sum_{l} A_{jl}v_{jl} - A_{ij}v_{jl} \\ &= 2\frac{A_{ij}}{d_j}\sum_{l} A_{jl}^2\frac{e_l}{d_l} - 2\lambda^*\frac{A_{ij}}{d_j}\sum_{l} A_{jl}^2\frac{e_j}{d_j} - A_{ij}A_{ji}\frac{e_i}{d_i} + \lambda^* A_{ij}A_{ji}\frac{e_j}{d_j} \\ &= 2\frac{A_{ij}}{d_j}\sum_{l} A_{jl}\frac{e_l}{d_l} - 2\lambda^*\frac{A_{ij}e_j}{d_j^2}\sum_{l} A_{jl} - A_{ij}\frac{e_i}{d_i} + \lambda^* A_{ij}\frac{e_j}{d_j} \\ &= 2\mu\frac{A_{ij}e_j}{d_j} - 2\lambda^*\frac{A_{ij}e_j}{d_j} - A_{ij}\frac{e_i}{d_i} + \lambda^* A_{ij}\frac{e_j}{d_j} \\ &= \frac{A_{ij}e_j}{d_j}(2\mu - \lambda^*) - A_{ij}\frac{e_i}{d_i} \\ &= \lambda(\frac{A_{ij}e_j}{d_j}(2\lambda^*\mu - \lambda^{*2}) - \lambda^* A_{ij}\frac{e_i}{d_i})\end{aligned}$$

which is true if $(2\lambda^*\mu - \lambda^{*2}) = 1$. It follows that $\lambda = \mu \pm i\sqrt{1-\mu^2}$. The remaining eigenvalues are ± 1 each with multiplicity $|E| - |V|$. So the spectrum of U is completely determined by the spectrum of the transition matrix T. However, the random walk induced by U still has advantages over its classical analogue. In the next section we consider how we can make use of the differences between the walks.

3.1 Interference and the Supporting Digraph

Unlike the classical walk, the quantum walk traverses all possible paths simultaneously with amplitudes corresponding to the probability of each path. These

walks are not independent but are able to constructively or destructively interfere; giving rise to probability distributions on the vertices dependent upon this effect. This appears to allow the walk to probe and distinguish graphs more effectively than happens classically. The state of the walk after t steps is given by U^t, the (i,j) entry giving the amplitude in the state $|i\rangle$ at time t for a walk starting in the state $|j\rangle$. Define the supporting digraph [18] for the unitary matrix V to be the digraph with adjacency matrix

$$\underline{U_{ij}} = \begin{cases} 1 & \text{if } V_{ij} \neq 0; \\ 0 & \text{otherwise.} \end{cases}$$

The digraph supporting the unitary U^t has a non-zero (i,j) entry if and only if there is a non-zero probability of the walk starting in the state j being observed in the state i after t steps, where each state corresponds to a graph vertex together with a coin state. For small values of t this pattern of non-zero entries is more complex than is the case for the classical walk. It is the use of this matrix that we suggest for the task of distinguishing graphs, to this effect we propose the following algorithm:

Let G and H be two graphs and let $sp(X)$ denote the spectrum of the matrix X.

1. Construct U_G and U_H.
2. Compute $sp(U_G)$ and $sp(U_H)$.
   (a) If $sp(U_G) \neq sp(U_H)$ then $G \neq H$.
   (b) If $sp(U_G) = sp(U_H)$ then go to the next step.

n. ($n \geq 3$) Compute $sp(\underline{U_G^{n-1}})$ and $sp(\underline{U_H^{n-1}})$.
   (a) If $sp(\underline{U_G^{n-1}}) \neq sp(\underline{U_H^{n-1}})$ then $G \neq H$.
   (b) If $sp(\underline{U_G^{n-1}}) = sp(\underline{U_H^{n-1}})$ then go to Step $n+1$.

The idea behind the algorithm is simple. Given any two matrices X and Y, if $sp(X) = sp(Y)$ then $sp(X^t) = sp(Y^t)$ for all t. This is not the case if we consider $\underline{A_G^t}$ and $\underline{U_G^t}$. However, it is usually the case that $(\underline{A_G^t}) = J_n$, the all-one matrix, even for small values of t. Consequently, $\{sp(A_G^t)\}_{t=2}^{\infty}$ is of little use for distinguishing graphs. On the other hand, $U_{G_{ij}} \neq 0$ does not imply that $U_{G_{ij}}^t \neq 0$, for all t. Thus $\underline{U_G^t}$ is not necessarily the all one matrix and $sp(\underline{U_G^t})$ may still contain information about G. In practice we have not needed to go beyond step 4 to distinguish non-isomorphic graphs.

4 Experiments

Traditional spectral methods for various graph matching tasks rely on the use of the spectrum of either the adjacency or Laplacian matrices, but these methods fail when faced with a pair of non-isomorphic cospectral graphs. Strongly

regular graphs (*SRG*) provide examples of such graphs. A graph is *k-regular* if every vertex has the same degree k. A SRG with parameters (n, k, λ, μ) is a k-regular graph on n vertices, where each pair of adjacent vertices and each pair of nonadjacent vertices have exactly λ and μ common neighbours, respectively [7]. The spectra of the adjacency and Laplacian matrices are completely determined by the parameter set and hence any two co-parametric SRG are cospectral with respect to these matrices. At step $n = 4$, that is considering $sp(\underline{U_G^3})$ and $sp(\underline{U_H^3})$, the algorithm in the previous section was able to distinguish all co-parametric non-isomorphic SRG tested, which we summarize Table 1. In addition, this method was also able to distinguish between pairs of co-immanantal trees as constructed in [9].

As an example, let us consider the two non-isomorphic SRG with parameters $(16, 6, 2, 2)$ in Figure 1. We have

$$sp(A_G) = sp(A_H) = \{[-2]^9, [2]^6, [6]\}.$$

and

$$sp(L_G) = sp(L_H) = \{[0]^1, [4]^6, [8]^9\}.$$

However,

$$sp(\underline{U_G^3}) = \{[-7-2i]^{15}, [-7+2i]^{15}, [-5]^9, [-1]^{18}, [1]^{27}, [3]^5, [5]^6, [45]^1\}$$

and

$$sp(\underline{U_H^3}) = \{[-7-2i]^{15}, [-7+2i]^{15}, [-5]^6, [-1]^{24}, [1]^{21}, [3]^2, [5]^9, [45]^1\}.$$

Hence the algorithm is able to distinguish between these graphs.

Within each set of co-parametric SRG in Table 1, we computed a vector, $\overrightarrow{e}$, of the ordered eigenvalues of $\underline{U_G^3}$, for every graph G in the set. We then calculated the matrix with entries $D_{GH} = |\overrightarrow{e}_G - \overrightarrow{e}_H|$, for all G and H in the set. We found that $D_{GH} = 0$ only when $G = H$, thus distinguishing all non-isomorphic graphs. As an example, the matrix for the set with parameters $(26, 10, 3, 4)$ is

Table 1. The SRG used to test the algorithm. These SRG were obtained from [19]

(n, k, λ, μ)	Number of co-parametric SRG
$(16, 6, 2, 2)$	2
$(25, 12, 5, 6)$	15
$(26, 10, 3, 4)$	10
$(28, 12, 6, 4)$	4
$(29, 14, 6, 7)$	41
$(35, 18, 9, 9)$	227
$(36, 14, 4, 6)$	227
$(40, 12, 2, 4)$	28
$(45, 12, 3, 3)$	78
$(64, 18, 2, 6)$	167

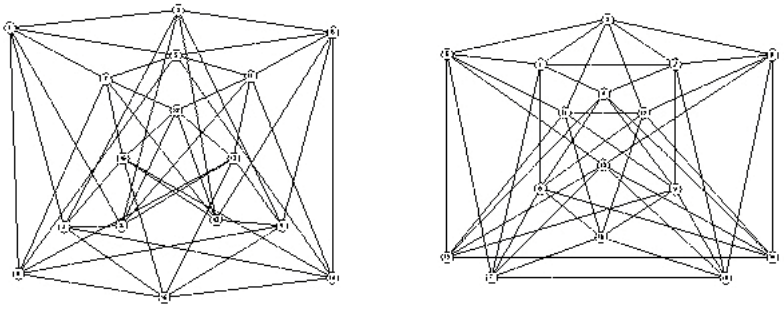


Fig. 1. Two non-isomorphic SGR (G left, H right) with the parameter set $(16, 6, 2, 2)$ (The graphs were drawn using Bill Kocay's "Graphs and Groups" program available at http://bkocay.cs.umanitoba.ca/G&G/G&G.html)

$$D = \begin{pmatrix}
0 & 4.1314 & 42.88 & 26.643 & 22.906 & 26.217 & 45.133 & 26.114 & 23.549 & 23.363 \\
4.1314 & 0 & 45.494 & 25.432 & 22.308 & 24.606 & 51.952 & 29.343 & 24.855 & 23.799 \\
42.88 & 45.494 & 0 & 53.421 & 55.587 & 58.849 & 15.507 & 96.276 & 53.686 & 57.496 \\
26.643 & 25.432 & 53.421 & 0 & 3.0823 & 3.8608 & 53.243 & 75.141 & 3.639 & 3.0694 \\
22.906 & 22.308 & 55.587 & 3.0823 & 0 & 2.4684 & 53.464 & 68.051 & 2.4905 & 1.178 \\
26.217 & 24.606 & 58.849 & 3.8608 & 2.4684 & 0 & 57.211 & 71.881 & 3.3889 & 2.5309 \\
45.133 & 51.952 & 15.507 & 53.243 & 53.464 & 57.211 & 0 & 94.333 & 51.902 & 55.511 \\
26.114 & 29.343 & 96.276 & 75.141 & 68.051 & 71.881 & 94.333 & 0 & 71.379 & 68.362 \\
23.549 & 24.855 & 53.686 & 3.639 & 2.4905 & 3.3889 & 51.902 & 71.379 & 0 & 1.8963 \\
23.363 & 23.799 & 57.496 & 3.0694 & 1.178 & 2.5309 & 55.511 & 68.362 & 1.8963 & 0
\end{pmatrix}.$$

5 Conclusions

We have reviewed how unitary matrices inducing discrete quantum walks (using the Grover coin) are constructed. We have shown how their spectra are related to the spectra of the transition matrix of the analogous classical random walk. We have looked at the supporting digraph of powers of such unitary matrices. These are related to the possible paths that a quantum walk can take. The spectra of the adjacency matrices of these digraphs are able to distinguish between pairs of non-isomorphic cospectral graphs, the classic examples of which are strongly regular graphs and certain trees.

References

1. Lovász, L.: Random Walks on Graphs: A Survey. In: Combinatorics, Paul Erdös is Eighty. Volume 2. János Bolyai Mathematical Society, Budapest (1996) 353–398
2. Sinclair, A.: Algorithms for random generation and counting: a Markov chain approach. Birkhauser Verlag, Boston (1993)
3. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems **30** (1998) 107–117

4. Robles-Kelly, A., Hancock, E.R.: String edit distance, random walks and graph matching. International Journal of Pattern Recognition and Artificial Intelligence **18** (2004) 315–327
5. Robles-Kelly, A., Hancock, E.R.: Edit distance from graph spectra. In: Proc. of the IEEE International Conference on Computer Vision. (2003) 234–241
6. Gori, M., Maggini, M., Sarti, L.: Graph matching using random walks. In: IEEE 17th International Conference on Pattern Recognition. (2004)
7. Cameron, P.J.: Strongly regular graphs. In: Topics in Algebraic Graph Theory. Cambridge University Press (2004) 203–221
8. Schwenk, A.J.: Almost all trees are cospectral. In Harary, F., ed.: New Directions in the Theory of Graphs, Academic Press (1973) 275–307
9. Merris, R.: Almost all trees are coimmanantal. Linear Algebra and its Applications **150** (1991) 61–66
10. Kempe, J.: Quantum random walks – an introductory overview. Contemporary Physics **44** (2003) 307–327
11. Ambainis, A.: Quantum walks and their algorithmic applications. International Journal of Quantum Information **1** (2003) 507–518
12. Shenvi, N., Kempe, J., Whaley, K.B.: A quantum random walk search algorithm. Phys. Rev. A **67** (2003)
13. Childs, A., Farhi, E., Gutmann, S.: An example of the difference between quantum and classical random walks. Quantum Information Processing **1** (2002) 35
14. Kempe, J.: Quantum random walks hit exponentially faster. In: Proc. of 7th Intern. Workshop on Randomization and Approximation Techniques in Comp. Sc. (RANDOM'03). (2003) 354–69
15. Ambainis, A.: Quantum walk algorithm for element distinctness. In: Proc. of 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04). (2004)
16. Aharonov, D., Ambainis, A., Kempe, J., Vazirani, U.: Quantum walks on graphs. In: Proc. 33th STOC, New York, NY, ACM (2001) 50–59
17. Rudolph, T.: Constructing physically intuitive graph invariants (2002)
18. Severini, S.: On the digraph of a unitary matrix. SIAM Journal on Matrix Analysis and Applications **25** (2003) 295–300
19. Spence, E.: Strongly Regular Graphs. www.maths.gla.ac.uk/es/srgraphs.htm (2004)

A Direct Algorithm to Find a Largest Common Connected Induced Subgraph of Two Graphs

Bertrand Cuissart and Jean-Jacques Hébrard

Groupe de Recherche en Électronique, Informatique et Imagerie de Caen,
CNRS-UMR6072, Université de Caen, France
{cuissart, hebrard}@info.unicaen.fr

Abstract. We present a direct algorithm that computes a largest common connected induced subgraph of two given graphs. It is based on an efficient generation of the common connected induced subgraphs of the input graphs. Experimental results are provided.

1 Introduction

Graphs are widely used to represent objects in various domains such as chemical information, computer imaging etc [1–p. 527]. Many applications in these domains necessitate the use of similarity measures which are often based on the calculation of a largest common subgraph of two graphs [2, 3, 4, 5, 6] .

There are numerous definitions of a largest common subgraph, depending on the notions of subgraph and size of a graph taken into account. We are interested here in the case where the considered subgraphs are the connected induced subgraphs, and where the size of a graph is the number of its vertices. This notion has been successfully applied in chemistry [7].

Recall that given two graphs G_1, G_2 and a positive integer k, the problem of deciding whether there exists a common subgraph of G_1 and G_2 of size greater than k is NP-complete [8]. In fact, the problem remains NP-complete if we restrict the search to common connected induced subgraphs. However, in practice, it is useful to define specialized algorithms for this particular case, since they could be significantly more efficient than a general one. A first algorithm that computes a largest common connected subgraph of two graphs was presented by I. Koch in [9]. This algorithm can be straightforwardly adapted to compute a largest common connected *induced* subgraph. It reduces the problem to the search of a largest clique of a compatibility graph associated with the input graphs. In this paper, we present a direct algorithm based on a procedure that efficiently generates the common connected induced subgraphs of the two input graphs. We experimentally compare the two algorithms.

2 Preliminaries

A *graph* G is denoted by $G(V, E)$ where V is the set of its vertices and E the set of its edges. The subgraph of G *induced* by a subset of its vertices, $V' \subseteq V$,

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 162–171, 2005.

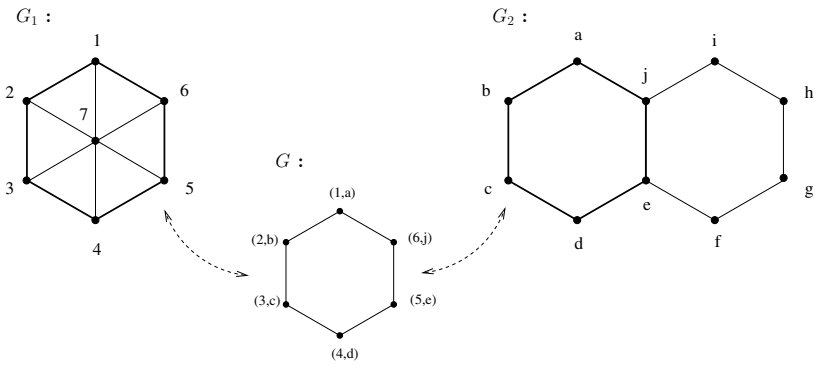


Fig. 1. A common subgraph of two graphs

is a graph consisting of V' and those edges of V with both vertices in V'. The subgraph of G induced by V' is denoted by $G[V']$. A graph G' is an induced subgraph of G if there exists $V' \subseteq V$ such that $G[V'] = G'$. *In all the paper, we say subgraph for induced subgraph.*

Two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are *isomorphic* if there exists a bijection $f : V_1 \longrightarrow V_2$ such that for every $u_1, v_1 \in V_1$, $\{u_1, v_1\} \in E_1$ if and only if $\{f(u_1), f(v_1)\} \in E_2$; f is called an *isomorphism*. A *common subgraph of* G_1 *and* G_2 is a set of ordered pairs $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ such that the function $f : \{u_1, \ldots, u_k\} \longrightarrow \{v_1, \ldots, v_k\}$ defined by $f(u_i) = v_i$ $(1 \leq i \leq k)$ is an isomorphism between $G_1[\{u_1, \ldots, u_k\}]$ and $G_2[\{v_1, \ldots, v_k\}]$.

Example 1. In Figure 1, $G = \{(1, a), (2, b), (3, c), (4, d), (5, e), (6, j)\}$ is a common subgraph of G_1 and G_2.

A graph G is *connected* if any two of its vertices are linked by a path in G. The problem *LCCIS (Largest Common Connected Induced Subgraph)* is to find a common connected induced subgraph C of two given graphs, such that the cardinality of C, $|C|$, is maximum.

Recall that a *tree* is a connected graph containing no circuit. The vertices of a tree will be called *nodes*. A *rooted tree* is a tree in which one node, the *root*, is distinguished. In a rooted tree, any node of degree one, unless it is the root, is called a *leaf*. If $\{u, v\}$ is an edge of a rooted tree such that u lies on the path from the root to v, then u is said to be the *parent* of v and v is a *child* of u. An *ancestor* of u is any node of the path from the root to u. If u is an ancestor of v, then v is a *descendant* of u, and we write $u \leq v$; if $u \neq v$, we write $u < v$. It is convenient to denote a tree T with root r by (T, r).

We present, in Section 5, an algorithm for LCCIS based on an efficient method for generating the common connected subgraphs of two graphs; this method is described in Section 4. But before generating the common connected subgraphs of two graphs, we must be able to simply generate the connected subgraphs of a graph; Section 3 is devoted to this question.

3 Connected Subgraphs Generation

We present a structure designed to generate the connected subgraphs of a graph.

Throughout this section, $G_1(V_1, E_1)$ is a fixed graph, and (T, r) is a rooted tree in which each node, except r, is labelled by an element of $\{v_+ : v \in V_1\} \cup \{v_- : v \in V_1\}$. The label of a node x, $x \neq r$, is denoted by $L(x)$. A label of the form v_+ (resp. v_-) is said to be *positive* (resp. *negative*). For each node x of T, we define $PL_+(x) = \{v \in V_1 : \exists y,\ y \neq r,\ y \leq x \text{ and } L(y) = v_+\}$; $PL_+(x)$ is the set of vertices of G_1 occurring in a positive label of an ancestor of x. Similarly, we define $PL_-(x) = \{v \in V_1 : \exists y,\ y \neq r,\ y \leq x \text{ and } L(y) = v_-\}$, and $PL(x) = PL_+(x) \cup PL_-(x)$; $PL(x)$ is the set of vertices of G_1 used for labelling the ancestors of x.

In order to generate the connected subgraphs of G_1, we build T in such a way that for each node x of T, $G_1[PL_+(x)]$ is connected. The set of vertices of G_1 that may label the children of x is denoted by $F(x)$ and defined as follows.

If $PL_+(x) \neq \emptyset$ then $F(x) = \{v \in V_1 : \exists u \in PL_+(x),\ \{v, u\} \in E_1\} \setminus PL(x)$, otherwise $F(x) = V_1 \setminus PL(x)$.

Example 2. In Figure 2, $PL_+(x) = \{a, b\}$, $PL(x) = \{a, b, c\}$, $F(x) = \{e\}$.

Remark that, by definition, $PL_+(r) = \emptyset$, $PL(r) = \emptyset$ and $F(r) = V_1$.

Definition 3. A tree of the connected subgraphs (TOCS) *of G_1 is a rooted tree (T, r) such that :*

1. *Every node of T, except r, is labelled by an element of $\{v_+ : v \in V_1\} \cup \{v_- : v \in V_1\}$.*
2. *For every node x of T, if $F(x) = \emptyset$, then x is a leaf, otherwise x has two children respectively labelled by v_+ and v_-, with $v \in F(x)$.*

See Figure 2 for an example of a TOCS.

We will now show that the leaves of a TOCS of G_1 and the connected subgraphs of G_1 are in a bijective correspondence. The following proposition is an immediate consequence of Definition 3.

Proposition 4. *Let f_1 and f_2 be two leaves of a TOCS. If $f_1 \neq f_2$ then $PL_+(f_1) \neq PL_+(f_2)$.*

Let $V' \subseteq V_1$ and (T, r) be a TOCS of G_1. We associate with V' the path of T, $(x_0, \ldots, x_k)$, defined as follows: $x_0 = r$, x_k is a leaf of T, and for all i $(0 \leq i < k)$, x_{i+1} is the child of x_i with a positive label v_+ if $v \in V'$, otherwise x_{i+1} is the child of x_i with a negative label. The leaf x_k is denoted by $l(V')$.

Example 5. In Figure 2, $l(\{a, b, e\}) = l(\{a, b, e, d\}) = f$.

Remark 6. Let $V' \subseteq V_1$. We have $PL_+(l(V')) \subseteq V'$, $PL_-(l(V')) \cap V' = \emptyset$ and $PL_+(l(V')) = \emptyset$ if and only if $V' = \emptyset$. Indeed, if $PL_+(l(V')) = \emptyset$, then $F(l(V')) = V_1 \setminus PL(l(V'))$, moreover, $F(l(V')) = \emptyset$ since $l(V')$ is a leaf, thus $PL(l(V')) = V_1$, $PL_-(l(V')) = V_1$, $V_1 \cap V' = \emptyset$ and consequently $V' = \emptyset$.

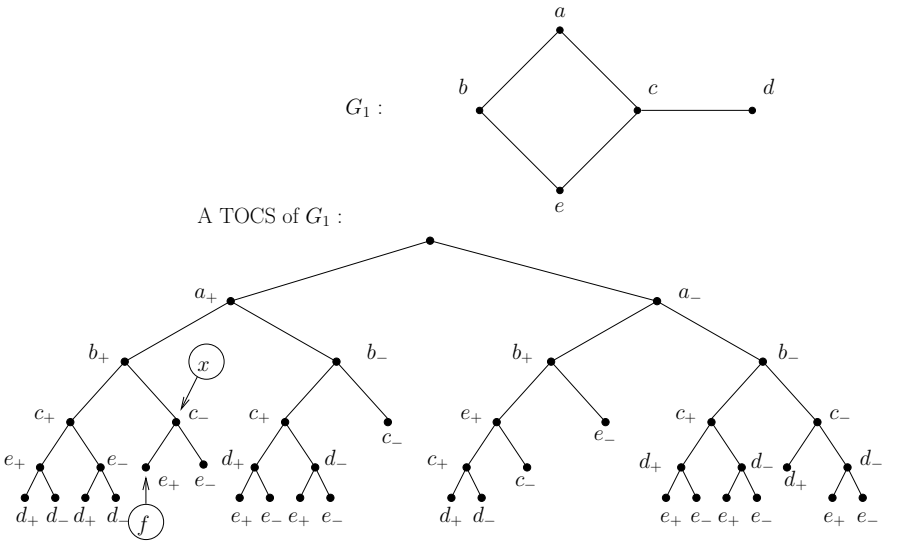


Fig. 2. A tree of the connected subgraphs of a graph

Proposition 7. *Let T be a TOCS of G_1.*

1. *(soundness) For every leaf f of T, $G_1[PL_+(f)]$ is connected.*
2. *(completeness) Let $V' \subseteq V_1$. If $G_1[V']$ is connected, then $V' = PL_+(l(V'))$.*

Proof. 1. Straightforward consequence of Definition 3.
2. 2. If $V' = \emptyset$ then $PL_+(l(V')) = \emptyset$ (Remark 6) and $V' = PL_+(l(V'))$.

If $V' \neq \emptyset$, then $PL_+(l(V')) \neq \emptyset$ (Remark 6). By definition of $l(V')$, we have $PL_+(l(V')) \subseteq V'$. Let $C = V' \setminus PL_+(l(V'))$. Suppose $C \neq \emptyset$. The sets C and $PL_+(l(V'))$ form a partition of V'. By hypothesis, $G[V']$ is connected, thus there exists an edge $\{u, v\} \in E_1$, with $u \in C$ and $v \in PL_+(l(V'))$. By definition of $l(V')$, $C \cap PL(l(V')) = \emptyset$, therefore $u \in F(l(V'))$. Now $l(V')$ is a leaf of T, thus $F(l(V')) = \emptyset$. A contradiction. Finally, $C = \emptyset$ and $V' = PL_+(l(V'))$. □

Example 8. In Figure 2, $G_1[\{a, b, e\}]$ is connected, $G_1[\{a, b, e, d\}]$ is not connected and $PL_+(f) = \{a, b, e\}$.

It follows from Prop. 4 and Prop. 7 that there exists a bijection from the leaves of a TOCS of G_1 to the connected subgraphs of G_1. As a consequence, the connected subgraphs of G_1 can be generated by simply performing a depth first traversal of a TOCS of G_1 [10].

4 Generation of the Common Connected Subgraphs of Two Graphs

We present an algorithm for generating the common connected subgraphs of two graphs.

Throughout this section, $G_1(V_1, E_1)$ and $G_2(v_2, E_2)$ are fixed graphs and T is a TOCS of G_1. For every leaf f of T, our algorithm build simultaneously all the subgraphs of G_2 isomorphic to $G_1[PL_+(f)]$.

We first define a rooted tree whose leaves are in a bijective correspondence with the common connected subgraphs of G_1 and G_2.

Definition 9. *Let (T, r) be a TOCS of G_1. The* tree of the common connected subgraphs of G_1 and G_2 (TOCCS) built from T *is a rooted tree (T', r') in which each node,except r', is labelled by an element of $V_1 \times V_2 \cup V_1 \times \{$"Excluded"$\}$. The label of a node x', $x' \neq r'$, is denoted by $L'(x')$. Each node x' of T' is associated with a node of T, called the* origin *of x' and denoted by $Or(x')$. (T', r') and the function Or are defined by induction as follows:*

1. *$Or(r') = r$.*
2. *Let x' be a node of T'.*
   (a) *If $Or(x')$ is a leaf of T, then x' is a leaf of T'.*
   (b) *Otherwise, $Or(x')$ has two children y and z respectively labelled by v_{1+} and v_{1-}, where $v_1 \in V_1$. Let $H(x') = \{v_2 \in V_2 :$ for any ancestor of x' labelled by (u_1, u_2) with $u_2 \in V_2, v_2 \neq u_2$ and $(\{v_1, u_1\} \in E_1 \iff \{v_2, u_2\} \in E_2)\}$, $k = |H(x')|$ and $w_1, \ldots, w_k$ denote the elements of $H(x')$. Then x' has $k+1$ children, $y'_1, \ldots, y'_{k+1}$, with $L'(y'_i) = (v_1, w_i)$ and $Or(y'_i) = y$ $(1 \leq i \leq k)$, $L'(y'_{k+1}) = (v_1,$ "Excluded") and $Or(y'_{k+1}) = z$.*

Example 10. Figure 3 displays a path from the root r' to a node x' in a TOCCS of the graphs G_1 and G_2. $Or(x') = x$, $F(x) = \{e\}$, x has two children y and z with $L(y) = e_+$ and $L(z) = e_-$. $H(x') = \{5\}$, x' has two children y' and z' with $L'(y') = (e, 5)$ and $L'(z') = (e,$"Excluded"$)$.

In a TOCCS, the labels of the form (v_1, v_2), with $v_2 \in V_2$, are called *positive* and the labels of the form $(v_1,$"Excluded"$)$ are called *negative*. For every node x' of a TOCCS, we define $PL'_+(x') = \{(v_1, v_2) \in V_1 \times V_2 :\ \exists y',\ y' \neq r',\ y' \leq x'$ and $L'(y') = (v_1, v_2)\}$.

The following proposition is an immediate consequence of Definition 9.

Proposition 11. *Let T' be a TOCCS of G_1 and G_2, and f'_1, f'_2 two leaves of T'. If $f'_1 \neq f'_2$ then $PL'_+(f'_1) \neq PL'_+(f'_2)$.*

We now establish the correspondence between the leaves of a TOCCS and the common connected subgraphs of G_1 and G_2.

Proposition 12. *Let T' be a TOCCS of G_1 and G_2.*

1. *(soundness) For every leaf f' of T', $PL'_+(f')$ is a common connected subgraph of G_1 and G_2.*
2. *(completeness) Let G be a common connected subgraph of G_1 and G_2. There exists a leaf f' of T', such that $PL'_+(f') = G$.*

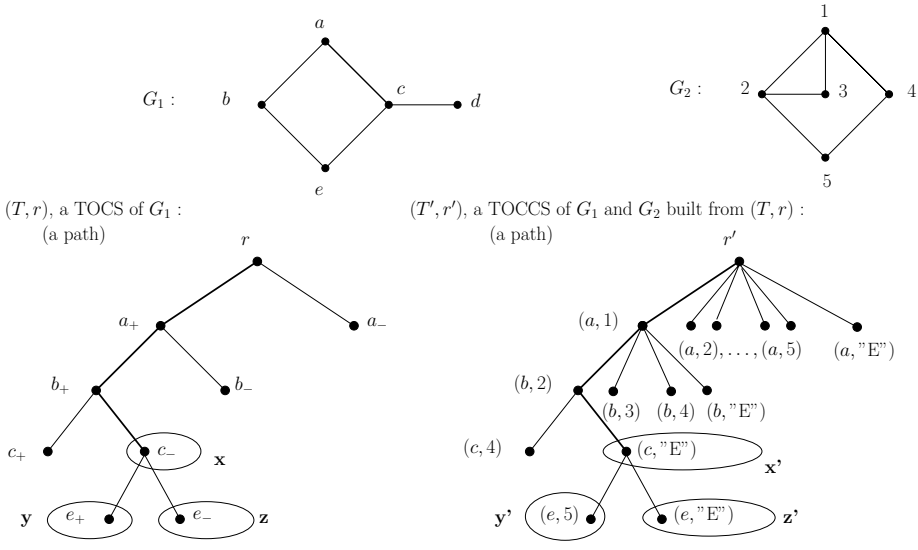


Fig. 3. A tree of the connected subgraphs common to two graphs

Proof (Sketch). 1. Straightforward consequence of Definition 9.
2. Let $G = \{(u_1, v_1), \ldots, (u_k, v_k)\}$ be a common connected subgraph of G_1 and G_2, $u_i \in V_1$ and $v_i \in V_2$ $(1 \leq i \leq k)$. Let (T, r) be the TOCS of G_1 from which (T', r') is built. By hypothesis, $G_1[\{u_1, \ldots, u_k\}]$ is connected. According to Prop. 7, we have $PL_+(l(\{u_1, \ldots, u_k\})) = \{u_1, \ldots, u_k\}$. Let $(x_0, \ldots, x_p)$ be the path from r to $l(\{u_1, \ldots, u_k\})$ in T.

For all i, $1 \leq i \leq p$, if $L(x_i)$ is a negative label, we write $\sigma(i) =$ "Excluded", otherwise, there exists a unique vertex $v \in V_2$ such that $(L(x_i), v) \in G$ and we write $\sigma(i) = v$.

Since G is a common connected subgraph of G_1 and G_2, there exists a path in T' $(x'_0, \ldots, x'_p)$ from r' $(r' = x'_0$) to a leaf f' $(f' = x'_p)$ such that $L'(x'_i) = (L(x_i), \sigma(i))$ $(1 \leq i \leq p)$. We have $PL'_+(f') = G$. □

It follows from Prop. 11 and Prop. 12 that there exists a bijection from the the leaves of a TOCCS of two graphs to their common connected subgraphs. Consequently, we can generate the common connected subgraphs of G_1 and G_2 by simply performing a depth first traversal of a TOCCS of G_1 and G_2. Notice that it is not necessary to compute and store the entire TOCCS. A depth first traversal only requires to store the path from the root to the node currently visited.

5 The Algorithm

A first algorithm for solving LCCIS consists in generating the common connected subgraphs of the two input graphs and returning a largest one. This algorithm

performs a complete traversal of a TOCCS of the input graphs. We will see that, in fact, parts of the TOCCS may be ignored.

Throughout this section, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are fixed graphs, (T, r) is a TOCS of G_1, and (T', r') is a TOCCS of G_1 and G_2 built from (T, r). Recall that for every node x of T, $PL_-(x) = \{v \in V_1 : \exists y,\ y \neq r,\ y \leq x$ and $L(y) = v_-\}$. For each node x' of T', we define $PL'_-(x') = \{(v_1, \text{"Excluded"}) : \exists y',\ y' \neq r', y' \leq x'$ and $L'(y') = (v_1, \text{"Excluded"})\}$. The following propositions allow us not to entirely examine the TOCCS explored by the algorithm.

Proposition 13. *Let $m \geq 0$ be an integer and x' a node of T'. If $|V_1| - |PL'_-(x')| \leq m$, then $|PL'_+(y')| \leq m$, for every descendant y' of x'.*

Remark that $|V_1| - |PL'_-(x')|$ is the number of vertices of V_1 not yet excluded, when x' is visited. Suppose the search has already discovered a common subgraph of size m. If $|V_1| - |PL'_-(x')| \leq m$, then it is useless to explore the principal subtree of T' at x', since, according to Prop. 13, we would not find any common subgraph of size greater than m.

Proposition 14. *Let x' be a node of T' such that $PL'_+(x') \neq \emptyset$, and f'_1 a leaf of T' descending from x' ($x' \leq f'_1$). If for every y' such that $x' < y' \leq f'_1$, $L'(y')$ is positive, then, for every leaf f'_2 descending from x', we have $|PL'_+(f'_2)| \leq |PL'_+(f_1)|$.*

Proof (Sketch). Let $x = Or(x')$, $f_1 = Or(f'_1)$ and $f_2 = Or(f'_2)$ (Definition 9). $G_1[PL_+(f_1)]$ is the largest connected subgraph of G_1 that contains the vertices of $PL_+(x)$ and does not contain the vertices of $PL_-(x)$. By hypothesis, $PL'_+(x') \neq \emptyset$, thus $PL_+(x) \neq \emptyset$. Consequently, $G_1[PL_+(f_2)]$ contains the vertices of $PL_+(x)$ and does not contain the vertices of $PL_-(x)$. Hence, $PL_+(f_2) \subseteq PL_+(f_1)$ and $|PL'_+(f'_2)| \leq |PL'_+(f'_1)|$.□

Let x' be a node of T' such that $PL'_+(x') \neq \emptyset$. After having explored a path p from x' to a leaf f'_1, such that p only contains nodes positively labelled (except x' possibly), it is useless to explore the other paths beginning at x'. Indeed we are sure, from Prop. 14, that these paths do not lead to any common subgraph larger than the one given by $PL'_+(f'_1)$.

Finally, our algorithm partially explores a TOCCS in a depth first manner, ignoring parts of it that cannot provide a common subgraph larger than the current largest one already found.

6 Experimental Results

It is essential to characterize graph matching algorithms by their performances [11, 12]. We experimentally compared our algorithm (T-TOCCS) with the algorithm based on the method proposed by I. Koch (C-CLIQUE) [9].

Material. The experiments were realized on a computer with 1 GB of memory and the AMD Athlon processor (2,2 GHz). We implemented both algorithms

Table 1. Experimental results

Randomly Connected Graphs, fixed density.									
S	M_C	E_C	M_T	E_T	S	M_C	E_C	M_T	E_T
	Density =0.1					Density =0.2			
21	5.235	3.37	0.462	0.345	16	1.259	0.133	0.172	0.0336
22	6.366	2.007	0.469	0.22	17	3.049	0.369	0.412	0.08
23	19.604	5.023	1.4	0.602	18	7.949	0.995	1.289	0.318
24	56.197	27.122	4.405	2.342	19	18.567	1.953	3.64	0.584
25	119	70.976	8.628	6.663	20	45.966	4.805	9.375	1.765
26	286.748	105.66	21.162	9.521	21	115.653	11.851	22.631	6.003
	Density =0.5					Density =0.85			
14	1.091	0.0332	0.272	0.048	10	0.7	0.07	0.017	0.0169
15	2.656	0.086	0.601	0.127	11	4.61	0.982	0.101	0.0712
16	6.049	0.228	1.397	0.052	12	14.658	1.974	0.17	0.16
17	13.821	0.451	3.848	0.394	13	62.386	14.122	0.913	0.826
18	31.078	0.95	9.718	1.6	14	242.79	36.954	3.858	2.966

Randomly Connected Graphs, fixed size.									
D	M_C	E_C	M_T	E_T	D	M_C	E_C	M_T	E_T
	Size = 18					Size = 22			
0.2	5.927	0.739	0.763	0.141	0.1	6.366	2.007	0.469	0.220
0.3	16.68	1.323	4.29	0.346	0.15	75.005	14.696	8.365	3.599
0.4	19.82	1.745	5.92	0.476	0.2	264.17	22.162	30.29	7.871
0.5	31.078	0.950	9.718	1.600					

Irregular 2d Meshes.									
D	M_C	E_C	M_T	E_T	D	M_C	E_C	M_T	E_T
Size = 9					Size = 16				
0.333	0	0	0	0	0.208	1.394	0.092	0.008	0.0064
0.417	0.009	0.0018	0	0	0.3	3.119	0.229	0.649	0.081
0.528	0.01	0	0	0	0.4	4.418	0.261	1.337	0.113
0.611	0.01	0	0	0	0.5	6.048	0.248	1.427	0.13
0.722	0.03	0	0	0	0.6	12.55	0.898	1.702	0.267
0.833	0.125	0.019	0	0	0.7	52.403	6.697	2.842	1.298
0.917	0.716	0.066	0.004	0.0064	0.75	127.571	12.325	4.995	1.836
1	13.173	0.0042	0	0					

S : Size of the input graphs, D : Density of the input graphs,

C-CLIQUE :
- M_C : Mean duration of the calculus (in seconds).
- E_C : Standard deviation.

T-TOCCS :
- M_T : Mean duration of the calculus (in seconds).
- E_T : Standard deviation.

in C, and compiled the code with the GNU gcc 3.3 compiler with appropriate optimizations.

The algorithms were tested on two kinds of graphs : *randomly connected graphs* and *irregular 2D meshes*. These graphs are described by M. De Santo et al. in [13, 14]. Notice that they are always connected. In a *randomly connected graph*, the probability of an edge connecting two vertices is independant on the vertices themselves. We adopted the model proposed in [15] to generate our instances. Since it is generally agreed that irregular 2D meshes represent a worst case for general graph matching algorithms [15], we also considered these graphs.

Results. We have compared the performances of the two algorithms on the same instances. The results are displayed on three tables (see Table 1). The first two tables concern the results obtained with randomly connected graphs, the third table is dedicated to irregular 2D meshes. A cell of these tables corresponds to a measure on a sample of ten instances. For example, the cell at the top left corner of the first table indicates that C-CLIQUE has taken on average 5.235 seconds to find a LCCIS between two randomly connected graphs of size 21 and of density 0.1. For each series, the measures were stopped as soon as the time required to solve one instance exceeded 500 seconds.

Discussion. The examination of the tables shows that T-TOCCS operates faster than C-CLIQUE on these instances.

When the input graphs are randomly connected graphs of density $\eta = 0.5$, T-TOCCS solves the problem 3 times faster than C-CLIQUE. This difference increases as the density of the input graphs moves away from 0.5. T-TOCCS operates on average 10 times faster than C-CLIQUE when the density is equal to 0.1. It operates 40 times faster when the input graphs have high density ($D = 0.85$).

The observations are similar for the 2D meshes. When the density of the meshes is close to 0.5, T-TOCCS operates about 3 times faster than C-CLIQUE. This ratio rapidly increases to reach 170 when the input graphs are the regular 2D meshes with 16 vertices. On the other side, this ratio reaches 25 when the input graphs have a density of 0.75.

References

1. Rosen, K., ed.: 8 : Graph theory, 9 : Trees. In: Handbook of discrete and combinatorial mathematics. CRC Press (2000) 495–628
2. Koch, I., Lengauer, T., Wanke, E.: An algorithm for finding common subtopologies in a set of protein structures. J. Comput. Biol. **3** (1996) 289–306
3. Koch, I., Lengauer, T.: Detection of distant structural similarities in a set of proteins using a fast graph based method. In Gaasterland, T., Karp, P., Karplus, K., Ouzounis, C., Sander, C., Valencia, A., eds.: Proc. 5th Int. Conf. on Intelligent systems for molecular biology, Menlo Park, AAAI Press (1997) 167–178
4. Raymond, J., Gardiner, E., Willett, P.: Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm. J. Chem. Inf. Comput. Sci. **42** (2002) 305–316

5. Raymond, J., Gardiner, E., Willett, P.: RASCAL : calculation of graph similarity using maximum common edge subgraphs. Comp. J. **45** (2002) 632–643
6. Garciá, G., Ruiz, I., Gómez-Nieto, M.: Step-by-step calculation of all maximum common substructures through a constraint satisfaction based algorithm. J. Chem. Inf. Comput. Sci. **44** (2004) 30–41
7. Cuissart, B., Touffet, F., Crémilleux, B., Bureau, R., Rault, S.: The maximum common substructure as a molecular depiction in a supervised classification context: experiments in quantitative structure/biodegradability relationships. J. Chem. Inf. Comput. Sci. **42** (2002) 1043–1052
8. Garey, M., Johnson, D.: Computers and Intractability: a guide to the theory of NP-completeness. Freeman (1979)
9. Koch, I.: Enumerating all connected maximal common subgraphs in two graphs. Theor. Comput. Sci. **250** (2001) 1–30
10. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: 22. Elementary graph algorithms. In: Introduction to algorithms. McGraw-Hill (2001)
11. Bunke, H., Foggia, P., Guidobaldi, C., Sansone, C., Vento, M.: A comparison of algorithms for maximum common subgraph on randomly connected graphs. In Caelli, T., Amin, A., Duin, R.P.W., Kamel, M.S., de Ridder, D., eds.: Joint IAPR International Workshops SSPR 2002 and SPR 2002, Proceedings. Volume 2396 of Lecture Notes in Computer Science., Springer-Verlag (2002) 123–132
12. Conte, D., Guidobaldi, C., Sansone, C.: A comparison of three maximum common subgraph algorithms on a large database of labeled graphs. In Hancock, E., Vento, M., eds.: IAPR Workshop GbRPR. Volume 2726 of Lecture Notes in Computer Science., Springer-Verlag (2003) 130–141
13. Foggia, P., Sansone, C., Vento, M.: A database of graphs for isomorphism and subgraph isomorphism benchmarking. In: Proc. 3rd IAPR TC-15 GbR Worshop. (2001) 176–187
14. DeSanto, M., Foggia, P., Sansone, C., Vento, M.: A large database of graphs and its use for benchmarking graph isomorphism algorithms. Pattern Recogn. Lett. **24** (2003) 1067–1079
15. Ullman, J.: An algorithm for subgraph isomorphism. Journal of the Association for Computing Machinery **23** (1976) 31–42

Reactive Tabu Search for Measuring Graph Similarity

Sébastien Sorlin and Christine Solnon

LIRIS, CNRS FRE2672, bât. Nautibus, University of Lyon I,
43 Bd du 11 novembre, 69622 Villeurbanne cedex, France
{sebastien.sorlin, christine.solnon}@liris.cnrs.fr

Abstract. Graph matching is often used for image recognition. Different kinds of graph matchings have been proposed such as (sub)graph isomorphism or error-tolerant graph matching, giving rise to different graph similarity measures. A first goal of this paper is to show that these different measures can be viewed as special cases of a generic similarity measure introduced in [8]. This generic similarity measure is based on a non-bijective graph matching (like [4] and [2]) so that it is well suited to image recognition. In particular, over/under-segmentation problems can be handled by linking one vertex to a set of vertices. In a second part, we address the problem of computing this measure and we describe two algorithms: a greedy algorithm, that quickly computes sub-optimal solutions, and a reactive Tabu search algorithm, that may improve these solutions. Some experimental results are given.

1 Introduction

Graphs are often used to model structured objects. In particular, graphs may be used for scene representation [2]: vertices represent scene regions, while edges represent binary relations between regions. In this context, image recognition and classification involves comparing graphs, *i.e.,* matching graphs to identify their common features [9]. This may be done by looking for an exact graph or subgraph isomorphism in order to show graph equivalence or inclusion. However, images are often corrupted by noise and distorsions and the assumption of the existence of an isomorphism is usually too strong. As a consequence, error-tolerant graph matchings such as maximum common subgraph and graph edit distance have been proposed [5, 9]. Such matchings drop the condition that the mapping must preserve all vertices and edges: the goal is to find a "best" mapping, *i.e.,* one which preserves a maximum number of vertices and edges.

Most recently, three different papers proposed to go one step further by introducing multivalent matchings, where a vertex in one graph may be matched with a set of vertices of the other graph:

– In [8], graphs are used to model design objects in a computer-aided design application. In this context, vertices are used to represent object components and one single component of an object may play the same role than a set

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 172–182, 2005.

of components of another object, depending of the granularity of object description. Therefore, the authors introduce a similarity measure based on multivalent mappings so that one vertex in a graph may be associated with a set of vertices of the other graph.
- In [4], graph matching is used for model-based pattern recognition of brain images. In this application, the assumption of a bijection between regions of the model and the image is too strong: model has a schematic aspect easy to segment while image is noised and usually over-segmented. Therefore, scene recognition is better expressed as a multivalent matching problem where a set of vertices of the scene may be linked to a same vertex of the model.
- In [2], a new graph edit distance is proposed, that introduces two new edit operations —vertex splitting and merging— in order to handle the fact that images may be over- or under- segmented.

Motivation and outline. A first goal of this paper is to "point out" the similarities between these three recent kinds of graph matchings. Another goal is to propose practical algorithms for multivalent graph matchings. Section 2 briefly introduces the graph similarity measure of [8]. Section 3 compares this measure with other graph matchings. Section 4 addresses the problem of computing this measure: we first propose a greedy algorithm that quickly computes an approximation of the similarity and then a reactive Tabu search approach, that may improve the computed approximation. Section 5 presents some experimental results.

2 A Generic Similarity Measure for Multi-labeled Graphs

A **directed graph** is defined by a couple $G = (V, E)$, where V is a finite set of vertices and $E \subseteq V \times V$ is a set of directed edges. Vertices and edges may be associated with labels that describe their properties. Given a set L_V of vertex labels and a set L_E of edge labels, a **multi-labeled graph** is defined by a triple $G = \langle V, r_V, r_E \rangle$ such that:

- V is a finite set of vertices,
- $r_V \subseteq V \times L_V$ is a relation associating labels to vertices, *i.e.,* r_V is the set of couples (v_i, l) such that vertex v_i is labeled by l,
- $r_E \subseteq V \times V \times L_E$ is a relation associating labels to edges, *i.e.,* r_E is the set of triples (v_i, v_j, l) such that edge (v_i, v_j) is labeled by l. Note that the set E of edges of the graph can be defined by $E = \{(v_i, v_j) | \exists l, (v_i, v_j, l) \in r_E\}$.

We shall call the tuples of r_V and r_E the vertex and edge features of G. The set $descr(G) = r_V \cup r_E$ of all vertex and edge features of a graph G completely describes the graph G.

We now briefly describe the graph similarity measure introduced in [8]; we refer the reader to [8] for more details. This similarity measure is defined for two multi-labeled graphs $G = \langle V, r_V, r_E \rangle$ and $G' = \langle V', r_{V'}, r_{E'} \rangle$, defined over the same sets of vertex and edge labels L_V and L_E, and such that $V \cap V' = \emptyset$.

The first step for measuring graph similarity is to map vertices. The mapping considered here is multivalent, *i.e.*, each vertex of one graph is mapped with a possibly empty set of vertices of the other graph. More formally, a **multivalent mapping** of the two graphs G and G' is a set $m \subseteq V \times V'$ which contains every couple $(v, v') \in V \times V'$ such that vertex v is mapped with vertex v'.

Once a multivalent mapping is defined, the next step is to identify the set of features that are common to the two graphs with respect to this mapping. This set contains all the features from both G and G' whose vertices (resp. edges) are matched by m to at least one vertex (resp. edge) that has the same feature. More formally, the set of common features $descr(G) \sqcap_m descr(G')$, with respect to a mapping m, is defined as follows:

$$\begin{aligned} descr(G) \sqcap_m descr(G') \doteq & \{(v,l) \in r_V | \ \exists (v,v') \in m,\ (v',l) \in r_{V'}\} \\ & \cup \{(v',l) \in r_{V'} | \ \exists (v,v') \in m,\ (v,l) \in r_V\} \\ & \cup \{(v_i,v_j,l) \in r_E | \ \exists (v_i,v_i') \in m, \exists (v_j,v_j') \in m\, (v_i',v_j',l) \in r_{E'}\} \\ & \cup \{(v_i',v_j',l) \in r_{E'} | \ \exists (v_i,v_i') \in m, \exists (v_j,v_j') \in m\, (v_i,v_j,l) \in r_E\} \end{aligned}$$

Given a multivalent mapping m, we also have to identify the set of split vertices, *i.e.*, the set of vertices that are mapped to more than one vertex, each split vertex v being associated with the set s_v of its mapped vertices:

$$\begin{aligned} splits(m) = & \{(v, s_v) \quad | \ v \in V, \quad s_v = \{v' \in V' | (v,v') \in m\}, |s_v| \geq 2\} \\ & \cup \{(v', s_{v'}) \mid v' \in V',\ s_{v'} = \{v \in V | (v,v') \in m\},\ |s_{v'}| \geq 2\} \end{aligned}$$

The **similarity** of G and G' with respect to a mapping m is then defined by:

$$sim_m(G,G') = \frac{f(descr(G) \sqcap_m descr(G')) - g(splits(m))}{f(descr(G) \cup descr(G'))} \tag{1}$$

where f and g are two functions that are introduced to weight features and splits, depending on the considered application. For example, if f is the cardinality function and g is the null function, then the similarity is proportional to the number of common features with respect to the total number of features. If g is the cardinality function, instead of the null function, then the similarity is decreased proportionally to the number of split vertices.

Finally, the **maximal similarity** $sim(G,G')$ of two graphs G and G' is the greatest similarity with respect to all possible mappings:

$$sim(G,G') = \max_{m \subseteq V \times V'} \frac{f(descr(G) \sqcap_m descr(G')) - g(splits(m))}{f(descr(G) \cup descr(G'))} \tag{2}$$

3 Generic Graph Similarity and Image Recognition

The measure of similarity described in section 2 has been first proposed for comparing objects in a computer-aided design application. However, this measure

is generic and it may be customized by properly defining functions f and g. In this section, we show how this measure relates to other graph matchings used in image recognition. These matchings are often defined for non labelled graphs. Hence we shall suppose that a non labelled graph is a particular labelled graph such that all vertices have a same label l_v and all edges have a same label l_e.

Graph isomorphism. The graph isomorphism problem between two graphs G=(V, E) and G'=(V', E') such that $|V|$=$|V'|$ consists in finding a bijective function $\phi : V \to V'$ such that $(v_1, v_2) \in E$ if and only if $(\phi(v_1), \phi(v_2)) \in E'$.

If functions f and g of formula (2) are defined as cardinality functions, then $sim(G, G')$=1 if and only if there exists a mapping m such that $descr(G) \sqcap_m descr(G')$=$descr(G) \cup descr(G')$ and $splits(m)$=$\emptyset$, *i.e.,* $sim(G, G')$=1 if and only if G and G' are isomorphic.

Partial subgraph isomorphism. The partial subgraph isomorphism problem between two graphs G=(V, E) and G'=(V', E') such that $|V| \leq |V'|$ consists in finding an injective function $\phi : V \to V'$ such that $(v_1, v_2) \in E \Rightarrow (\phi(v_1), \phi(v_2)) \in E'$.

Let us define function g of formula (2) as the cardinality function and function f as a weighted sum where the weight of the features of G (resp. G') is 1 (resp. 0). In this case, $sim(G, G')$=1 if and only if there exists a mapping m such that $descr(G) \subseteq descr(G) \sqcap_m descr(G')$ (as $f(descr(G) \cup descr(G'))$=$|descr(G)|$) and $splits(m)$=$\emptyset$, *i.e.*, $sim(G, G')$=1 if and only if there exists a partial subgraph isomorphism.

Subgraph isomorphism. The subgraph isomorphism problem is a special case of partial subgraph isomorphism: it adds the constraint that for each couple $(v_1, v_2) \in V^2$, if (v_1, v_2) is not an edge of G, then $(\phi(v_1), \phi(v_2))$ must neither be an edge of G'.

To check for subgraph isomorphism, we have to add "not-an-edge" labels to all couples of vertices that are not edges, and then to check that all "not-an-edge" labels of G are preserved by the mapping. More formally, given a graph G=(V, E), we define the labelled graph G_{label}=(V, r_V, r_E) such that r_V=$\{(v, l_v) | v \in V\}$ and r_E=$\{(u, v, l_e) | (u, v) \in E\} \cup \{(u, v, l_{notE}) | (u, v) \in V \times V - E\}$. Let us then define functions f and g of formula (2) as done for the partial subgraph. In this case, $sim(G_{label}, G'_{label})$=1 if and only if there exists a subgraph isomorphism.

Maximum common partial subgraph (mcps). The *mcps* of two graphs G and G' is the largest graph (with respect to the number of vertices and edges) which is a partial subgraph of both G and G'.

Let us define function f of formula (2) as the cardinality function, and function g so that split vertices are forbidden (*i.e.,* $g(S)$=$+\infty$ if $S \neq \emptyset$ and $g(\emptyset)$=0). In this case, the mapping m that maximizes formula (1) is the mapping which maximizes the number of common features in $descr(G) \sqcap_m descr(G')$, while forbidding split vertices, and therefore it corresponds to a *mcps*.

These definitions can be extended to the problem of finding the maximum common subgraph (*mcs*) of two graphs, *i.e.,* the problem of finding the largest non partial subgraph. This is done by considering labelled graphs that associate

"not-an-edge" labels to all couples of vertices which are not edges, like for the subgraph isomorphism problem. The *mcs* of two graphs is used in [7, 6, 5] to define the similarity of two graphs as $sim_{mcs}(G, G')=\frac{|mcs(G,G')|}{\max(|G|,|G'|)}$. One can appropriately define functions f and g in such a way that the mapping which maximizes formula (1) corresponds to the mapping which maximizes $sim_{mcs}(G, G')$.

Graph edit distance (ged). The *ged* of two graphs G and G' is the minimum cost set of weighted operations needed to transform G into G'. Considered operations are insertions, substitutions, and deletions of vertices and edges. [5] shows that, when considering appropriate weight definitions, *ged* is closely related to the maximum common subgraph, and therefore it is also closely related to the similarity measure of formula (2).

If we consider non labelled graphs (so that one only perform insertion and deletion operations) then, given a mapping m, each vertex or edge feature contained in $descr(G) - (descr(G) \sqcap_m descr(G'))$ (resp. in $descr(G') - (descr(G) \sqcap_m descr(G'))$) corresponds to a vertex or an edge deletion (resp. insertion). Let us then define function f as a weighted sum where weights are defined by operation costs. In this case, the mapping m which maximizes formula (1) gives the set of insertion and deletion operations which minimizes the *ged*.

If we consider labelled graphs (where each vertex and edge is associated with a single label), then substitution operations may be performed to change vertex or edge labels. If two vertices (or edges) are mapped by m but have a different label in G and G', then these labels will not belong to the set of common features $descr(G) \sqcap_m descr(G')$. Hence, one can also define a function f such that the mapping m which maximizes formula (1) gives the set of operations, including substitution operations, which minimizes the *ged*.

Extended ged. In order to compare over- and under-segmented images, [2] proposes to extend *ged* with two new operations: vertex splitting —to split one vertex of G into several vertices of G'— and vertex merging —to merge several vertices of G into one single vertex of G'.

Given a mapping m, the set of couples $(v, s'_v) \in splits(m)$ such that $v \in G$ corresponds to the set of splitting operations whereas the set of couples $(v', s_v) \in splits(m)$ such that $v' \in G'$ corresponds to the set of merging operations. Hence, if function g of formula (2) is defined as a weighted sum, where weights correspond to splitting and merging costs, and if f is defined as done for non extended *ged*, then the mapping which maximizes formula (1) corresponds to the extended *ged*.

Non bijective graph matching problem. This problem is introduced in [4] to find the best matching between models and over-segmented images of brains. Given a model graph $G=(V, E)$ and an image graph $G'=(V', E')$, a matching is defined as a function $\phi : V \rightarrow \wp(V')$ which associates to each vertex of the model graph G a non empty set of vertices of G', and such that (i) each vertex of the image graph G' is associated to exactly one vertex of the model graph G, (ii) some couples $(v, v') \in V \times V'$ are forbidden so that v' must not belong to $\phi(v)$, and (iii) the

subgraph induced by every set $\phi(v)$ must be connected. A weight $s^v(v_i, v'_i)$ (resp. $s^e(e_i, e'_i)$) is associated with each couple of vertices $(v_i, v'_i) \in V \times V'$ (resp. of edges $(e_i, e'_i) \in E \times E'$). The goal is to find the matching which maximizes a function depending on these weights of matched vertices and edges.

One can define functions f and g so that the mapping which maximizes formula (1) corresponds to the best matching as defined in [4]. To handle the fact that couples of vertices and edges are associated with weights, and also that condition (ii) is verified, we have to associate labels to vertices and edges in such a way that the label (v, v') (resp. (e, e')) belongs to $descr(G) \sqcap_m descr(G')$ if and only if v is mapped to v' (resp. e to e'). We can then define f as a weighted sum where a label (v, v') (resp. (e, e')) can be weighted with respect to $s^v(v, v')$ (resp. $s^e(e, e')$) or with negative infinite weight if mapping v to v' is forbidden. Function g is defined in such a way that mappings that do not verify conditions (i) or (iii) are forbidden, *i.e.,* g returns an infinite value when when image vertices are split or merged vertices are not connected.

Discussion

The graph similarity measures proposed in [2] and [4] are based on multivalent mappings (*i.e.*, one vertex may be mapped to several vertices). Both measures are used for image recognition. Indeed, images are often over- or under-segmented so that one has to associate one (under-)segmented region of an image to several (over-)segmented regions of another image.

However, the two similarity measures of [2] and [4] are specific to the addressed problem. In particular, [4] is used for matching brain images to models, and in this context they added specific constraints (*e.g.*, all model vertices must be mapped and each image vertex must be mapped to exactly one model vertex).

The similarity measure proposed in [8] is also based on multivalent mappings, but it is more generic, in the sense that specific constraints or preferences can be expressed thanks to functions f and g. The advantage of such a generic measure, where application-dependent constraints are specified via the two parameters f and g, is that algorithms for computing this measure can be used for different applications. As a counterpart, these algorithms may be less efficient than tailor-made programs, that have been designed for a particular application so that they can exploit specific knowledge to speed-up the solution process.

Moreover, the similarity measure proposed in [8] is defined for multi-labelled graphs, such that each vertex and edge can be associated with a set of labels that describe its properties. Such a multi-labelling could be very useful to describe images more accurately. For example, vertices could be labelled by colours, shapes, or sizes of corresponding regions, while edges could be labelled by distances, relative positions, or relative sizes of corresponding couples of regions.

4 Algorithms for Measuring Graph Similarity

All matching problems described in section 3 are NP-complete or NP-hard problems, except for graph isomorphism, the complexity of which is not exactly

stated, and for particular graphs (such as trees or planar graphs) for which some problems are polynomial ([1, 13, 15]).

Complete algorithms have been proposed for computing the mapping which maximizes formula (1) in [8] and for computing the cheapest set of edit operations in [2]. This kind of algorithms, based on an exhaustive exploration of the search space combined with pruning techniques, guarantees solution optimality. However, these algorithms are limited to very small graphs.

Therefore, incomplete algorithms, that do not guarantee optimality but have a polynomial time complexity, appear to be good alternatives. In particular, [4] proposes a randomized construction algorithm —that quickly computes a set of possible non-bijective graph matchings— and a local search algorithm that improves these matchings until a locally optimal point is reached. These algorithms are dedicated to the particular application of matching models with real images.

In this section, we describe three incomplete algorithms —a greedy one, a tabu search one and a reactive tabu search one— for measuring the similarity of two labelled graphs as defined by formula (2). These algorithms are generic in the sense that they are parameterized by the two functions f and g that contain domain-dependant knowledge.

Greedy algorithm. This algorithm has been first proposed in [8]. We briefly describe it because it is used as a starting point of tabu search algorithms. More information can be found in [8].

The algorithm starts from the empty mapping $m = \emptyset$, and iteratively adds to m couples of vertices chosen within the set of candidate couples $cand = V \times V' - m$. At each step, the couple to be added is chosen in a greedy way: we first select from *cand* the subset of couples that most increase the similarity as defined by formula (2). This subset often contains more than one candidate. To break ties between them, we look ahead the potentiality of each candidate (v, v') by taking into account the features that are shared by edges starting from (resp. ending to) both v and v' and that are not already in $descr(G) \sqcap_{m \cup \{(v,v')\}} descr(G')$. If there are still more than one couple which maximizes these looked-ahead common edge features, then one couple is randomly chosen. This greedy addition of couples to m is iterated until m is locally optimal, *i.e.*, until no more couple addition can increase the similarity.

This greedy algorithm has a polynomial time complexity of $\mathcal{O}((|V| \times |V'|)^2)$, provided that the computation of the f and g functions have linear time complexities with respect to the size of the mapping. As a counterpart of this rather low complexity, this algorithm never backtracks and is not complete. Hence, it may not find the best mapping; moreover, even if it actually finds the best mapping, it cannot be used to prove its optimality. Note however that, since this algorithm is not deterministic, we may run it several times and keep the best found mapping.

Local search. The greedy algorithm returns a "locally optimal" mapping in the sense that adding or removing one couple of vertices to this mapping cannot improve it. However, it may be possible to improve it by adding and/or removing

more than one couple to this mapping. A local search [12, 14] tries to improve a solution by locally exploring its neighborhood: the neighbours of a mapping m are the mappings which can be obtained by adding or removing one couple of vertices to m:

$$\forall m \in \wp(V \times V'), neighbourhood(m) = \{m \cup \{(v,v')\} | (v,v') \in (V \times V') - m\} \cup \{m - \{(v,v')\} | (v,v') \in m\}$$

From a good initial mapping, computed by the greedy algorithm, the search space is explored from neighbour to neighbour until the optimal solution is found (when the optimal value is known) or until a maximum number of moves have been performed. A heuristic selects the next neighbour to move on at each step.

fonction Tabu($G = \langle V, r_V, r_E \rangle, G' = \langle V', r_{V'}, r_{E'} \rangle, k, optBound, maxMoves$)
return a mapping $m \subseteq V \times V'$
  $m \leftarrow Greedy(G, G')$; $best_m \leftarrow m$; $nbMoves \leftarrow 0$
  while $sim_m(G, G') < optBound$ **and** $nbMoves < maxMoves$ **do**
    $cand \leftarrow \{m' \in neighbourhood(m) / sim_{m'}(G, G') > sim_{best_m}(G, G')\}$
    if $cand = \emptyset$ **then**/* *no aspiration* */
      $cand \leftarrow \{m' \in neighbourhood(m) / isNotTabu(m, m', k)$
    end if
    $cand \leftarrow \{m' \in cand / m'$ is maximal wrt formula (2) and look-ahead$\}$
    choose randomly $m' \in cand$
    $makeTabu(m, m', k)$; $m \leftarrow m'$; $nbMoves \leftarrow nbMoves + 1$
    if $sim_m(G, G') > sim_{best_m}(G, G')$ **then** $best_m \leftarrow m$ **end if**
  end while
  return $best_m$

Fig. 1. Tabu algorithm

Tabu meta-heuristic. Tabu search [12, 10, 16] is one of the best known heuristic to choose the next neighbour to move on. At each step, one chooses the best neighbour with respect to the same criteria than for the greedy algorithm. Note that this best neighbour may be worse than the current mapping if it is locally optimal. Hence, to avoid to stay around locally optimal mappings by always performing the same moves, a Tabu list is used. This list has a length k and memorizes the last k moves (*i.e.,* the last k added/removed couples) in order to forbid backward moves (*i.e.,* to remove/add a couple recently added/removed). An exception named "aspiration" is added: if a forbidden move reaches a better mapping than the best known mapping, the move is always done. Figure 1 describes the Tabu algorithm for computing formula (2).

Reactive tabu search. The length k of the tabu list is a critical parameter that is hard to set: if the list is too long, search diversification is too strong so that the algorithm converges too slowly; if the list is too short, intensification is too strong so that the algorithm may be stuck around local maxima and fail in improving the current solution. To solve this parameter tuning problem, [3] introduces *reactive Tabu search* where the length of the Tabu list is dynamically adapted

during the search. To make the Tabu algorithm reactive, one must evaluate the need of diversification of the search. When the same mapping is explored twice, the search must be diversified. In order to detect such redundancies, a hashing key is memorized for each explored mapping. When a collision occurs in the hash table, the list length is increased. On the contrary, when there is no collision during a fixed number of moves, thus indicating that search is diversified enough, one can reduce the list length. Hashing codes are incrementally computed so, this method has a negligible added cost.

5 Experimental Results

5.1 Experimental Settings

We now experimentally compare the three previously introduced algorithms: iterated greedy (which repeatedly computes 500 mappings with the greedy algorithm, and return the best one), Tabu search and reactive Tabu search. Non reactive version of Tabu search obtains its best average results when the length k of the Tabu list is between 10 and 20. Note that small variations on this length may have an important influence on the results and that the best setting for k is different from one instance to another. Best reactive Tabu search parameters are 10 (resp. 50) for the minimal (resp. maximal) length of the list, 15 for the size of extension and shortening of the list and 1000 moves for the frequency of reducing of the list. By opposition to (non reactive) Tabu search, these parameter settings are more "robust" in the sense that small variations on them do not significantly change performances.

5.2 Graph and Subgraph Isomorphism Problems

We first have done a set of experiments on graph and subgraph isomorphism problems coming from [11]. The iterated greedy algorithm solves 80% of hard graph isomorphism problem instances (regular graphs having 100 vertices, see [11]) in less than 10 seconds (on a Pentium IV 2Ghz, 512Mo of RAM) ; within the same time limit, reactive local search solves 100% of these problems. Moreover, reactive local search can solve 100% of any kind of graph isomorphism problems on graphs having up to 200 vertices in less than 20 seconds.

Subgraph isomorphim problems are much harder: within a limit of 200s, reactive Tabu search solves 66% of subgraph isomorphism problems on graphs with 100 vertices and iterated greedy algorithms only 4,4%. These rather poor results can be explained by the fact that our algorithms do not use any kind of filtering techniques and potentialy explore all kinds of mappings, even multivalent ones.

5.3 Multivalent Mapping Problems

In order to compare our three algorithms on multivalent mapping problems, we have used a random graph generator to generate "similar" pairs of graphs: it randomly generates a first graph and applies some vertex splitting/merging and some edge and vertex insertion/suppression to build a second graph which is

similar to the first one. From the set of transformations, a minimal bound of the similarity is computed. It is used to evaluate our algorithms, by counting the number of times they succeeded in reaching this bound (or an higher one).

When graph components have many different labels, the best mapping is trivially found as nearly all vertices/edges have different labels. Therefore, to obtain harder instances, we have generated 100 graphs such that all vertices and edges have the same label. These graphs have between 80 and 100 vertices and between 200 and 360 edges. The second graph is obtained by doing 5 vertex merging/splitting and 10 edge or vertex insertion/suppression. Functions f and g of formula (2) are defined as cardinality functions.

Comparative results. Each algorithm has been run 200 times on each of the 100 generated problems. 51 problems appeared to be "easy" ones as they were always solved by the iterated greedy algorithm. Over the 49 remaining "harder" problems, that could not be solved by the iterated greedy algorithm, 35 were easily —and always— solved, both by reactive and non reactive Tabu search (in less than 500 moves, corresponding to less than 4 seconds). The 14 last instances appeared to be really hard ones, that needed more than $25,000$ moves to be solved. For these 14 instances, (non reactive) Tabu search succeeded in finding a solution for 64% of the runs whereas reactive Tabu search succeeded for 79% of the runs. Furthermore, reactive Tabu search appeared to be more robust than its non reactive version in the sense that parameter settings influence less the results. So, the reactive Tabu search is more efficient and easier to tune than the non reactive version.

6 Conclusion

We have shown that the graph similarity measure of [8] is more generic than other graph similarity measures used in image recognition [6, 9]. It is based on multivalent graph matching so that over- and under- segmentation problems [2, 4] can be overrided by linking one vertex of a graph to a set of vertices of the other graph. We have given three algorithms with a polynomial complexity: a greedy algorithm, a local search based on Tabu meta-heuristic and an improved version of this search named "reactive Tabu search". These algorithms can compute a similarity measure based on multivalent mapping of two graphs having 100 vertices in a reasonable amount of time.

Further works. We have shown that the similarity measure proposed in [8] is generic in the sense that it can be used to formulate many graph similarity measure. Next step will be to test our algorithms on extended graph edit distance problems [2] and on non-bijective graph matching problems [4], in order to evaluate the efficiency of this formulation in a pratical point of view, and the usefulness of our algorithm in image recognition.

Our generic similarity measure associated with the expressive power of multi-labelled graphs could be used to define new powerful image similarity measures and, therefore, we plan to evaluate them in the field of image recognition and for content-based image querying system.

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison Wesley, 1974.
2. R. Ambauen, S. Fischer, and H. Bunke. Graph Edit Distance with Node Splitting and Merging, and Its Application to Diatom Identification. In *IAPR-TC15 Wksp on Graph-based Representation in Pattern Recognition*, pages 95–106, 2003.
3. R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. In Springer-Verlag, editor, *Algorithmica*, volume 29, pages 610–637, 2001.
4. M. Boeres, C. Ribeiro, and I. Bloch. A randomized heuristic for scene recognition by graph matching. In *WEA 2004*, pages 100–113, 2004.
5. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *PRL: Pattern Recognition Letters*, 18, 1997.
6. H. Bunke. Graph matching : Theoretical foundations, algorithms, and applications. In *Proc. Vision Interface 2000, Montreal*, pages 82–88, 2000.
7. H. Bunke and X. Jiang. *Graph Matching and Similarity*, volume Teodorescu, H.-N., Mlynek, D., Kandel, A., Zimmermann, H.-J. (eds.): Intelligent Systems and Interfaces, chapter 1. Kluwer Academic Publishers, 2000.
8. P.-A. Champin and C. Solnon. Measuring the similarity of labeled graphs. In *5th International Conference on Case-Based Reasoning (ICCBR 2003)*, volume Lecture Notes in Artificial Intelligence 2689-Springer-Verlag, pages 80–95, 2003.
9. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.
10. R. Dorne and J. Hao. *Tabu Search for graph coloring, T-coloring and Set T-colorings*, chapter 3. I.H. Osman et al. (Eds.), Kluver Academic Publishers, 1998.
11. P. Foggia, C. Sansone, and M. Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 176 –187, 2001.
12. F. Glover. Tabu search - part I. *Journal on Computing*, pages 190–260, 1989.
13. J.E. Hopcroft and J-K Wong. Linear time algorithm for isomorphism of planar graphs. 6^{th} *Annu. ACM Symp. theory of Comput.*, pages 172–184, 1974.
14. S. Kirkpatrick, S. Gelatt, and M. Vecchi. Optimisation by simulated annealing. In *Science*, volume 220, pages 671–680, 1983.
15. E.M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer System Science*, pages 42–65, 1982.
16. S. Petrovic, G. Kendall, and Y. Yang. A Tabu Search Approach for Graph-Structured Case Retrieval. In *STAIRS 2002*, pages 55–64, 2002.

Tree Matching Applied to Vascular System

Arnaud Charnoz[1,3], Vincent Agnus[1], Grégoire Malandain[2], Luc Soler[1], and Mohamed Tajine[3]

[1] Virtual Surg, IRCAD, Strasbourg, France
[2] Épidaure Research group, INRIA, Sophia Antipolis, France
[3] LSIIT, CNRS/ULP, Strasbourg, France

Abstract. In this paper, we propose an original tree matching algorithm for intra-patient hepatic vascular system registration. The vascular systems are segmented from CT-Scan images acquired at different time, and then modeled as trees. The goal of this algorithm is to find common bifurcations (nodes) and vessels (edges) in both trees.

Starting from the tree root, edges and nodes are iteratively matched. The algorithm works on a set of matching hypotheses which is updated to keep best matches. It is robust against topological modification, as the segmentation process can fail to detect some branches.

Finally, this algorithm is validated on the Visible Human with synthetic deformations thanks to the simulator prototype developed at the INRIA which provides realistic deformations for liver and its vascular network.

1 Introduction

1.1 Motivations

Matching and registration are fields in medical imaging with a great impact on visualization, diagnosis and surgery planning. In this paper, we focus on intra-patient follow-up of the hepatic vascular system between two acquisitions. We propose an automatic method which allows to match vessels and bifurcations. This approach is motivated by the fact that the liver is a very high deformable organ. The most reliable landmarks to estimate deformations sustained by the liver are provided by its vascular network.

The principal application of this work is to estimate the deformation of liver between two different times and to make a follow-up of tumors (see previous work [2]).

1.2 Previous Works

Related works propose algorithms to match and/or register vascular systems (brain, liver and, in a similar manner, lung airway). Generally, veins are modeled as graphs computed from segmented images and skeletons [8]. Some authors use some tree structure notions in their algorithms to register a tree with an image [1]

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 183–192, 2005.

or two trees [3]. Other approaches really match structures (nodes and vessels), but use general graph matching methods [9, 4, 5] or too specific methods like subtree isomorphism [7]. To summarize, the vascular tree matching problem is more specific than graph matching because the structure is simpler. On the other hand, it cannot be considered as a subtree isomorphism problem because of the segmentation problems. As a matter of fact, the segmentation process can miss some branches. This implies a (virtual) pruning on both trees, and thus an edge in a tree could be represented by several successive edges on the other tree.

In our previous work [2], vascular systems are modeled as a tree and then graph vertices have been matched together without taking into account possible segmentation errors. The previous algorithm works well on most branches but suffers from a lack of robustness in complex (but real) cases.

1.3 Proposal

The new algorithm proposed in this paper manages a matching hypotheses graph (MHG) where each matching hypothesis is associated with a cost. The MHG is updated as the matched branches set grow. This global approach allows us to find the best match (which minimizes a cost function) and not only a local solution.

The remainder of this paper is organized as follows. The first part presents the tree matching. We describe the generation of hypotheses and their associated cost functions. We explain how we update the MHG by keeping the best potential solutions.

The second part shows results and the algorithm's efficiency. We explain the validation protocol and we discuss tests for virtual and real patient. We finish with a discussion on future possible improvements.

2 Tree Matching

The proposed algorithm is a tree matching. Indeed, trees are a representation of skeletons computed from segmented vascular systems. The orientation symbolizes blood circulation flow. Nodes represent bifurcations and edges correspond to vessels between two bifurcations. Vessels has some geometric attributes: 3D positions, radius, vessel path.

Our goal is to find common bifurcations in both trees. Trees represent the same vascular system. However, their topology differ due to segmentation errors as well as 3D positions due to deformations applied on them. Furthermore, we assume that the tree roots are known (detection of vascular system entrance) and that the tree deformations are small (standard case).

In the next sections, we explain this tree matching. After introducing some notations, we see the framework used to generate all matching hypotheses. More specifically, we detail the two steps of the algorithm. Then, we focus on the selection of the best matching hypothesis.

2.1 Notations

We work on a tree noted $T = (V, E, r)$ where V represents the set of vertices, $E \subset V \times V$ the set of edges and r the root. For a node u in a tree T, $T(u)$ denotes the subtree of T induced from u. For a vertex v, sons(v) denotes the set of their child vertices, and father(v) its father vertex. For a vertex v, $out(v)$ denotes the set of out-edges of v, and $in(v)$ its in-edge. For an oriented edge $e = (v, u)$, we define $src(e) = v$ and $tgt(e) = u$. For two vertices $v, w \in V$, $P(v, w)$ is the unique path in T linking v to w. A path is a subtree of T. Let e an edge and its target vertex v, and let $DV_L(e) = \{u, \forall u \in \text{vertices of } T(v), \|P(v, u)\| \leq L\}$ denote the descendant vertex set composed of L-first depth level vertices in subtree induced from e. Let a vertex v, $T_+(v)$ denotes the subtree $T(v)$ where father(v) is added to vertex set and $in(v)$ to the edge set.

We introduce also some notations on functions. Let A and B two set with same size. Let $\mathcal{B}_{A,B}$ the bijection class from A to B. Let $\mathcal{C}^k_{A,B}$ the function class which defines a subset of k elements of A and subset of k elements of B. If $h \in \mathcal{C}^k_{A,B}$,then $h(A)$ and $h(B)$ denotes these subsets.

2.2 Framework of the Algorithm

Our algorithm searches for the best tree matching between path of $T_1 = (V_1, E_1, r_1)$ and $T_2 = (V_2, E_2, r_2)$ starting from roots (r_1 match with r_2). This algorithm process with a depth first search on T_1 and T_2. Two successive steps are repeated during the process and different hypotheses are studied (see figure 1): the first step determines the best out-edge matching set from a vertex. The second step determines the next best vertex matching in each out-edge subtree.

As the number of possible solutions is too large, some "bad" hypotheses are eliminated. The difficulty with this approach is the choice of *best match* at each step. This algorithm builds a research tree representing all possible matches where only the most probable configurations are studied.

2.3 Step I: Out-Edge Matching Set Hypothesis

Assumption: Let v_1 and w_1 (respectively v_2 and w_2) be two vertices of V_1 (respectively V_2). $\mathcal{P}(w_1, v_1, w_2, v_2)$ denotes a path matching between $P(w_1, v_1)$ and $P(w_2, v_2)$. Actually, we assume that v_1 and v_2 (respectively w_1 and w_2) represent the same bifurcation in a vascular system. At this step, $T_1(v_1)$ and $T_2(v_2)$ are not yet matched.

Generation: First, to continue the matching process between $T_1(v_1)$ and $T_2(v_2)$, the best out-edge matches between $out(v_1)$ and $out(v_2)$ have to be determined (figure 2). Let $O_1 = out(v_1)$ and $O_2 = out(v_2)$. An out-edge matching set hypothesis is noted $\mathcal{H}e(v_1, v_2)$. An hypothesis is represented by an out-edge matching set $\mathcal{E}_f(v_1, v_2)$ which characterizes a match between k elements of O_1 and O_2. $\mathcal{E}_f(v_1, v_2) = \{(e, f(e)), \forall e \in h(O_1)\}$ where $f \in \mathcal{B}_{h(O_1),h(O_2)}$ and $h \in \mathcal{C}^k_{O_1,O_2}$. Indirectly, this out-edge matching set assumes that some out-edges of O_1 (respectively O_2) noted $h(O_1)^c$ (respectively $h(O_2)^c$) have no association. Thus, some subtrees have no match in the other graph.

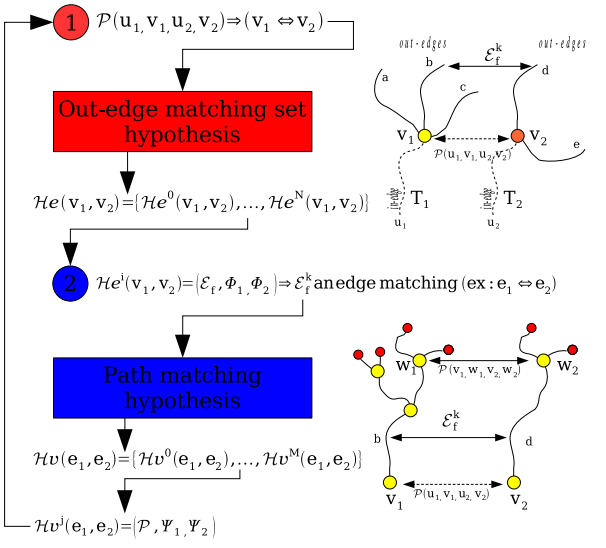


Fig. 1. This figure shows the successive steps of tree matching process and hypotheses generation

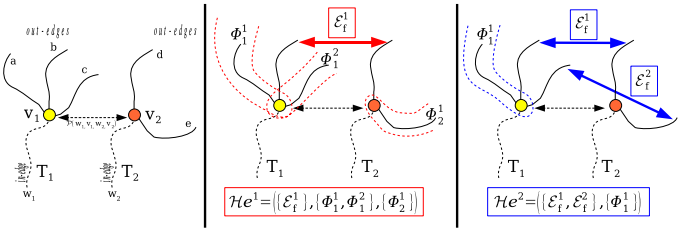


Fig. 2. The figure shows the creation of out-edge matching set hypotheses from a vertex matching. The left illustration resumes previous hypothesis. Other show 2 possible solutions where an out-edge matching set is chosen for each solution. Hypotheses suppose that few out-edges have no their equivalent in other tree and thus that the subtree correspondent is not match

Let $\phi(v, E) = \{T_+(u), \forall (v, u) \in E\}$ the subtree induced by a vertex and a subset of its out-edges. $\phi(v_1, h(O_1)^c)$ represents subtrees starting from v_1 that have no match.

If we assume that $|O_1| \leq |O_2|$, the possible hypotheses are given by:

$$\begin{gathered} \mathcal{H}e(v_1, v_2) = \{(\mathcal{E}_f(v_1, v_2), \phi(v_1, h(O_1)^c), \phi(v_2, h(O_2)^c)\}, \\ \forall k \in [0, |O_1|], \forall h \in \mathcal{C}_{O_1,O_2}^k, \forall f \in \mathcal{B}_{h(O_1),h(O_2)} \end{gathered} \tag{1}$$

Combinatory: When this association rule is respected, all out-edge matching sets can be created. Let $k \in [0, |O_1|]$, the number of possible function h which

choices two subsets with k elements in O_1 and O_2 is $|\mathcal{C}^k_{O_1,O_2}| = C^k_{|O_1|} \times C^k_{|O_2|}$. Moreover, the number of possible bijections between two subsets with k elements is $|\mathcal{B}_{h(O_1),h(O_2)}| = k!$.Thus, the number of out-edge matching set hypotheses is $|\mathcal{H}e(v_1, v_2)| = \sum_{k=0}^{N_{min}} k! C^k_{|O_1|} C^k_{|O_2|}$ where $N_{min} = min(|O_1|, |O_2|)$.

2.4 Step II: Path Matching Hypothesis

Supposition: An out-edge matching, noted $\mathcal{E}^i_f(v_1, v_2) = (e_1, e_2)$, assumes that an edge $e_1 \in O_1$ and an edge $e_2 \in O_2$ match (represent the same start vessel). This step purpose consist in finding the next common bifurcation in subtrees $T_1(tgt(e_1))$ and $T_2(tgt(e_2))$ closest to v_1 and v_2 then we restart at step I. Due to segmentation defects, $tgt(e_1)$ and $tgt(e_2)$ not necessarily represent the same bifurcation. For this fact, we search a vertex matching in subtrees and not only between $tgt(e_1)$ and $tgt(e_2)$ (Fig. 3).

Generation: The research of next vertex matching is restricted on the L first level of subtrees $T_1(tgt(e_1))$ and $T_2(tgt(e_2))$. Thus, we search the best vertex matching between $DV_L(e_1)$ and $DV_L(e_2)$.

Now, Let (w_1, w_2) a vertex matching with $w_1 \in DV_L(e_1)$ and $w_2 \in DV_L(e_2)$. w_1 are not necessary equal to $tgt(e_1)$ and this vertex matching imply a path matching $\mathcal{P}(v_1, w_1, v_2, w_2) = (P(v_1, w_1), P(v_2, w_2))$. This match also imply that some subtrees starting from $P(v_1, w_1)$ are not matched. We note this forest of no matching subtrees as $\psi(v, w) = \{T_+(u), \forall u \in \text{sons}(k), \forall k \in V_P, T_+(u) \cap P(v, w) = \{k\}\}$ where V_P = vertices of $(P(v, w))/\{v, w\}$. The set of possible path matching is defined as:

$$\begin{gathered} \mathcal{H}v(e_1, e_2) = (\mathcal{P}(v_1, w_1, v_2, w_2), \psi(v_1, w_1), \psi(v_2, w_2)), \\ \forall w_1 \in DV_L(e_1), \forall w_2 \in DV_L(e_2) \\ \text{with } v_1 = src(e_1) \text{ and } v_2 = src(e_2) \end{gathered} \tag{2}$$

Combinatory: Many path matches can be created. Thus, if we assume that $T_1(tgt(e_1))$ and $T_2(tgt(e_2))$ are complete on the L-first level and if in each bifurcation there are two out-edges, the number of path matching hypotheses is $|\mathcal{H}v(e_1, e_2)| = \sum_{k=0}^{L} 2^k \times \sum_{k=0}^{L} 2^k = (2^{L+1} - 1)^2$.

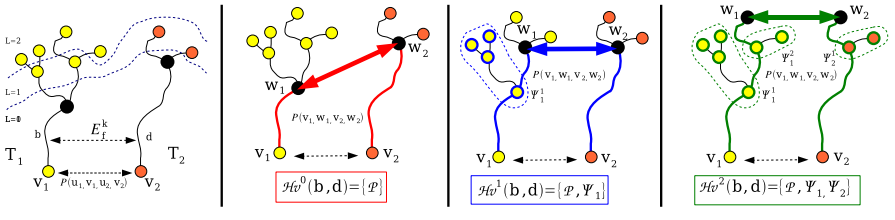


Fig. 3. Figure shows the creation of path matching hypotheses from an out-edge matching. Three solutions are illustrated

2.5 Hypotheses Selection

In the previous sections, we have seen how to generate all matching hypotheses. However, all possible tree matchings can not be explored due to huge combinatory and only the best hypotheses must be kept. The matching criterion is computed on the current match and only best solutions are kept to explore subgraphs. In fact, we want to minimize a global cost function (sum of local criteria) and discard temporary solutions with high cost. Nevertheless, we cannot accurately compare the same matchings between hypotheses. We have introduced a weight for hypotheses which represents the tree area already processed. It is used to compute a relative cost and thus to compare hypotheses.

In this manner, the n best out-edge matching set hypotheses $\mathcal{H}e^i$ must be selected for the step I and the m best path matching hypotheses $\mathcal{H}v^i$ for step II.

The local cost functions are computed for each hypothesis, and are used to distinguish two hypotheses and keep the best choices.

$$\begin{aligned} cost(\mathcal{H}e^i(v_1,v_2)) &= \textstyle\sum_{i=1}^{N_1} cost(\mathcal{E}_f^i(v_1,v_2)) \; + \sum_{i=1}^{N_2} cost(\phi^i(v_1,h(O_1)^c)) \\ &\quad + \textstyle\sum_{i=1}^{N_3} cost(\phi^i(v_2,h(O_2)^c)) \\ cost(\mathcal{H}v^i(e_1,e_2)) &= cost(\mathcal{P}(v_1,w_1,v_2,w_2)) \; + \textstyle\sum_{i=1}^{N_1} cost(\psi^i(v_1,w_1)) \\ &\quad + \textstyle\sum_{i=1}^{N_2} cost(\psi^i(v_2,w_2)) \end{aligned} \tag{3}$$

In these equations, we can observe three types of cost: a matching cost between two out-edges, a cost between two paths and a cost for subtrees which are no matched. In next sections, we detail each costs.

Out Edge Matching Cost: To simplify notation, we note an out-edge matching cost $cost(\mathcal{E}_f^i(v_1,v_2)) = oemc(e_1,e_2)$. Remember that an edge e represents a vessel between two bifurcations. In the following expression costs, $e(t)$ is the 3D parametric curve representation of the vessel, $r(t)$ represents the vessel's radius along the curve and l is the curve's length. With *oemc*, we compare edge orientation and edge radius.

$$\begin{aligned} oemc(e_1,e_2) = &\int_0^{l_{min}} \|e_1(t) - e_2(t) + e_1(0) - e_2(0)\|^2 dt \; + \\ &\gamma \int_0^{l_{min}} \|1 - \frac{r_1(t)}{r_2(t)}\|^2 dt \end{aligned} \tag{4}$$

Path Matching Cost: $P(v_1,w_1)$ denotes a path composed of successive edges (vessels). However to simplify notations, we note $P(v_1,w_1) = e_1$ where e_1 represents a virtual edge. Notations become $cost(\mathcal{P}(v_1,v_2,w_1,w_2)) = pmc(e_1,e_2)$. In this cost, weights are added to favor path with same length and short paths, as there are many common nodes in both studied subtrees.

$$\begin{aligned} pmc(e_1,e_2) = (1 + \alpha\frac{l_{max}}{l_{min}} + \beta\frac{l_1+l_2}{2}) \times (\gamma &\int_0^1 \|1 - \frac{r_1(t\times l_1)}{r_2(t\times l_2)}\|^2 dt \; + \\ &\int_0^1 \|e_1(t\times l_1) - e_2(t\times l_2) + e_1(0) - e_2(0)\|^2 dt) \end{aligned} \tag{5}$$

No Matching Tree Cost: We have previously considered a cost for no inclusion subtree in the matching solution. We have noted these costs $cost(\phi^i(u, E))$ and $cost(\psi^j(u, v))$. These subtrees $T_+(w)$ are defined by a vertex w. To simplify notations, we replace the previous expression cost by $nmtc(w)$. This cost is very important and the choice for weighs is difficult. If this cost is too high then all nodes are matched and conversely, if it is too low, we have no selected match.

$$nmtc(v) = (1 + \delta \frac{|T(v)|}{|T|}) \times pmc(e, g(e)) + \sum_{k=1}^{|\text{sons}(v)|} nmtc(w_k)$$
$$\text{with: } g(e) = e \text{ but with } r(t) = R_{min} \text{ minimum radius to vessel segmentation} \quad (6)$$

3 Experiments and Validation

3.1 Validation Protocol on Virtual Patients

To test and validate our algorithm, we have worked on a liver and its hepatic vascular system. To work on a complex vascular system (280 nodes), the Visible Man (cf. The Visible Human Project of the NLM) has been segmented.

To simulate deformations, we have used the minimally invasive hepatic surgery simulator prototype (Fig. 4) developed at the INRIA [6]. The goal of this simulator is to provide a realistic training framework to learn laparoscopic gestures. For this paper, we used it only to simulate deformations of the liver and its vascular system. This simulator uses complex biomechanical models, based on linear elasticity and finite element theory which include anisotropic deformations.

To simulate segmentation errors on our phantom, we have pruned random tree branches. It's more probable to loose small vessels than to loose large vessels.

3.2 Results on a Virtual Patient

The results on a virtual patient are good (figure 6) and fast (about 4 minutes to register 380 nodes on 1GHz PC). We have realized 10 different deformations on

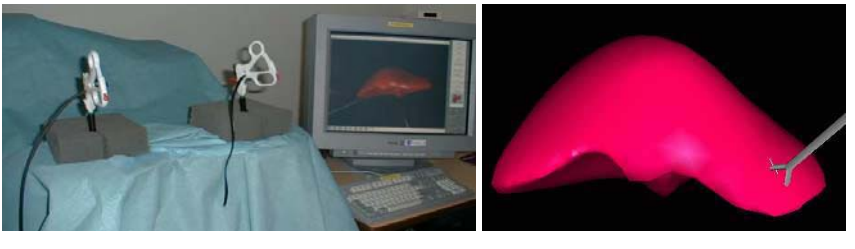


Fig. 4. [Left] Surgery simulator prototype developed by INRIA. [Right] Modeling a contact between a surgical tool and the liver soft tissue model

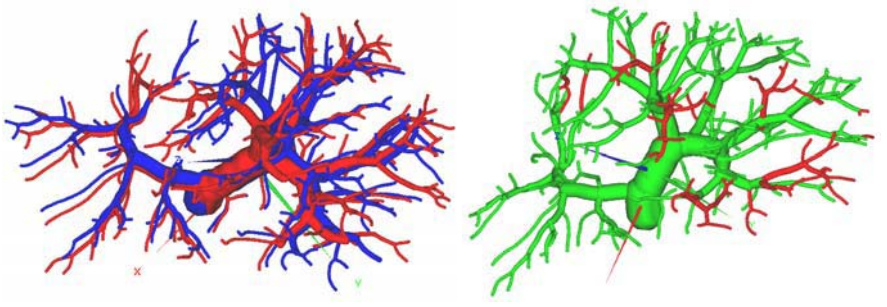


Fig. 5. [Left] Example of small deformations realized with the simulator. [Right] Example of a pruning representing 20% of the surface tree

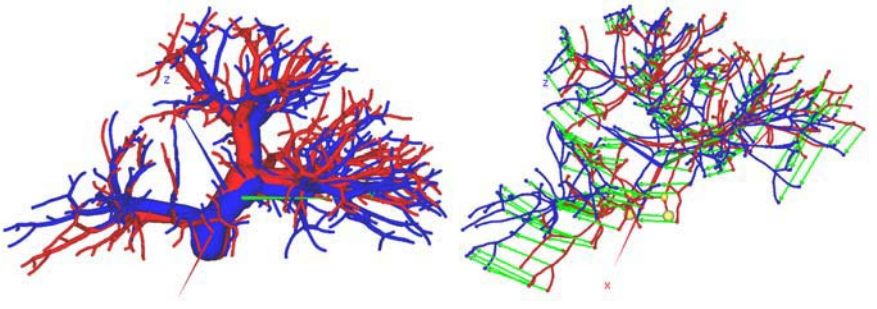


Fig. 6. [Left] Deformation and pruning on the Visible Man computed by the INRIA simulator. [Right] Figure shows result of our oriented tree matching, match are represented by arrows and represent 90% of all nodes

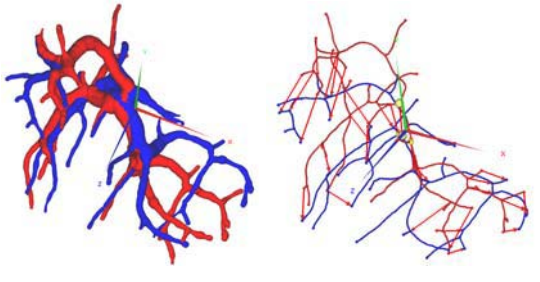


Fig. 7. [Left] Real patient where the vascular system has been segmented between two acquisitions. [Right] Figure shows result of our oriented tree matching, match are represented by arrows and represent 95% of all nodes

the Visible Man's liver. For each deformation, 50 random prunings are computed to lost approximately 20% of surface branches in both trees (figure 5).

We match 90% of all common nodes. The most part of matching errors (incorrect node correspondences and lost branches) is localized on terminal edges. On these nodes, the algorithm suffers from a lack of information (no subtree, dense node concentrations, small vessels). This make the matching task harder.

3.3 Results on a Real Patient

We have tested our algorithm on a real patient between two acquisitions during his therapy. The trees are simpler for our virtual patient and cost weights have been modified in order to get better matches. However, the result is good and promising for the next validation on a database of real patients: we have matched 95% of all common nodes (figure 7).

4 Conclusions and Future Work

We have presented an original new method to match vascular system between two acquisitions with a tree matching. This method is specific, fast and robust on a complex vascular system. The early stage of validation is very encouraging: most nodes are matched correctly. Thanks to the virtual database generated by the INRIA simulator we could test several configurations. Nevertheless, a lot of work needs to be done.

Presently, we concentrate our efforts on the design of cost function and its relative cost weights to get an algorithm more robust on large deformations. We will soon propose to apply the estimated deformations on a subtree of matched nodes to superimpose them.

In parallel, we will validate our works on a real patients database with the collaboration of Strasbourg hospital and also propose a new tool for automatic diagnosis of tumors evolution in the liver. The graph matching algorithm could ease the vessel segmentation process by detecting missed branches on the other graph.

Acknowledgments. We thank the Strasbourg hospital and their surgeons for providing images as well as their advices on "standard" deformations applied on the liver. This work have benefitted from the segmentation program of vascular system developed by the IRCAD R&D team. The realistic liver deformations are provided by the INRIA simulator from the Epidaure project. Many thanks to Clément Forest for his assistance during the use of the simulator.

References

1. S.R. Aylward, J. Jomier, S. Weeks, and E. Bullitt. Registration and analysis of vascular images. *IJCV*, 55(2-3):123–138, 2003.
2. A. Charnoz, V. Agnus, and L. Soler. Portal vein registration for the follow-up of hepatic tumours. In *MICCAI*, volume 3217 of *LNCS*, pages 878–886, Saint-Malo, France, September 2004. Springer Verlag.

3. T. Lange, S. Eulenstein, M. Hunerbein, H. Lamecker, and P.-M Schlag. Augmenting intraoperative 3d ultrasound with preoperative models for navigation in liver surgery. In *MICCAI*, volume 3217 of *LNCS*, pages 534–541, Saint-Malo, France, September 2004. Springer Verlag.
4. Y. Park. *Registration of linear structures in 3-D medical images.* PhD thesis, Osaka University, Japan. Departement of informatics and Mathematical Science, 2002.
5. M. Pelillo, K. Siddiqi, and S.W. Zucker. Matching hierarchical structures using association graphs. *PAMI*, 21:1105–1120, November 1999.
6. G. Picinbono, J-C. Lombardo, H. Delingette, and N. Ayache. Improving realism of a surgery simulator: linear anisotropic elasticity, complex interactions and force extrapolation. *JVCA*, 13(3):147–167, jully 2002.
7. C. Pisupati, L. Wolff, W. Mitzner, and E. Zerhouni. Tracking 3-d pulmonary tree structures. In *MMBIA*, page 160. IEEE Computer Society, 1996.
8. L. Soler, H. Delingette, G. Malandain, J. Montagnat, N. Ayache, J.-M. Clément, C. Koehl, O. Dourthe, D. Mutter, and J. Marescaux. A fully automatic anatomical, pathological and fonctionnal segmentation from ct-scans for hepatic surgery. In *Medical Imaging*, SPIE proceedings, pages 246–255, San Diego, February 2000.
9. J. Tschirren, K. Palágyi, J.M. Reinhardt, E.A. Hoffman, and M. Sonka. Segmentation, Skeletonization, and Branchpoint Matching - A Fully Automated Quantitative Evaluation of Human Intrathoracic Airway Trees. In *MICCAI*, volume 2489, pages 12–19. Springer-Verlag Heidelberg, 25 September 2002.

A Graph-Based, Multi-resolution Algorithm for Tracking Objects in Presence of Occlusions

Donatello Conte[1], Pasquale Foggia[2], Jean-Michel Jolion[3], and Mario Vento[1]

[1] Dipartimento di Ingegneria dell'Informazione ed Ingegneria Elettrica,
Università di Salerno,
Via Ponte don Melillo, I-84084 Fisciano (SA), Italy
{dconte, mvento}@unisa.it
[2] Dipartimento di Informatica e Sistemistica,
Università di Napoli, Via Claudio 21, I-80125, Napoli, Italy
foggiapa@unina.it
[3]Lyon Research Center for Images and Information Systems,
FRE CNRS 2672 Bat. 403 INSA Lyon, 69621 Villeurbanne Cedex, France
jolion@rfv.insa-lyon.fr

Abstract. In a video surveillance system the object tracking is one of the most challenging problem. In fact objects in the world exhibit complex interactions. When captured in a video sequence, some interactions manifest themselves as occlusions. A visual tracking system must be able to track objects which are partially or even fully occluded. In this paper we present a novel method of tracking objects through occlusions using a multi-resolution representation of the moving regions. The matching between objects in two consecutive frames to recognize the trajectories is preformed in a graph theoretic approach. The experimental results on the standard database PEST2001 show that the approach looks promising.

1 Introduction

Real-time object tracking is recently becoming more and more important in the field of video analysis and processing. Applications like traffic control, user–computer interaction, on-line video processing and production and video surveillance need reliable and economically affordable video tracking tools. In the last years this topic has received an increasing attention by researchers. However many of the key problems are still unsolved.

Occlusions represent one of the most difficult problems in motion analysis. Most of early works in this field either do not deal with occluding cases at all [14] or impose rigid constraints on camera positioning [2], in order to make the problem tractable. While it is possible to find applications where this is acceptable, in many real world cases the ability to deal with occlusions is essential to the practical usefulness of a tracking system. In fact, very often the camera positioning is dictated by architectural constraints (e. g., inside of a building), and furthermore it may not be desirable to have a point of view that is too high with respect to the observed scene since this

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 193 – 202, 2005.

may limit the ability to classify the tracked objects. Hence a tracking system should be prepared to face the possibility that the object being followed gets (at least partially) covered either by a static element of the scene (e.g. a tree or a wall) or by another moving object. This is particularly true when the objects being tracked are persons, since even in a not crowded scene two persons are likely to walk so close to each other (say because they are talking) to form an occlusion.

In this paper we present a novel algorithm for object tracking that is able to deal with partial occlusions of the objects. In particular, our algorithm is based on a pyramidal decomposition of the objects being tracked, using a multi-resolution approach. When an object is partially occluded by another, the algorithm descends to a more detailed level of representation in order to assign each part of the collapsed region to the proper object.

In the paper we focus our attention only to the tracking phase of the system. This phase is built upon a foreground detection technique described in another work [3], which identifies for each frame of the video which pixels (grouped into connected components) belong to the moving regions.

The rest of the paper is organized as follows. A review of related research is presented in Section 2. Section 3 describes the architecture of our tracking system. In Section 4 and Section 5 our algorithm is described with some examples. Section 6 presents the experimental results on a standard database of video sequences. Finally, some discussion and conclusions can be found in Section 7.

2 Related Works

In the last decade there has been several works on tracking moving objects through occlusions.

A first group of algorithms [4,15] deals with occlusions by estimating positions and velocities of the objects participating to the occlusion. After the occlusion, the correct object identities are re-established using estimated object locations. In [15] the estimation is performed by Kalman filters. In these systems if during the occlusion one of the objects suddenly changes its direction the estimate will be inaccurate, possibly leading to a tracking error. In [4] when there is a merge between regions, the position of the single regions are the same of the merged region that is the objects belonging to the region take the coordinates, the center, etc. of this latter. In all the works of this group, statistical features measured before the occlusion begins are used to resolve the labels after the occlusion; however, the systems is not able to decide about which pixels belong to which object during the occlusion event.

A different framework used in object tracking is layered image representation [18], in which image sequences are decomposed into a set of layers ordered by depth, along with associated maps defining motions, intensities and opacities. The layer ordering, together with the shape, motion, appearance of all layers, provide complete information for occlusion reasoning. The drawback of these algorithms is the high cost in terms of computational complexity of the algorithm that assigns each pixel to the correct layer. In order to reduce the cost, approximate solutions to the layer assignment problem are described in [18].

Most of the published works [1, 5, 6, 9, 11, 12, 13, 16,17] construct appearance models of the tracked objects so that their identity can be preserved through occlusion events. In [1] a feature-based method is used. In order to handle occlusions, instead of tracking entire objects (specifically vehicles), object sub features are detected and tracked. Then, such features are grouped into objects, using also motion information derived from the tracking. This approach however is very computationally expensive; also, it cannot be easily extended to non-rigid objects (such as persons). The works of Haritaoglu et al. [6] and Wren et al. [17] present a system to recognize people. Both use silhouettes to model a person and his parts; however [17] assumes that there is only a single person in the scene while [6] allows multiple person groups and isolated people. During an occlusion, a group (a detected moving region) is segmented into its constituent individuals finding the best matching between people models and parts of the region. The other works [5, 9, 11, 12, 13,16] use color information to represent the objects when they are isolated (building the so-called appearance model of the object). This enables the correct segmentation of the objects when they are overlapping during an occlusion: the algorithms search for the labeling of the subregions that maximizes the likelihood of the appearance of the detected region given the models of the individual objects. In [16] the appearance model of an object is a template mask derived by shape and color information. In [11, 12,13] an object is modelled as a set of vertically aligned blobs (since the objects to detect are people) where a blob is a region with a coherent color distribution. These works differ from each another in the definition of the coherence criterion. In [5] two levels of the tracking are performed: a blob level tracking (in a graph-based framework) that is the tracking of the connected moving regions and an object level tracking in which an object consists of one or more blobs, and a blob can be shared among several objects. In [9] simple shape and color information are used because the work proposes to deal with low – resolution video sequences.

The method we propose can be included in the last category, since it is based on an appearance model. In particular, we model a region with a hierarchical decomposition represented as a graph pyramid; then, during an occlusion, for each region that represents a group of objects, we try to segment the region by matching its hierarchy with the hierarchies of the objects present before the occlusion. The advantage of our proposal with respect to similar techniques is that the matching is performed in a top down fashion: the algorithm starts with the topmost level of the hierarchies (that is, the less detailed one), iteratively resorting to lower, more detailed levels only in case of ambiguities. In this way, the computational cost is kept low in the average case, while retaining the ability to perform a precise segmentation in the infrequent difficult cases.

3 The Proposed Representation and Algorithm

Let us clarify some terms to better understand the description of the algorithm: we mean by term regions (or equally detected regions) the connected regions coming out by foreground detection step. With the term objects we mean the real objects of interest (people, car, etc.) present in a frame, so during an occlusion a single detected region can be composed by more objects. The simplest representation of a detected moving region is provided by its bounding box, which contains enough information

for tracking when there are no occlusions. Our method is based on a more complex representation, based on a graph pyramid, that in absence of occlusions retains the simplicity and effectiveness of the bounding box, but enables a more accurate object matching to take place during an occlusion. Namely, each moving region is represented at different levels of resolution using a graph for each level. At the topmost level, the apex of the pyramid, the graph is composed by a single node, containing as attributes the position and dimension of the bounding box, and the average color. At lowest level there is an adjacency graph, where the nodes represent single pixels, and the edges encode the 4-connected adjacency relation. The intermediate levels are obtained by a bottom-up process, using the classical decimation-grouping procedure described in [10], where color similarity is used to decide which nodes must be merged. Notice that the number of levels in a pyramid is not fixed, but depends on the color uniformity of the region. Each intermediate node represents a sub region of the whole region, and its attributes are the bounding box and average color of that sub region. Since the construction of the pyramids is an expensive task, it is performed only on the first frame for all the moving regions. Then, for objects not occluded, the pyramid at successive frames is incrementally derived by the one at the previous frame. Only when an occlusion happens, the corresponding pyramid needs to be recomputed from scratch.

During the tracking process, each node in the pyramid receives a label representing the identity of the object to which the corresponding (sub)region belongs. When a (sub)region contains parts of more than one object, the corresponding node is labelled as MULTILABEL.

Now lets turn our attention on the tracking algorithm. The goal of this algorithm can be stated as follows: given the labelled representation of a frame I_t and the representation of the next frame I_{t+1}, the algorithm should find a labelling for this latter that is consistent with object identities.

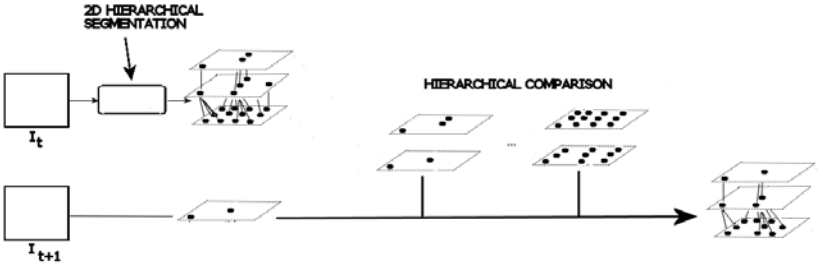


Fig. 1. Scheme of the Tracking Algorithm

To achieve this result, the algorithm proceeds as follows: after a first initialization step, in which the graph pyramid is computed for each region, and labels are assigned assuming that each region contains a single object, the algorithm compares the topmost levels of each pyramid in I_t with those in I_{t+1}. The way the comparison is performed is detailed in section 4. If the comparison outcome is sufficient to assign a label to each node, the algorithm terminates. Instead, if some ambiguities arise (as is

the case when two objects overlap), the algorithm is repeated using the next levels of the pyramids, until either a consistent labelling is found, or the bottom of the pyramid is reached. In this (infrequent) case, the algorithm outputs a "best effort" labelling. The whole process is sketched in Fig 1, while Fig. 2 illustrates the pyramidal representation of a toy image.

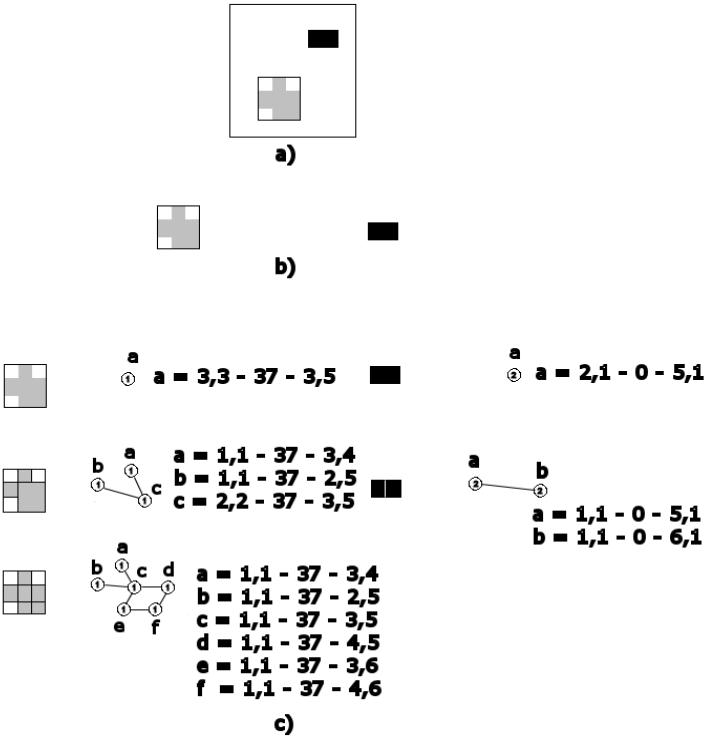


Fig. 2. The hierarchical representation a) The input image; b) the detected moving regions; c) the graph pyramids representing the detected regions

4 Comparison and Processing of Adjacent Frames

Let I_t and I_{t+1} be two consecutive frame images. At each level of the hierarchical representation there is a graph for each detected region, and the algorithm needs to compare each graph of I_t with each graph of I_{t+1}. In order to perform this comparison, we will first introduce a similarity measure between graph nodes, and then we will explain how this graph similarity is used in the comparison and labelling process.

As we told in previous section, each node is described by the dimension and the position of the bounding box and by the color (or the gray level, for monochrome videos). We have defined a color similarity $S_c \in [0,1]$, a dimension similarity $S_d \in [0,1]$ and a position similarity $S_p \in [0,1]$. These three functions are based on euclidean distance in the corresponding spaces, and are normalized so that their value is 1 if and

only if the corresponding features are equal. Then, we define the similarity between two nodes n and m as the product $S(n, m) = S_c \cdot S_d \cdot S_p$. In order to avoid to consider the matching between nodes that are too dissimilar, we have introduced a threshold T_2, and we truncate S to 0 (meaning that the nodes should never be matched).

We will now explain with more details how the comparison and labelling process is performed. First, at the current level of the comparison, the algorithm computes the similarities among all the nodes of graphs of I_t and all those of I_{t+1} that have not been labelled in the previous steps. Then an *association graph* is built. The association graph a bipartite graph where each node n of I_t is linked to each node m of I_{t+1} such that $S(n, m)>0$. Finally, a weighted bipartite graph matching (WBGM, see [8]) is performed, finding the matching between the nodes in I_t and those of I_{t+1} that maximizes the sum of the corresponding similarities. At this point, each pair (n, m) of matched nodes is examined. The similarity $S(n, m)$ is compared with a threshold T_1 (which is, clearly, greater than the previously defined T_2). If $S> T_1$, then the label of n is assigned to m. If this label is not MULTILABEL, it is also propagated to the descendant of m at the successive levels; otherwise, the descendants of m will receive the label of the corresponding descendants of n in a recursive process. If $S< T_1$, then the algorithm attempts a refined matching at the next level, marking the children of m and the children of all the nodes n' of I_t for which $S(n', m)>T_2$ as candidates for further processing in the next level of resolution. If a node m has no matching for which $S>T_2$, and it is at the first level of resolution, then it is considered as a new object, and a new label is generated for it. Otherwise, if this node it is at a more detailed level, then it is marked as a new sub region, and will be labelled later. The algorithm terminates when all the nodes of the current level receive a label, or the bottom level is reached. In this latter case, the algorithm uses the labelling derived by the found WBGM also when $S<T_1$. At this point, the nodes that have been expanded for processing at a higher resolution are labelled in a bottom-up process, propagating to the parent the label of the children, if they have the same label, or labelling the parent as MULTILABEL. During this process, also new sub regions are labelled, using the label of their sibling with the greatest similarity.

5 An Example

In order to clarify the algorithm, we will now present its phases with reference to a toy problem. In Fig. 3 are shown the two adjacent frames that we will use for this example, together with their representation. As it can be seen, there are two objects (A1 and A2) in the first frame, that in the second frame overlap forming a single region B1.

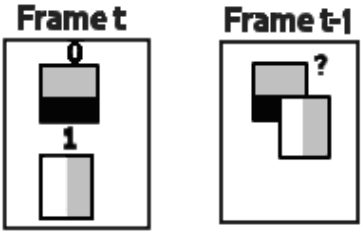


Fig. 3. The example: two adjacent frames with the moving object detected

Fig. 4 shows the comparison at the first resolution level. Both S(A1, B1) and S(A2, B1) are above T_2, but none of them is above T_1, so the node representing B1 cannot yet receive a label, and must be expanded together with A1 and A2.

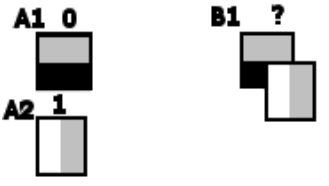


Fig. 4. The example: comparison at the first resolution level

Fig. 5 shows what happens at the second resolution level. Now the pairs (A11, B11), (A22, B14) and (A21, B13) have a similarity greater than T_1, and so can be labelled. However, the similarity between A12 and B12 is still under the threshold T_1, so these two nodes need to be further expanded.

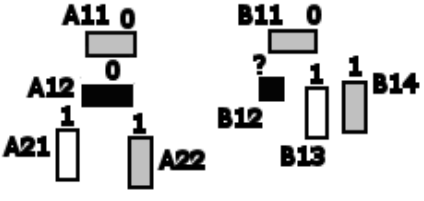


Fig. 5. The example: second resolution level

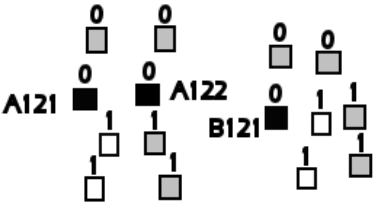


Fig. 6. The example: last level matching

The third level matching is depicted in Fig. 6. Now A121 is matched with B121 having $S>T_1$, while A122 is left unmatched. So B121 receives a label, that is propagated to B12. Now B1 can be labeled as MULTILABEL.

6 Experimental Results

The goal of our effort was to develop a tracking system for handling occlusion. Given this focus, we report results on PETS 2001 database [7]. The PETS dataset, is a stan-

dard database to evaluate the performances of tracking systems. We perform our experimentation on test dataset 1, camera 1. In the video sequence there are moving people and cars that occlude each other. The same dataset is used by Fuentes [4] to evaluate his tracking algorithm. In Tab. 1 the characteristics of chosen video are shown: the video consist of 2688 frames in which there are about 400 frame that present occlusions.

Table 1. PETS 2001 characteristics

Dataset name	Number of frames	Percentage of occlusions	Environment
PETS 2001 Test Dataset 1 Camera 1	2688	15%	Outdoor

In order to provide a quantitative measure of the tracker performance we use the performance index P presented in [3]. In Tab. 2 experimental results are shown.

Table 2. Experimental results: Comparison with a standard tracking algorithm

Experiment	TP	TN	FP	FN	P
#1 Whole dataset on our algorithm	5776	14	0	124	0.979
#2 "Occlusions" sequence with our algorithm	58	2	0	10	0.857
#3 "Occlusions" sequence with a standard tracking algorithm	37	2	0	31	0.557
#4 While dataset on a standard tracking algorithm	5744	14	7	156	0.972

The first row are the values of performance for the whole dataset. In the whole video sequence the performances are very high. In the second row we show the performances on about one hundred frames in which there are only occlusions (in Fig. 7 three frames of this sequence with the detected objects are shown). The results are quite promising considering that in real situations the percentage of occlusions is relatively low; besides the performance is much greater than a simple algorithm that does not handle with occlusions (third row). Finally, we shown the results on the whole dataset of the simple tracking algorithm. The two algorithms are comparable. This proves the effectiveness of our algorithm that deals with occlusions also if the latter are, in percentage, not much with respect of the entire video sequence.

Following a more interesting comparison is shown. We compared the algorithm with another algorithm (Senior et al. [16]) that deals with occlusions. In [16] some indexes, suitable to the evaluation of this kind of algorithms, are shown. Besides Senior et al. use the same dataset. We computed the values of indexes for our algorithm and we compared them with those presented by the authors themselves. Tab. 3 shows the effectiveness of our approach: all the values of error indexes are much less than those of the algorithm [16].

Table 3. Experimental results: Comparison with another algorithm that deals with occlusions

	Senior algorithm [16]	**Our algorithm**
Track error fp	16/14	4/14
Track error fn	4/14	1/14
Average position error	5.51	0.24
Average area error	-346	-8.4
Average detection lag	1.71	0
Average track completeness	0.12	0.04

7 Conclusions

In this paper we discussed an object tracking algorithm based on a graph theoretic approach with a multi-resolution representation to handle with occlusions. We demonstrated that by using a multi-level representation of moving objects in the scene together with a graph-based approach, it is possible to deal with occlusions: the algorithm can recognize, in a unique connected moving region, the parts that belong to different objects.

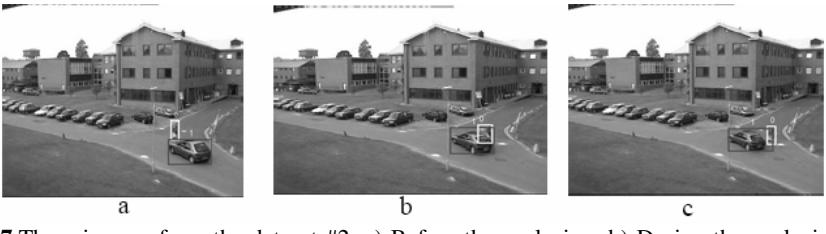


Fig. 7. Three images from the dataset #2. a) Before the occlusion; b) During the occlusion; c) After the occlusion

A future development of our proposed method will investigate on several enhancements of the algorithm: one of this is the increasing of the speed of the algorithm. In fact the execution time of the algorithm is relatively high because of the construction of the pyramid. The discussed idea of keeping the pyramid of an object for several frames is one of the possible solutions, but we have to investigate also other ones.

References

1. D. Beymer, J. Malik. “Tracking vehicles in congested traffic”. SPIE Vol. 2902, Transportation Sensors and Controls: Collision Avoidance, Traffic Management, and ITS, pp. 8 - 18. 1996.
2. A. Bobick, S. Intille, J. Davis, F. Baird, C. Pinhanez, L. Campbell, Y. Ivanov, A. Schtte, A. Wilson. “The KidsRoom: A perceptually-based interactive and immersive story environment”. PRESENCE: Teleoperators and Virtual Environments Vol. 8 – 4, p. 367- 391. 1999.

3. D. Conte, P. Foggia, C. Guidobaldi, A. Limongiello, M. Vento. “An Object Tracking Algorithm Combining Different Cost Functions”. LNCS, vol. 3212, pp. 614-622, 2004.
4. L. M. Fuentes, S. A. Velastin. “People tracking in surveillance applications”. Second IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, 2001.
5. S. Gupte, O. Masoud, R. F. K. Martin, N. P. Papanikolopoulos. “Detection and Classification of Vehicles”. IEEE Transactions on intelligent transportations systems, IEEE Computer Society Press. Vol. 3 – 1, pp. 37 - 47. 2002.
6. I. Haritaoglu, D. Harwood, L. S. Davis. “W^4: Real-Time Surveillance of People and Their Activities”. IEEE Transactions on PAMI. Vol. 22 – 8, pp. 809 – 830. 2000.
7. http://www.cvg.cs.rdg.ac.uk/PETS2001/pets2001-dataset.html
8. H.W. Kuhn, “The Hungarian Method for the Assignment Problem”, Naval Research Logistics Quarterly, Vol.2 pp.83-97, 1955.
9. P. KaewTrakulPong, R. Bowden. “A real time adaptive visual surveillance system for tracking low-resolution colour targets in dynamically changing scenes”. Image Vision Computing. Vol. 21 – 10, pp. 913-929. 2003.
10. J. M. Jolion, A. Montanvert. “The Adaptive Pyramid: A Framework for 2D Image Analysis”. CVGIP: Image Understanding, Academic Press. Vol. 55 – 3, pp. 339 – 348. 1992.
11. J. Li, C.S. Chua, Y.K. Ho, "Color Based Multiple People Tracking". Seventh International Conference on Control, Automation, Robotics and Vision, pp. 309-314, 2002.
12. S.J. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, and H. Wechsler. ``Tracking Groups of People". Computer Vision and Image Understanding. Vol. 80, pp. 42 – 56. 2000.
13. N. M. Oliver, B. Rosario, and A. Pentland. “A Bayesian computer vision system for modeling human interactions”. IEEE Transactions on PAMI, Vol. 22 – 8, pp. 831 – 843. 2000.
14. T.J. Olson, F.Z. Brill. “Moving Object Detection and Event Recognition Algorithm for Smart Cameras”. Proceedings DARPA Image Understanding Workshop, pp. 159-175, 1997.
15. R. Rosales, S. Sclaroff. “Improved Tracking of Multiple Humans with Trajectory Prediction and Occlusion Modeling”. Proceedings IEEE Conference on Computer Vision and Pattern Recognition. Workshop on the Interpretation of Visual Motion, 1998.
16. A. Senior, A. Hampapur, Y. L. Tian, L. Brown, S. Pankanti, R. Bolle. “Appearance Models for Occlusion Handling”. Proceedings of Second International Workshop on Performance Evaluation of Tracking and Systems. 2001.
17. C. R. Wren, A. Azarbayejani, T.J. Darrell, A.P. Pentland. “Pfinder: realtime tracking of the human body”. IEEE Transaction on PAMI, Vol. 19 – 7, pp. 780 – 785. 1997.
18. Y. Zhou, H. Tao. “A Background Layer Model for Object Tracking through Occlusion”. International Conference on Computer Vision, Nice, France, pp. 1079-1085. 2003.

Coarse-to-Fine Object Recognition Using Shock Graphs

Aurelie Bataille and Sven Dickinson

University of Toronto

Abstract. Shock graphs have emerged as a powerful generic 2-D shape representation. However, most approaches typically assume that the silhouette has been correctly segmented. In this paper, we present a framework for shock graph-based object recognition in less contrived scenes. The approach consists of two steps, beginning with the construction of a region adjacency graph pyramid. For a given region, we traverse this scale-space, using a model shock graph hypothesis to guide a region grouping process that strengthens the hypothesis. The result represents the best subset of regions, spanning different scales, that matches a given object model. In the second step, the correspondence between the region and model shock graphs is used to initialize an *active skeleton* that includes a shock graph-based energy term. This allows the skeleton to adapt to the image data while still adhering to a qualitative shape model. Together, the two components provide a coarse-to-fine, model-based segmentation/recognition framework.

1 Introduction

Object recognition is one of the primary goals of computer vision, allowing an image signal to be semantically labelled according to a priori knowledge of objects in the world. Early object recognition work in the 60's and 70's focused on the *categorization* or *generic object recognition* problem, in which exemplar objects, i.e., specific object instances, were matched to coarse, prototypical models designed to be invariant to within-class shape and appearance deformation. Although an admirable goal, the low- and intermediate-level infrastructure did not exist to bridge this representational gap [3, 4], leading to systems tested on contrived scenes under contrived viewing conditions.

Over the next 30 years, in a drive toward the recognition of more realistic objects under more realistic imaging conditions, recognition systems became more exemplar-based, beginning with the CAD-based vision systems of the 80's, then moving toward the appearance-based models of the 90's and the recently popular interest-point models. For the first time, complex objects can be recognized in cluttered scenes under varying illumination. However, since such systems are based on the distinguishing local textures of objects and not their prototypical shape, they are ineffective for generic object recognition.

Different object exemplars belonging to a single class may have different color, texture, exact geometry, and part articulation. But at some coarse level of

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 203–212, 2005.

description, the exemplars in a class have similar shape. It is for this reason that the generic object recognition community has focused primarily on shape as *the* class-defining feature. Moreover, the silhouette has emerged as a popular feature with which to characterize the shape of an object. Unlike extracted contours internal to the object, which may reflect either shape or texture, the occluding contour of the object is guaranteed to reflect only shape information. Since the occluding contour depends on viewpoint, 3-D object recognition systems are view-based, in which each generic object is represented as a set of characteristic silhouettes. Provided that an imaged object's silhouette can be extracted from an image, it is matched to a database of silhouettes grouped by object. The closest matching silhouette defines both the identity of the object as well as its pose (depending on the sampling resolution of the viewing sphere over which the silhouettes are captured).

An exact characterization of a silhouette would be appropriate for exemplar-based object recognition. However, since our goal is generic object recognition, we require a silhouette-based representation that is invariant not only to scale, translation, rotation, and occlusion, but part articulation, within-class shape deformation, and minor rotation in depth. The shock graph [11] has emerged as a powerful, generic shape description possessing these properties, and is based on a labelling and partitioning of the skeleton points (shocks) making up the medial axis transform of a shape. Shocks are labelled according to four qualitatively-defined classes, with contiguous clusters of homogeneously labelled shocks comprising the nodes in a shock graph. In the last 5 years, shock graphs have led to a number of successful silhouette-based recognition systems based on graph matching, e.g., [12, 8, 13, 9, 6, 5].

A careful examination of the shock graph-based recognition literature will show that most, if not all, approaches are typically applied to unoccluded, pre-segmented closed contours, with a only few approaches, e.g., [12, 9], tested on occluded shapes. Clearly, the shock graph-based recognition community has focused more on the shape description and matching problem and less on the segmentation of the shapes. The shock graph community has therefore, and understandably, met with resistance and skepticism from those who claim that since region segmentation is an unsolved problem, and since the occluding contour (silhouette) of an object requires that the object's region be correctly segmented, the whole notion of a shock graph rests on a weak foundation.

One can argue that the space of region segmentation errors is equivalent to the space of possible occlusions, for region over-segmentation can be modelled as an undetectable occluder (yielding a truncated silhouette) and region under-segmentation can be modelled as the union of a detectable occluder and the target object (yielding a silhouette that extends beyond the object). However, even though this argument has been made, e.g., in [12, 10, 6], it has not been made convincingly with extensive simulation of segmentation errors. Until testing is performed on real images of real objects, with massive over- and under-segmentation, the shock graph recognition framework will remain on the fringe of the object recognition community.

Occlusion testing in the shock graph community typically involves subjecting a target shape to minor occlusion. The shock graph is a distributed representation, with nodes corresponding to distinct parts, and one would expect that the occlusion of one part will not affect the representation of another. Although this is indeed true for well-separated parts on an object, occlusion can, in fact, result in major changes in the topological structure of an object's skeleton, yielding major changes in its shock graph structure. Thus, in the presence of significant region over- and under-segmentation, the resulting region's shock graph may bear little resemblance to the model shock graph to which it should match. Since one cannot guarantee that region segmentation errors are minor, we need an approach that couples object recognition using shock graphs with the underlying region segmentation problem, yielding a segmented shape that closely resembles a model object. This paper addresses this problem, and proposes a two-part solution.

2 Region Segmentation and Description

We begin by constructing a region adjacency graph pyramid or scale space, based on varying the parameters of the region segmentation algorithm (Comaniciu and Meer [1]). By varying the segmentation parameters, we can obtain a variety of segmentations, from heavily under-segmented to heavily over-segmented. The resulting regions at a given level may not correspond to objects in the image due to segmentation errors. However, the correct boundary of a given object may, in fact, span multiple scales. Each region segmentation level yields a region adjacency graph, with nodes representing region boundaries and edges specifying region adjacency.[1] Each node (region boundary), in turn, is represented by a shock graph. The region adjacency graphs are linked together to form a tree or pyramid, with a node at a coarser level pointing to one or more component nodes at the next finer level.

3 Model-Based Region Grouping

Given our pyramid of region adjacency graphs, our goal is to try and segment and recognize the object(s) in the scene. Using a model hypothesis for a given region in the image, we will search the space of possible merges of adjacent regions, at different scales, in an effort to strengthen the hypothesis beyond some appropriate threshold. We proceed in a top-down manner, starting with hypotheses for regions at coarser levels before proceeding to region hypotheses at lower levels. By merging adjacent regions at different scales, we consider the space of discrete "outward" perturbations of a region's shape, whereas by descending to a lower level when generating hypotheses, we consider the space of discrete "inward" perturbations of a region's shape. Such perturbations amount to moving

[1] We gratefully acknowledge the region adjacency graph construction module provided by Sven Wachsmuth.

between adjacent levels in the pyramid, and we use model hypotheses, invoked by region shape, to guide the traversal of the pyramid.

The model hypotheses are generated according to the framework described in [6], as shown in Figure 1. For each region at the coarsest level, the region's shock graph is indexed into the database of model shock graphs, returning a small set of promising candidates. These candidates, along with their similarity to the model (computed by a matching algorithm), form the initial "open list", sorted by a cost function, in the traditional graph search algorithm (Nilsson [7]). As shown in the algorithm described in Figure 2, the first element, or state, on the list is removed and tested to see if it's a solution. If not, the state is expanded to yield a set of successor states, in this case the set of possible merges of adjacent regions at the current or finer scales. If any of these successors results in a region whose shock graph is closer to the model than the expanded hypothesis, the successor is merged onto the open list (according to its evaluated cost). Although the algorithm terminates when a solution has been found, it may continue if there are other objects in the scene, i.e., regions not accounted for.

To illustrate the generation of successors, consider the example shown in Figure 3, depicting three levels of segmentation. Consider the red ellipse at the coarsest level. It's successors at that level include its merge with the light blue region to the left, and its merge with the pink region to the right. Its footprint is shown in levels 2 and 3 by the dotted lines. At level 2, its two successors are shown with black outline, while at level 3, its three successors are also shown with black outline. Only those successors that improve the quality of the match between the region's shock graph and the hypothesized model's shock graph, are added to the open list; the rest are discarded. Finally, the cost function used to rank the states on the open list governs the order in which hypotheses are considered. In our experiments, we adopt a "best-first" approach, in which the most promising hypotheses are expanded first, regardless of which levels their component regions are drawn from. We have also explored a "breadth-first" strategy, which favors hypotheses at coarser levels, as well as a "depth-first" strategy, which favors hypotheses at finer levels.

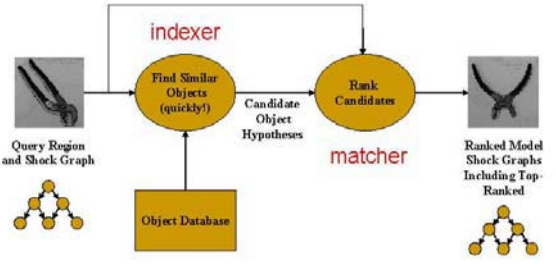


Fig. 1. Generating Model Shock Graph Hypotheses for a Given Image Region Shock Graph

```
for level = 0 to maxLevel do {consider each level, from coarse to fine}
  for all region R of level do {match all regions at the current level}
    OPEN.push(match(R))
  end for
  while OPEN not empty do {if open is empty, go to the next level}
    sort(OPEN, f) {sort open given the cost function selected}
    (R, M) = OPEN.pop() {Consider the first element of the open list}
    CLOSED.push((R, M)) {Move it to the closed list.}
    if (R, M) is a solution then {if a solution is found, exit with success}
      exit((R, M))
    end if
    for lev = level to maxLevel do {if it is not a solution, expand its children at each finer level}
      R' = footPrint(R, lev) {get the footprint of the region}
      ADJREGIONS = adjRegions(R') {get the regions adjacent to it}
      for all region R'' in ADJREGIONS do {try merging each adjacent region with the
      "footprint"}
        R* = merge(R', R'')
        if sim(R*, M) > sim(R, M) then {add the merge only if it improves the similarity with
        the model}
          OPEN.push((R*, M))
        end if
      end for
    end for
  end while
end for
exit("NO_SOLUTION") {if did not exit earlier, no solution was found}
```

Fig. 2. Algorithm for Performing Model-Based Region Grouping

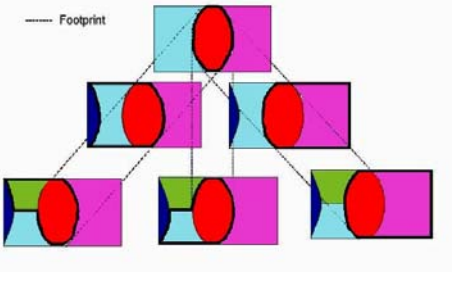


Fig. 3. Successor Generation (see text for explanation)

4 Model-Based Region Fitting

The search space for our model-based merging process is clearly richer than a single region segmentation. However, there may not exist a region segmentation parameter setting (or at least one sampled in the construction of the scale space) that recovers part of an object's boundary. In this case, no amount of model-based merging will recover that part of the boundary. It is here that we use whatever matching contour we have accumulated and return to the image in an attempt to find the contour. Just as we used a shock graph to guide our search through the space of possible region segmentations, we will again use a shock graph to guide our search for the missing contour, in an effort to fine-tune our region to be even closer to the model.

We employ an active contour-like approach, and build in model constraints *based on the shock graph*. Thus, like a traditional active contour approach, the

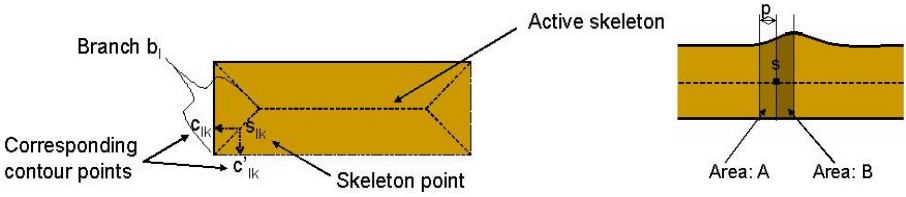


Fig. 4. Region Refinement using an Active Skeleton. Left: active skeleton. Right: the integrated radius function over small windows to the right (B) and left (A) of a skeleton point are used to define the shock graph energy term. For example, in the case of the constant cross-section type 3 node, the energy term would be proportional to the absolute value of the difference in areas

contour is data-driven by gradient structure in the image. However, we diverge from the traditional active contour in two important ways. First, we introduce the concept of an *active skeleton*, to which external (image gradient-based) and internal (smoothness) forces apply. Second, we introduce an energy term to the active skeleton energy minimization that ensures that its shape, while adapting to the image data, conforms to the model shock graph.

A hypothesis emerging from our model-based region segmentation step defines an explicit correspondence between branches (nodes) in the shock graph corresponding to the region group and branches in the model shock graph. Each corresponding branch pair defines a set of corresponding contour points, since each skeleton point defines a pair of contour points, as shown in Figure 4 (left). These matching contour points are used to align the model silhouette to the region group boundary. The same transformation is used to project the model skeleton into the image, representing our initial active skeleton. As image gradient "forces" attract the model silhouette, the positions and/or radii of the active skeleton points are updated to better fit the boundary data. As is common in active contour formulations, the active skeleton is subject to internal smoothness constraints.

Our second departure from traditional active contours is the explicit incorporation of model shape information as a deformation constraint. Since we know the qualitative shape class of a model branch, we can penalize changes in skeleton point position and/or radius that deform the branch shape out of its model class. An example of this additional energy term is shown in Figure 4 (right), in which an energy term, proportional to the slope of the radius function over a window, is used to maintain the branch's type 3 (constant radius) shape. In this way, a qualitative shape model can be folded into an active contour (in this case, skeleton) formulation, providing much stronger deformation constraints.

Our algorithm for region refinement is shown in Figure 5. We loop through each skeleton point on each branch, sampling nearby positions and radii and updating the position and radius of the point if its energy decreases. Once all skeleton points have been visited once, a branch adjustment is performed, allowing an updated branch to "pull" any connected branches in order to maintain the

while there is a branch that hasn't converged and MaxIterations have not been exceeded **do**
  for l in $\{1, ..., m\}$ **do** {loop on branches}
    if $move[l]$ **then** {consider only the branches that haven't converged yet}
      $curE = 0$
      for k in $\{1, ..., n\}$ **do** {loop on skeleton points}
        $E_{min} = infinity$
        for all s in $U(s_k)$ **do** {consider points in s' neighbourhood}
          for all r in $R(s)$ **do** {radius variation}
            compute the locations of the corresponding contour points c and c' given s and r
            $E'(s) = \alpha * E_{sm1}(s) + \beta * E_{sm2}(s) + \gamma * E'_{im}(s) + \delta * E_{shock}(s)$ {compute energy at s}
            if $E'(s) < E_{min}$ **then** {check if it is the minimum energy; if it is, store the skeleton and contour point locations}
              $E_{min} = E'(s)$
              $s_{min} = s$, $c_{min} = c$, $c'_{min} = c'$
            end if
          end for
        end for
        $curE = curE + E_{min}$
        move s to s_{min} {move skeleton and contour points to the location that minimizes the energy}
        move c to c_{min}
        move c' to c'_{min}
      end for
      move skeleton points connected to the branch given branch junction
      if $|prevE[l] - curE| < minE$ **then** {if the new energy did not improve by much, the branch converged}
        $move[l] = 0$
      else {no convergence yet}
        $prevE[l] = curE$
      end if
    end if
  end for
end while

Fig. 5. Refining the Model using an Active Skeleton

connectivity and integrity of the active skeleton network. Here, we draw on the concept of an active contour network ([2]), in which a set of active contours are connected at junctions using spring forces. The algorithm then iterates, visiting each branch a second time, unless the branch has converged. When all branches have converged, the algorithm terminates.

5 Results

To evaluate the model-based merging procedure, we captured model silhouettes for a variety of views for each of 13 different objects. These objects, as well as different objects belonging to the same set of object classes, were imaged in 15 test scenes, examples of which are shown in Figure 6. Each test scene was region segmented at four levels, resulting in a region segmentation pyramid. Shock graphs were computed for each region and models hypothesized. To evaluate our algorithm, the correctness of the labelled regions (determined from ground truth) was compared to the best results obtained if one were to opportunistically choose from the set of four region segmentations that which yielded the best results without region grouping.

In terms of degree of improvement, in the worst case (best baseline segmentation), 33%, and in the best case (worst baseline segmentation), 45% of the

Fig. 6. Examples of Test Images

hypotheses were incorrect in the baseline system, and became correct under our framework, while there was a 0% change in the other direction. Our system is therefore improving the recognition process considerably. Moreover, among those hypotheses that stayed correct, the majority improved their matching score, with a significant number passing over the solution threshold.

We now demonstrate the model-based region fitting process. Figure 7 shows the four levels of region segmentation used to construct the region adjacency graph pyramid for an input image containing a pair of scissors. Figure 8 illustrates the process of determining that portion of the region group which matches the model and therefore participates in the aligning transformation. Finally, Figure 9 illustrates the initial model aligned in the image, along with the final resting position of the active skeleton. Despite the presence of ambiguous contours in the image, the active skeleton adapts to those contours which preserve the shock graph labels of the individual branches. The classical, somewhat weak active contour formulation is thus strengthened to include a much more flexible shape model that, unlike statistical (active) shape models, needs no extensive training while supporting full articulation and within-class deformation.

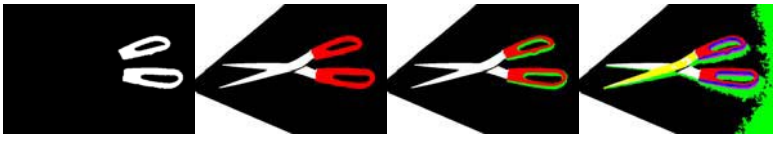


Fig. 7. The four segmentation levels of a test scene containing a scissors

Fig. 8. The points used to compute the model alignment. Left: result of the model-based region grouping, with region group (in green) and matched shock graph (skeleton in blue). Middle: silhouette of model shock graph. Right: those portions (red) of the region group skeleton that match the model shock graph, along with their corresponding contour points (yellow) used to compute the aligning transformation

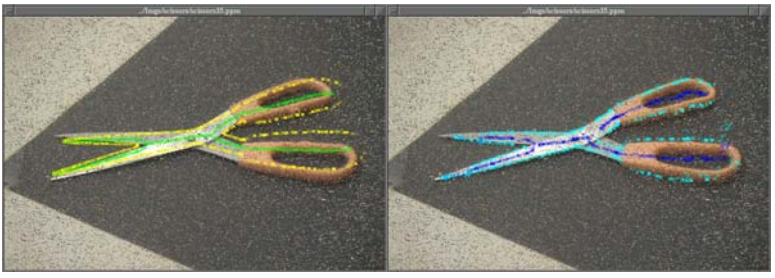


Fig. 9. Model-Based Region Fitting using an Active Skeleton. Left: initialized active skeleton (green) with corresponding contour points (yellow). Right: final active skeleton (dark blue) with corresponding contour points (light blue)

6 Conclusions

Shock graphs offer a powerful framework for representing and matching qualitative shape. Unfortunately, little effort has been devoted to their use in realistic scenes in which a silhouette cannot be properly segmented. In this paper, we attempt to address this problem, drawing on the assumption that since shock graphs do provide locality of representation, portions of regions that are properly segmented provide important clues as to what object is present (i.e., accounts for a particular region). Introducing a region segmentation hierarchy, we can use this model hypothesis to guide a search through a large space of possible splits and merges of the regions. The resulting grouping may, in fact, span many levels of the segmentation hierarchy.

We apply a standard state space search algorithm, and have explored a number of heuristics for ordering the search. In comparing the results to a baseline single region segmentation, we found that our approach often found the correct hypothesis whereas the baseline system did not, and that baseline correct hypotheses improved significantly in our approach. Preliminary results indicate that our multiscale, model-based region grouping framework significantly improves object recognition. Moreover, it is based entirely on a shock graph representation and matching framework, offering hope that shock graphs can be used under more realistic imaging conditions.

The model-based merging framework can be thought of as a mechanism for guiding the search through a discrete space of large-scale perturbations. Unfortunately, there is no guarantee that the correct shape exists in this space, requiring that we return to the image to explore a continuous space of fine-scale perturbations. In the second part of the paper, we again draw on the shock graph, but this time, we use it to constrain an active contour that will settle on the data subject to maintaining a qualitative shape model. We introduce the notion of an active skeleton, an active contour that represents the skeleton and adapts to the image data. Moreover, we add an energy term that keeps the individual skeleton "parts" from deviating from their specified model classes.

References

1. D. Comaniciu and P. Meer. Robust analysis of feature spaces: Color image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 750–755, 1997.
2. S. Dickinson, P. Jasiobedzki, G. Olofsson, and H. Chri stensen. Qualitative tracking of 3-d objects using active contour networks. *Computer Vision and Pattern Recognition*, pages 812–817, June 1994.
3. Y. Keselman and S. Dickinson. Bridging the representation gap between models and exemplars. In *IEEE Computer Society Workshop on Models versus Exemplars in Computer Vision*, Kauai, Hawaii, December 2001.
4. Y. Keselman and S. Dickinson. Generic model abstraction from examples. In *IEEE Conference on Computer Vision and Pattern Recognition*, Kauai, Hawaii, December 2001.
5. S. Kosinov and T. Caelli. Inexact multisubgraph matching using graph eigenspace and clustering models. In *9th International Workshop on Structural and Syntactic Pattern*, 2002.
6. D. Macrini, A. Shokoufandeh, S. Dickinson, K. Siddiqi, and S. Zucker. View-based 3-D object recognition using shock graphs. In *Proceedings, Internal Conference on Pattern Recognition*, Quebec City, August 2002.
7. N. J. Nilsson. *Principles of Artificial Intelligence*, chapter 2, Search Strategies for AI Production Systems. Tioga Publishing Co., 1980.
8. M. Pelillo, K. Siddiqi, and S. W. Zucker. Matching hierarchical structures using association graphs. In *European Conference on Computer Vision*, volume 2, pages 3–6, Freiburg, Germany, 1998.
9. T. B. Sebastian, P. N. Klein, and B. B. Kimia. Recognition of shapes by editing shock graphs. In *IEEE International Conference on Computer Vision*, pages 755–762, 2001.
10. A. Shokoufandeh, S. Dickinson, K. Siddiqi, and S. Zucker. Indexing using a spectral encoding of topological structure. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 491–497, 1999.
11. K. Siddiqi and B. Kimia. Toward a shock grammar for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1996.
12. K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker. Shock graphs and shape matching. In *ICCV*, pages 222–229, 1999.
13. A. Torsello and E. R. Hancock. Computing approximate tree edit-distance using relaxation labelling. In *Workshop on Graph-based Representations in Pattern Recognition*, pages 125–136, 2001.

Adaptive Pyramid and Semantic Graph: Knowledge Driven Segmentation

Aline Deruyver [1], Yann Hodé [2], Eric Leammer [3], and Jean-Michel Jolion [4]

[1] LSIIT,UMR7005CNRS-ULP,
Parc d'innovation, Bd Sébastien Brant, BP10413, 67412 Illkirch CEDEX.
aline@eavr.u-strasbg.fr
[2] FORENAP, CH Rouffach,
[3] Laboratoire PHASE, CNRS, 23 Rue du Loess, Strasbourg
[4] Laboratoire Reconnaissance de Formes et Vision, Bât. J. Verne INSA,
20 av. Albert Einstein, Villeurbanne
Jean-Michel.Jolion@insa-lyon.fr

Abstract. A method allowing to integrate syntactic and semantic approaches in an automatic segmentation process is described. This integration is possible thanks to the formalism of graphs. The proposed method checks the relevancy of merging criteria used in an adaptive pyramid by matching the obtained segmentation with a semantic graph describing the objects that we look for. This matching is performed by checking the arc-consistency with bilevel constraints of the chosen semantic graph. The validity of this approach is experimented on synthetic and real images.

1 Introduction

In image segmentation, human judgement is often required to control the quality of the result and to tune the segmentation parameters. This judgment may be seen as a semantic process. Introducing a semantic analysis in a segmentation process may strongly improve this segmentation. But how to do it?

The graph representation is a very helpful tool to reach this goal. At the level of image content, many objects may be described by semantic graph whose nodes represent parts of an object and the arcs represent spatial constraints between object subparts [1],[3],[5],[6],[14],[16]. Up to now, the limitation of many works on semantic graph lies in the necessity of starting with a correctly segmented image such that it can be labeled as in [1]. Unfortunately, obtaining a correct segmentation remains an open problem for many images of real life. A way to work with graph representation at the level of segmentation (from the level of pixels to the level of regions), is to work with adjacency graph. This approach is chosen by many authors due to some interesting properties which make this formal representation very convenient to describe an image. Performing a segmentation is possible, for example, by building a pyramid of adjacency graph [2],[7],[8],[9],[11],[12]. In this kind of process the nodes representing pixels or set of pixels are merged in a succession of steps to produce a pyramid of graphs whose nodes represent meaningful regions. The merging process

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 213–222, 2005.

may be controlled by several local factors with the hope that the ultimate merging will provide a meaningful result. So the question is, how it is possible to introduce an auto tuning in the merging process such that it will check automatically the meaningfulness of the result or in other words the semantic compatibility? We note that the representation of adjacency graph is close to the one of semantic graph. The difference is that in adjacency graph the nodes are regions of pixels and in semantic graph, the nodes are components of the image semantic content. Then, it seems natural to combine this two types of representation. We propose to apply a semantic judgment on the ultimate adjacency graph obtained after a low level segmentation process. This can be done by verifying if the adjacency graph may be matched with the semantic graph. We present in this paper an approach of segmentation using a pyramidal merging process whose control criterion is automatically tune by feedback with a semantic final checking. In section 2, we introduce the notions of semantic graphs and arc-consistency checking. The implementation of complex spatial and morphological constraints will be studied. In section 3, the knowledge driven segmentation algorithm will be detailed. A set of experiments on synthetic and real images will be presented in section 4. Section 5 will comment and conclude this study.

2 Semantic Graphs and Arc Consistency Checking

High level interpretation of images consists usually in matching each part of an image with a meaningful representation. To perform the matching between a graph and the different subparts of a shape it is necessary to check the global consistency of the graph. Unfaithfully, it is a NP-complete problem. However, it is often possible to reduce the time complexity of this process by only taking into account local constraints: checking the arc-consistency. Several authors [3][14][16] proposed fast arc-consistency checking algorithms. These algorithms try to associate only one value with one node. In high level interpretation of images, this assumption supposes to have an ideal segmentation (with one node of the graph is associated only one region). In practice it is very rare to obtain such a segmentation. It is the case for each segmentation stage of a pyramid. Then if we want to have a correct semantic analysis of an over segmented image, it is necessary to associate with one node a set of regions. In the following sections the notions of semantic graphs and arc consistency checking applied to oversegmented images will be defined. Then some ways to describe complex spatial relations are proposed. In the last sub-section the implementation of the arc-consistency checking algorithm with bilevel complex constraints is described.

2.1 Semantic Graphs

In semantic graphs, the high level constraint represented by the edges are supposed to be known at the beginning of the matching process. These constraints are not created during the computation and are imposed by the application. Then, it is assumed that some specific constraints exist in the image and the aim is to find sets of regions satisfying these constraints (See Figure 1). We use the following conventions:

- Variables are represented by the natural numbers 1, ... n. Each variable i has an associated domain D_i. We use D to denote the union of all domains and d the size of the largest domain.
- All constraints are binary and relate two distinct variables. A constraint relating two variables i and j is denoted by C_{ij}. $C_{ij}(v,w)$ is the Boolean value obtained when variables i and j are replaced by values v and w respectively. Let $\mathcal{R}$ be the set of these constraining relations.

A Finite-Domain Constraint Satisfaction Problem (FDCSP) consists of finding all the sets of values $\{a_1, ..., a_n\}$, $a_1 \times ... \times a_n \in D_1 \times ... \times D_n$, for(1, ..., n) satisfying all relations belonging to $\mathcal{R}$. In this classical definition of FDCSP, one variable is associated with one value. This assumption cannot hold for some classes of problems where we need to associate a variable with a set of linked values as described in [5][6]. To cope with this difficulty, we defined the Finite-Domain Constraint Satisfaction Problem with Bilevel Constraints ($FDCSP_{BC}$). It consists in introducing two levels of constraints, one level between each couple of nodes (spatial relations between objects associated with a node) and one level between each couple of regions classified inside one node (spatial relations between subparts of the object associated with a node).

Definition 1. Let $\mathcal{C}_{mpi}$ be a compatibility relation, such that $(a,b) \in \mathcal{C}_{mpi}$ iff a and b are compatible. Clearly $\mathcal{C}_{mpi}$ is reflexive. Let C_{ij} be constraint between i and j. Let be a pair S_i, S_j such that $S_i \subset D_i$ and $S_j \subset D_j$, $S_i, S_j \mapsto C_{ij}$ means that (S_i, S_j) satisfies the oriented constraint C_{ij}.

$S_i, S_j \mapsto C_{ij} \Leftrightarrow \forall a_i \in S_i, \exists (a'_i, a_j) \in S_i \times S_j$, such that $(a_i, a'_i) \in \mathcal{C}_{mpi}$ and $(a'_i, a_j) \in C_{ij}$
and $\forall a_j \in S_j, \exists (a'_j, a_i) \in S_j \times S_i$, such that $(a_j, a'_j) \in \mathcal{C}_{mpj}$ and $(a_i, a'_j) \in C_{ij}$.
Sets $\{S_1 ... S_n\}$ satisfy $FDCSP_{BC}$ iff $\forall C_{ij} \quad S_i, S_j \mapsto C_{ij}$.

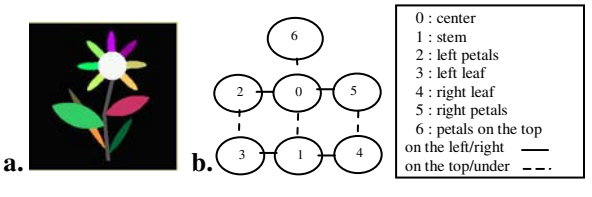


Fig. 1. A semantic graph (b.) describing a flower and a segmented image (a.). In this case, the domain D is made up of the set of regions of the image

A graph G is associated to a constraint satisfaction problem as follows: G has a node i for each variable i. Two directed arcs (i,j) and (j,i) are associated with each constraint C_{ij}. Arc(G) is the set of arcs of G and e is the number of arcs in G. Node(G) is the set of nodes of G and n is the number of nodes in G.

In the following, after having define the notion of intra-node relational constraints (corresponding to the notion of compatibility relation $\mathcal{C}_{mpi}$), we will focus on the notion of arc-consistency.

2.2 Intra-node Relational Constraints

Regions associated with a given node support themselves if their union satisfies constraints imposed on the node. Unfaithfully, checking this condition is a combinatorial problem. It can be helpful to soften the toughness of these conditions. We propose two weaker conditions:

- A local condition: two regions r_j and $r_k \in D_i$ (domain of node i) support themselves if it exists a union $A_{l,(r_j,r_k)}$ of regions $\in D_i$ such that r_j and $r_k \in A_{l,(r_j,r_k)}$, and such that the characteristics of $A_{l,(r_j,r_k)}$ are compatible (it means that the constraints are partially satisfied) with the constraints imposed on the node i.
- A global condition: let $A_{l,(r_j,r_k)}$ be defined as previously. Let be $r_j \in D_i$, let be $B_j = \bigcup_{r_k \in D_i, l \in N} A_{l,(r_j,r_k)}$ and $E = \bigcup_{r_n \in B_j, r_m \in D_i, p \in N} A_{p,(r_n,r_m)}$. The characteristics of E have to be compatible with the constraints of the node i. E is a set of regions corresponding to nodes of the largest connected graph describing the link between regions following relational constraints for the node i.

The Figure 2 illustrates these notions: Region P is the union of all the regions belonging to the object represented by the node i. P is not known and is the region to identify. We denote N_x any region satisfying intra node relational constraints (local condition) and belonging to P. We denote M_y any region satisfying intra node relational constraints and not belonging to P. We denote $E= ((\cup N_x) \cup (\cup M_y))$. We know that some of the following predicate are true $\{N_x \subseteq P, M_y \subseteq P\}$ but we do not know which ones. We also know that $P \subseteq E$. Checking if this last inclusion may be true corresponds to check for the global condition. The representation of these intra-node constraints will be described in the sequel.

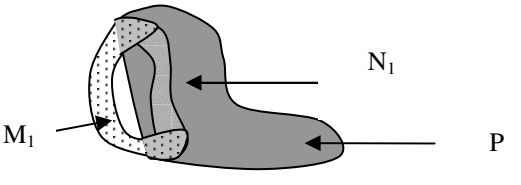


Fig. 2. Example illustrating the notions of weak conditions

Local Condition. The intra-node compatibility relation $\mathcal{C}_{mpi}$ can represent elementary relations like direct spatial relations (on the right, on the left, upon, under). However, in some cases, it cannot be enough to describe the compatibility between objects of a same node. For example, one could want to verify if a region is directly or indirectly upon or directly or indirectly under another region. It can be noticed that this relation can be described by a regular expression $a^* \cup b^*$. One could also want to limit the depth of the relation. For example, one region have to be under another region but it

must not be separated by more than two regions. It could be described by the expression b^3. Then, it is necessary to describe complex compatibility relations by using a combination of elementary relations. These relational descriptions of object was used in image analysis by several authors [4][13][15]. In our context, the description of these relations can be done with the words of a language LC_i. Its alphabet Σ_{Ti} is made up of the different spatial elementary relations. The words of this language describe the sequences of constraints to go from an element $a_1 \in D_i$ to an element $a_n \in D_i$ by the way of elements $a_j \in D_i$. As the relations are intentionally limited to relations that can be described by some regular expressions, LC_i can be recognized by a finite automaton. Let LC be the language made up of the set of words that can be generated from the elements of the alphabet $\Sigma_T = \{a_1, ...a_n\}$ where a_1 ... a_n are elementary relations. Each language LC_i associated with a node i of the graph is a subset of the language LC and describes how regions can be associated to make the object of the node i. It is reasonable to think that LC_i is a finite set of finite words. Then, LC_i can be recognized by a non deterministic finite automaton AF_i where $\Sigma_{Ti} \subset \Sigma_T$, $\Sigma_{Qi} = \{q_0, ..., q_{ni}\}$ is the set of states and $Q_{Fi} \subset \Sigma_{Qi}$ is the set of final states. Verifying if two regions E_1 and E_2 are compatible consists in finding if a word exists belonging to LC_i such that it is possible to reach the region E_2 from the region E_1. Then, a rewriting system with constraint $S_i(LC_i)$ can be defined on D_i. It allows to rewrite an element $a_1 \in D_i$ in an element $a_n \in D_i$ with respect to the language of constraint LC_i. The set of rewriting rules R_i of $S_i(LC_i)$ is defined by:

$R_i = \{l_j \rightarrow_{cj} r_j \mid l_j , r_j \in D_i,\ c_j \in \Sigma_{Ti},\ c_1 ...c_{j-1}\, c_j\, c_{j+1} ... c_n \in LC_i$ and $l_1 \vdash^*_{c1\, c2\, ...c\, j-1}\ l_j\}$

Finally verifying a complex intra-node constraint $\mathcal{C}_{mpi}$ can be assimilated to solving a reachability problem and such a constraint $\mathcal{C}_{mpi}$ will be defined by:

Definition 2. $\mathcal{C}_{mpi}(a,b,S_i(LC_i), D_i) \Leftrightarrow a \vdash^*_{Si(LCi)} b$

Global Condition. Let P be the union of all the regions belonging to the object represented by the node i. P is not known but we know that $P \subseteq P(Di)$. According to the definition of the set E given at the beginning of the section, $P \subseteq E$. Building the set E may need some calculations and to simplify the problem we propose to use the relation $P \subseteq P(Di)$ instead of $P \subseteq E$ to apply global constraint. This choice lowers the level of constraint but makes constraint satisfaction checking easier. $P \subseteq P(Di)$ cannot be checked because P is unknown. In an other hand the constraint on P are known. Then if $P \subseteq P(Di)$, the merging of all the regions of Di that we denote MDi , should be compatible with some constraints on the node i. For example: Surface(MDi)≥SurfaceMin (P), or Width(MDi)≥WidthMin (P) or Height(MDi)≥HeightMin(P).

The parameter Surface(MD_i), Width(MD_i), Height(MD_i) are easily calculated from the parameters of all the regions of D_i. If relations like those mentioned previously (surface, width and height) are not satisfied, then the regions of D_i do not satisfy the global intra-node constraints and the arc consistency fails.

2.3 Arc-Consistency Problem Applied to Over-Segmented Image

A class of problems called arc-consistency problems with bilevel constraints (AC_{BC}) is defined. It is associated with the $FDCSP_{BC}$ and it is defined as follows:

Let $\mathcal{P}(D_i)$ be the set of sub parts of the domain D_i.

Definition 3. Let (i,j) ∈ arc(G). Arc (i,j) is arc consistent with respect to $\mathcal{P}(D_i)$ and $\mathcal{P}(D_j)$ iff $\forall Si \in \mathcal{P}(Di)\ \exists Sj \in \mathcal{P}(Dj)$ such that $\forall v \in Si\ \exists t \in Si, \exists w \in Sj$, $\mathcal{C}_{mpi}(v,t)$ and $C_{ij}(t,w)$ (v and t could be identical).

Definition 4. Let $P= \mathcal{P}(D_1) \times .... \times \mathcal{P}(D_n)$. A graph G is arc-consistent with respect to P iff $\forall$ (i,j) ∈ arc(G): (i,j) is arc-consistent with respect to $\mathcal{P}(D_i)$ and $\mathcal{P}(D_j)$.

The purpose of an arc-consistency algorithm with bilevel constraints is, given a graph G and a set P, to compute P', the largest arc-consistent domain with bilevel constraints for G in P

2.4 Algorithm of Arc Consistency Checking with Bilevel Constraints

Implementation of the Arc-Consistency Checking Algorithm with Bilevel Constraints. Considering the previous remarks, we adapt the AC4 algorithm proposed by Mohr and Henderson in 1986 [3][14] to solve the AC_{BC} problem. We call this algorithm $AC4_{BC}$ (see [5] for the details of the algorithm).

In $AC4_{BC}$, a node belonging to node(G) is made up of a kernel and a set of interfaces associated with each arc which comes from another linked node. In addition, an intra-node compatibility relation $\mathcal{C}_{mpi}$ is associated with each node of the graph. It describes the semantic link between different subparts of an object which could be associated with the node. As in algorithm AC4, the domains are initialized with values satisfying unary node constraints and there are two main steps: an initialization step and a pruning step. However, whereas in AC4 a value was removed from a node i if it had no direct support, in $AC4_{BC}$, a value is removed if it has no direct support and no indirect support obtained by using the compatibility relation $\mathcal{C}_{mpi}$. The indirect supports are found thanks to the notion of interfaces. It has been proved that the $AC4_{BC}$ algorithm is correct and always terminates.

It has been also proved that the time complexity of the cleaning step is in $O(n^2d)$ in the worst case and that the time complexity of $AC4_{BC}$ is in $O(en^3d^2)$ in the worst case (n is the number of nodes and d is the size of the largest Domain D)

Implementation of Intra-node Compatibility Relations. The checking of intra-node compatibility relations is made in two steps. The first step checks if the local intra-node constraints are satisfied and the second step checks if the global intra-node constraints are satisfied. Section 2.2 has shown that intra-node spatial constraints (local constraints) can be easily verified thanks to a rewriting system Si (LCi). The implementation of this system can be done as follows: let $LC_i^n = \{\omega \in LC_i \mid |\omega| \leq n\}$ and $SR_i^n = \{ u \rightarrow_{Si(LCi^n)} v \mid u \in D_i \text{ and } v \in D_i \}$, let be GenereLC($LC_i$,n) the function generating the set LC_i^n, let the function GenereSR($S_i(LC_i^n)$, a) generating the set of the possible rewritings from the element a by applying $S_i(LC_i^n)$. The algorithm is:

```
n:=1 ; reach:=false; end:=false;
while not reach and not end   do
begin
if  a ⊢*_{Si(LCi^n)}  b   then    reach:=true;
else begin
```

```
        LC_i^{n+1} := GenereLC( LC_i ,n+1);
        SR_i^{n+1} := GenereSR( S_i (LC_i^{n+1}) , a );
        if  SR_i^n = SR_i^{n+1}     then     end := true;
        end
n:=n+1;
end
```

The intra-node global constraints (morphological constraints) are implemented as follows: After the first step, only regions satisfying the local spatial constraints are kept in the kernel of the node. Then a search of all the sets of connected regions is performed on the set of regions of the kernel. If a given set does not satisfy the global constraint associated with the considered node, all the regions of this set are removed from the node.

3 Knowledge Driven Segmentation Algorithm

Segmentation based on graph decimation process assumes that it is possible to define which regions are similar and which are not. Usually, the proposed method of decimation use statistical criteria computed on the grey levels of regions [12]. However, it cannot be always enough. Sometimes, expert knowledge describing the goal (the morphology and the spatial relations of each object that we look for) can be helpful to find the best segmentation. Spatial relations may be easily described in a semantic graph. Then, we integrate such a semantic graph in the decimation process of an adaptive pyramid [8]. The key point of this decimation process is the choice of the threshold deciding which regions are similar and which are not. In this study we consider that two regions cannot be merged if the difference of mean intensity is greater than a given threshold. The best threshold is the one providing a segmentation with the lowest number of regions and which is compatible with the knowledge described by the semantic graph. Since the obtained segmentation can be over-segmented, it is necessary to check the compatibility with the arc-consistency checking with bilevel complex constraint algorithm. The algorithm follows an iterative process. While the obtained segmentation provides an arc-consistent graph, the threshold is incremented. If the graph becomes un-consistent, we consider that the correct segmentation has been obtained with the previous threshold.

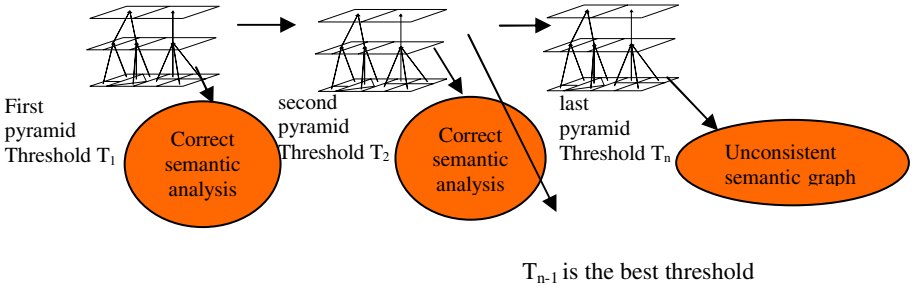


Fig. 3. Principle of the algorithm

The problem of the uniqueness of the solution have to be considered. In order to ensure this uniqueness, the constraints imposed in the semantic graph are defined in such a way that they are true on any subparts of an object as well as on the whole object. Then, if the graph is consistent at a given level of segmentation, it is consistent for any sub-level of over-segmentation.

4 Experiments

The algorithm has been tested on synthetic image representing a flower (see Figure 1-a) described by a semantic graph (see Figure 1-b). This graph represents the spatial relationship (on the left/on the right, on the top/under, etc…) of the different subparts (stem, leaves, petals, center) of a flower. The intra-node spatial constraints are of two kinds: "on the top/under" with a depth of two for the stem and "around of" with a depth of one for the other objects. An intra-node morphological constraints has also been imposed for the stem which must be a thin vertical object.

The synthetic image was intentionally over-segmented (see Figure 4-a) and it is made up of 32 regions. The aim is to retrieve the initial flower by merging the regions. Thanks to the arc consistency checking, the stem has been completely identified and the sleeves are correctly separated from the petals (see Figure 4-c). The regions belonging to the heart of the flower are identified as well, even if two petals are merged in the heart of the flower. The curves of the Figure 5 show that the threshold value equal to 32 gives the best segmentation with a low number of regions. If this value is increased the leaves are merged with the stem and the semantic graph becomes un-consistent.

Other experiments have been made on pictures of human faces. The aim is to recognize subparts corresponding to hair, eyes, mouth, skin and background. Although the floor of the pyramid can be at the level of pixels, it is more convenient to start with regions so long as they are small enough to avoid an unwilled sub-segmentation. The computation of the watersheds is a good choice to obtain these small regions. The semantic graphs is made up of 6 nodes and 63 arcs and 5 different kinds of spatial relations are used. On the first example (see Figure 7, 1st row) the graph is consistent until the threshold value is equal to 53. With this value we obtain a segmentation made up of 192 regions and each region are correctly labeled. With the just higher threshold, the right eye is merged with the skin and the graph becomes un-consistent. On the second example (see Figure 7, 2nd row) the threshold which gives the best understandable (arc consistent) segmentation is 71.

Fig. 4. a. over segmented image, b. the four stages of the pyramid with a threshold equal to 32 **c.** the labeled image after arc-consistency checking

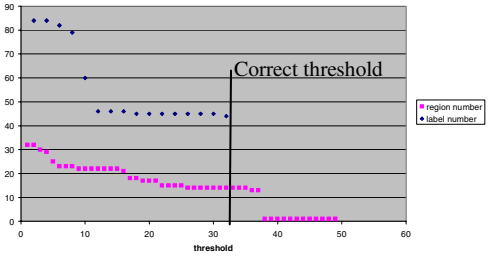


Fig. 5. Evolution of the number of regions and the number of labels with respect to the applied threshold

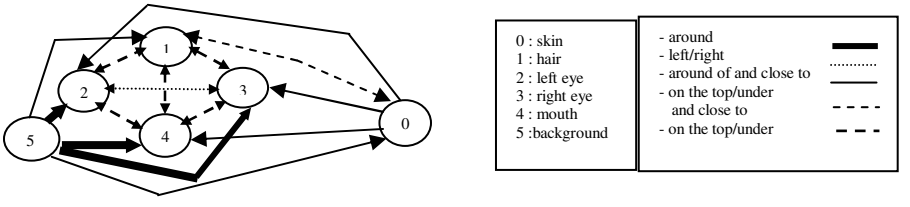


Fig. 6. Semantic graph describing a face

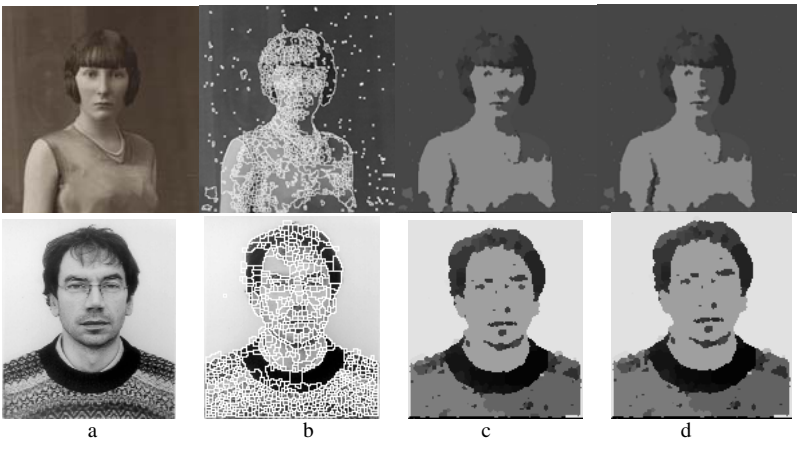


Fig. 7. Original image. b. segmentation obtained with a watershed algorithm. c. final segmentation with the optimal merging threshold d. image obtained with the just higher merging threshold: the semantic graph becomes un-consistent

5 Comments and Conclusion

To improve the process of image segmentation which is far from being resolved for the majority of images, our idea is to drive the segmentation process with a semantic control. The possibility to represent with graphs, spatial relations between regions and

spatial relations between subparts of an object is the key point of our approach. The semantic checking of the pertinence of the value of a given criterion chosen to drive the decimation process inside an adaptive pyramid, allows to select the best value. This choice gives an optimal result with the chosen criterion of segmentation. To our knowledge, it is the first time that a decimation process in an adaptive pyramid is driven by semantic criteria. The design of the semantic graph is in our case a crucial step. This step could become automatic if the endeavor of others group as [10] who try to extract automatically model graph from a set of images may produce reliable and efficient semantic graph representation of many kind of common objects.

References

1. Bauckage C., Braun E., Sagerer G., "From image features to symbols and vice versa – Using graphs to loop data- and Model-driven processing in visual assembly recognition", International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 497-517
2. Bertolino P., Montanvert A. "Multiresolution segmentation using the irregular pyramid" in proceeding IEEE ICIP96, Lausane Suisse, (1996) 357-360
3. Bessière C., "Arc-consistency and arc-consistency again", Artificial intelligence 65 (1991) 179-190
4. Bunke H., Sanfeliu A. Syntactic and structural pattern Recognition: theory and applications World Scientific Teaneck N. J. (1990)
5. Deruyver A., Hodé Y. "Constraint satisfaction problem with bilevel constraint: application to interpretation of over segmented images", Artificial Intelligence 93 (1997) 321-335
6. Deruyver A., Hodé Y., "Image interpretation with a semantic graph: labeling oversegmented images and detection of unexpected objects", in proceedings GBR 2001, Ischia Italie, Edition Cuen, (2001) 137-148
7. Glantz R., Pelillo M., Kropatsch W.G., "Matching segmentation hierarchies", International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 397-424
8. Jolion J.M. "Stochastic pyramid revisited", Pattern recognition letter 24 (2003) 1035-1042
9. Jolion J. M., Montanvert A., "The adaptive pyramid: a framework for 2D image analysis", CVGIP: Image Understanding 55 (3) (1992) 339-348
10. Keselmann Y., Dickinson S., "Generic Model Abstraction from Examples", submitted to IEEE Transaction on PAMI.
11. Laemmer E., Deruyver A., Sowinska A., "Watershed and adaptive pyramid for determining the apple's maturity state", in proceeding IEEE ICIP 2002, Rochester USA, (2002) 789-792
12. Lallich S., MuhlenbachF., Jolion J.M. "A test to control a region growing process within a hierarchical graph", Pattern recognition Letter 36 (10) (2003) 2201-2211
13. Ledley R.S. "High-Speed Automatic Analysis of biomedical Pictures", Science 146 (3461) (1964) 216-223
14. Mohr R., Henderson T., "Arc and path consistency revisited", Artificial Intelligence 28 (1986) 225-233
15. Shaw A.C., "Parsing of Graph-Representable Pictures", J. ACM 17 (3) (1970) 453-481
16. Van Hentenryck P., Deville Y., Teng C.M., "A generic arc-consistency algorithm and its specializations", Artificial Intelligence 57 (2) (1992) 291-321

A Graph-Based Concept for Spatiotemporal Information in Cognitive Vision*

Adrian Ion, Yll Haxhimusa, and Walter G. Kropatsch

Pattern Recognition and Image Processing Group 183/2,
Institute for Computer Aided Automation,
Vienna University of Technology, Austria
{ion, yll, krw}@prip.tuwien.ac.at

Abstract. A concept relating story-board description of video sequences with spatio-temporal hierarchies build by local contraction processes of spatio-temporal relations is presented. Object trajectories are curves in which their ends and junctions are identified. Junction points happen when two (or more) trajectories touch or cross each other, which we interpret as the "interaction" of two objects. Trajectory connections are interpreted as the high level descriptions.

1 Introduction

Even though there is no generally accepted definition of cognitive vision yet, presumptions about the cognitive capabilities of a system can be made by comparing it's results with that of an entity, already 'known' and accepted to have these capabilities, the human. Also, the *Research Roadmap of Cognitive Vision* [15], presents this emerging discipline as 'a point on a spectrum of theories, models, and techniques with computer vision on one end and cognitive systems at the other'. A conclusion drawn from the previous, is that a good starting point for a representation would bring together the following:

- enable easy extraction of data for human comparison;
- bridge together high and low level abstraction data used for cognitive and computer vision processes.

After 'watching' (analyzing) a video of some complex action, one of the things, that we would expect a cognitive vision system to do, is to be able to correctly answer queries regarding the relative position of occluded objects. Let us take the video[1] given by a simple scenario of two black cups and a yellow ball and describe the scene in simple English words (see the description in Table 1). The description contains: **objects**: hand, cup, ball, table ; **actions**: grasp, release, move, shift etc., and **relations**: to-the-left, to-the-right, in-front-of etc.

* Supported by the Austrian Science Fund under grant FSP-S9103-N04.

[1] http://www.prip.tuwien.ac.at/Research/FSPCogVis/Videos/Sequence_2_DivX.avi

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 223–232, 2005.

Later, we could use this kind of description to compare the results given by the system with ones made by humans. While observing a dynamic scene, an important kind of information is that of the change of an object's location, i.e. the change of topological information. In most of the cases, this kind of change is caused by an active object (e.g. agent: hand, gravity, etc) acting on any number of passive objects (e.g. cup, ball, etc.). Queries like '*where is the ball?*' could be answered if the history of topological changes is created.

From all the work done in the domain of qualitative spatial and temporal information we would like to enumerate the following: Interval calculus [1] is used in systems that require some form of temporal reasoning capabilities. In [1] 13 interval-interval relations are defined: 'before', 'after', 'meets', 'met-by', 'overlaps', 'overlapped-by', 'started-by', 'starts', 'contains', 'during', 'ended-by', 'ends' and 'equals'. In [13], motivated by the work in [1, 7, 8], an interval calculus-like formalism for the spatial domain, the so called region connection calculus (RCC) was presented. The set of 8 region-region base relations defined in [13] ($RCC-8$) are: 'is disconnected from', 'is externally connected with', 'partially overlaps', 'is a tangential proper part of', 'is non-tangential proper part of', 'has a tangential proper part', 'has non-tangential proper part', and 'equals'. A more expressive calculus can be produced with additional relations to describe regions that are either inside, partially inside, or outside other regions ($RCC-15$). Different graph based representations have been used to describe the changes/ events in a dynamic space. In [6] graphs are used to describe actions (vertices represent actions). Graphs are also used in [2], but here vertices represent objects. Balder [2] argues that arbitrary changes can be best described by state approach: the state of the world before and after the change characterizes the change completely. The Unified Modeling Language, in its state diagram, also defines a graph based representation for tracking temporal changes. The General Analysis Graph (GANAG) [14] is a hierarchical, shape-based graph that is build and used in order to recognize and verify objects. *The analysis graph can be seen as a 'recipe' for solving industrial applications, stating which kind of decisions have to be made at which stage* [14].

In Section 2 we give the spatiotemporal story-board of the video sequence. In Section 3 we describe two methods of contraction of trajectory of movements: first the spatial contraction followed by a temporal contraction (Section 3.1) and than the temporal contraction followed by a spatial contraction (Section 3.2).

2 Spatiotemporal Story Board of a Film

The scene history is a description of the actions and spatial changes in the scene. It should depict the spatiotemporal changes in the scene, in a way that could be used to create a human-like description (similar to the one presented in Section 4). For this we propose a graph based representation where vertices represent spatial arrangement states and edges represent actions (see Figure 1a).

Each vertex contains a topological description of the spatial arrangement of the objects in the scene, that results through a transition from a previous state,

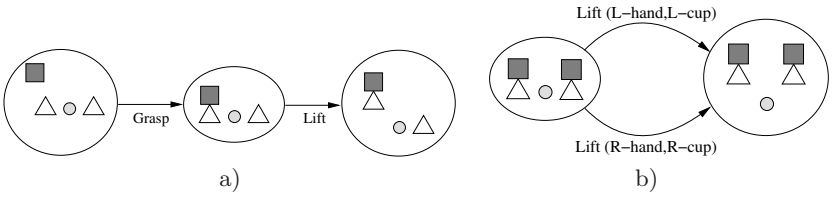


Fig. 1. a) History graph. b) Parallel actions. □ Hand, ◯ Ball, △ Cup

by applying the actions that link it to the current. What we refer to as objects are actually detected relevant visual entities, which in the ideal case would be objects or, groups of objects in a "special" physical relation e.g occluding, containing, etc. Vertices are added when the topological description of the spatial arrangement changes. There are no vertices that contain (identify) the same topological description (scene state). If the scene enters a state, which has a topological description identical to one of the descriptions already identified by a vertex in the scene history graph (it has been in the same state in the past), then an edge/edges from the vertex identifying the previous state, to the existing vertex should be added.

Edges are associated with actions and identify the type/class of the action. Also, each edge links to the objects (from the source and destination state vertex) involved in this particular action. If an object taking part in the action cannot be identified as one of the known objects, a new instance should be created and the edge linked to it. Later on, through reasoning, the new created instance, can be identified as a previously known object or a new one (or some presumption can be made, using certain criteria). In case of simultaneous actions, more than one edge is used to connect 2 vertices. Each edge should describe the actions that happened in parallel. (Figure 1b) shows how to describe 2 hands lifting 2 cups at the same time)

The representation of the scene history as a graph allows us to create higher level abstractions. A straight forward example results from the 're-usage' of vertices (disallowing multiple vertices identifying the same state). Imagine the scenario of a hand grasping and releasing the cup 10 times in a row. Besides saving space by not adding a big number of additional vertices, by identifying cycles, we can easily determine repeated actions and find the shortest way from one configuration to another. Higher level abstractions replace more complex subgraphs containing parallel actions and long sequences of actions resulting in small or unimportant changes for the objects in the system's attention.

A type of information that can be directly extracted from the spatiotemporal graph is the one of 'all known actions'. This information can be represented by a directional graph in which vertices represent unique classes of objects part in any previous action and edges represent simple actions that can involve the connected vertices (usually actions that a class of objects can perform on another class). E.g.: a hand can lift, move, grasp, release, etc. a cup.

We can observe that, in time, for a fixed set of classes of objects involved, if the actions vary enough, the graph of 'all known actions' will converge to the

graph of 'all possible actions' and the presented spatiotemporal history graph, will converge to the graph 'of all possible states' (The latter is something that should be avoided, because storing/remembering everything up to the smallest details is guaranteed to sooner or later cause time and memory issues).

Another type of information, that is obtained directly (e.g. tracking) or through reasoning, is that of an object occluding or containing other objects (totally or partially, but still unrecognizable by the detection level). To store this type of information, a relabeling of the class of the occluding object should be done i.e. a cup that has been found out to contain a ball should be labeled 'cup with ball inside'.

3 Contraction in Spatiotemporal Space

The idea here would be to contract in 3D (2D space + time) along 'the trajectory' of the movements. Every frame could be represented by a region adjacency graph. In order to stretch this into time, these region adjacency graphs (region adjacency combinatorial maps) should be matched to each other, i.e. the region adjacency graph at time t is matched with the one in $t+1$ and so on. In this sense we could define a 'trajectory' of each region This trajectory becomes a curve in 3D and with the techniques analogous with that of contraction of a 2D curve pyramid in [11], we can contract regions adjacent along this curve to produce the more abstract representation of the scene, e.g. where the movement started, where it ended etc (Figure 2).

If the analyzed scene has a structured background, then, depending on it's granularity, this is enough to detect movement using only topological information. On the other hand, this will increase the number of consecutive frames that differ with respect to topological relations. To reduce the abundance of topological states, to a set containing the most relevant ones, a set of adaptive pyramids is used. There are no constraints regarding the time intervals between 2 consecutive states. Actually, it is expected that in most of the cases where natural movement is present (not robots repeating some predefined action) these time intervals will differ quite a lot.

In subsections 3.1 and 3.2 we present two approaches, to the problem, which basically differ only in the order in which contraction in the spatial and temporal domains, is done. The first, avoids the difficult problem of graph matching by creating pyramids in the first step and then doing the matching using the pyramids. The second, while needing graph matching to be done, should have a lower memory usage. Moreover, in the ideal case, the resulting top level of the 2 approaches should be the same.

3.1 Spacial Contraction Followed by Temporal Contraction

For each frame, whose topological description is different from the one of the previous frame, a space-contraction pyramid is build, that preserves only the spatial information required by the higher functionality levels (i.e reasoning) and by the time-contraction. A space-contraction pyramid is a pyramid where

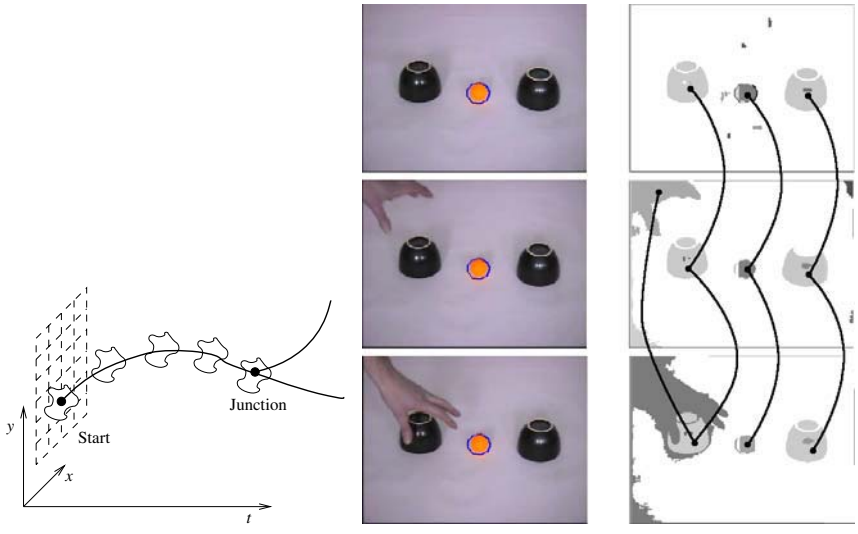


Fig. 2. a), b) Trajectory of movements

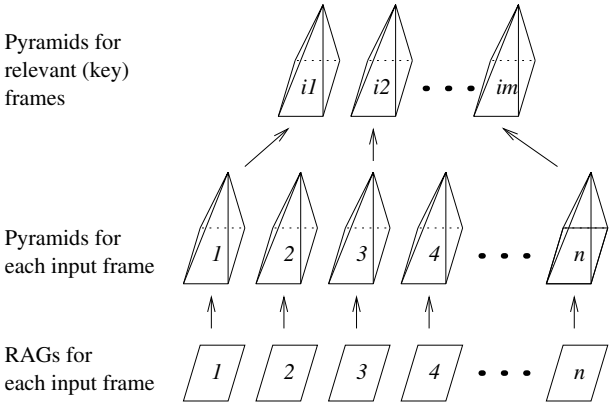


Fig. 3. Space time contraction

elements, from the same scene state, neighbored from a spatial point of view are contracted, and a time-contraction pyramid is a pyramid where elements, neighbored from a temporal point of view (consecutive scene states) are contracted.

To obtain the base level of the time-contraction pyramid from the set of space-contraction pyramids a matching step has to be performed (Figure 3). Each 2 consecutive pyramids (from a chronological perspective) have to be matched, and the vertices that represent the same object/visual entity should be linked by an edge (if it is possible i.e. if the same object/visual entity exists in both structures - existed in both frames). If a certain object/visual entity, that exists

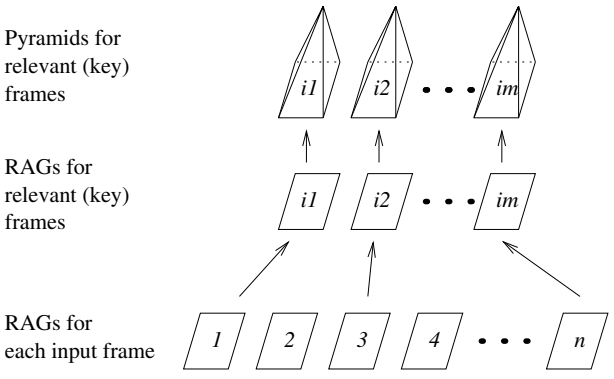


Fig. 4. Time space contraction

in one of the pyramids, does not exists in the other (occlusion, moved out of the field of view, etc.), no connecting edge can be created, thus obtaining a trajectory endpoint. If similar entities disappear and reappear at different time intervals, it will be the job of the reasoning part to decide whether it was the same instance of the same class or not.

The base level of the time-contraction pyramid contains a vertex for each of the frames in the source video, that differ in topological relations from the previous frame. Each vertex will contain the space-contraction pyramid for the region adjacency graph of the respective scene state. These vertices are linked together in a chronological manner i.e. each vertex is linked to the one of the previous and next frames. Also, as a result of the pyramid matching process mentioned before, the vertices from the consecutive space-contraction pyramids are linked together, showing the trajectories of the regions from the first through the last frame. For example: take the topological descriptions for each frame and represent them in a 3D space, where one of the dimensions is time, and the other 2 are used to represent the planar region adjacency graphs. If for every 2 consecutive graphs, the vertices representing the same object/visual entity are linked together by an edge, then following these inter-state connection edges will produce the regions trajectory in 3D space.

Each level of the time-contraction pyramid is a chronologically ordered list of space-contraction pyramids, each element describing the topological relations of a certain scene state. The space-contraction step reduces the spatial information in areas that are not of our interest. The purpose of the time-contraction pyramid is to skip the unnecessary frames caused by the presence of the structured background (which is needed for movement detection using only topological information).

3.2 Temporal Contraction Followed by Spatial Contraction

The base level of the time-contraction pyramid contains a vertex for each of the frames in the source video, that differ in topological relations from the previous

frame (Figure 4). Each of these vertices contains the region adjacency graph (RAG) for the respective frames. Through a preliminary process of matching, each vertex in a region adjacency graph should be connected with the vertex(vertices), from the two neighboring graphs, that represent the same object/visual entity (if it is possible i.e. if the same object/visual entity exists in the neighboring region adjacency graphs frame). In other words, the base level of the pyramid is the discretized evolution of the region adjacency graph of the presented scene with the exception that identical consecutive states are merged into a single state.

If we would represent the base level structure in a N dimensional space (3D for 2D state descriptions + time) we would see that we have obtained curves representing the trajectories of the different regions analyzed. A line segment parallel to the time axis, will denote a static region through the respective time interval. Each level of the pyramid is made out of a sequence of region adjacency graphs. Each vertex in a region adjacency graph should be connected with the vertex(vertices), from the two neighboring graphs, that represent the same object/visual entity.

With each new level added to the time contraction pyramid, the number of topological states decreases. After reducing the number of topological states, a contraction of topological information for each state can be considered (at this level the detail regarding the background should not be important any more).

There are 2 ways that can be considered for doing this:

- contract each state independently (create a pyramid for each of the topological states at the top level of the time-contraction pyramid)
- contract all the graphs together (allow contraction kernels to span along more than one state graph)

3.3 Spatiotemporal Entities

The trajectories of (moving) objects (visual entities resulted from segmentation and tracked through the whole time span) represent curves connecting start, end and the junction points. Junction points happen when two (or more) trajectories touch or cross each other, which we interpret as the 'interaction' of two objects.

Following the work of Kropatsch [11] the trajectory, which is a curve in 3D, and the cells, which are vertices of the graph, can be related as follows:

0-cell - an empty cell (no trajectory motion within the receptive field)
1-cell - the trajectory starts or ends in this cell (it leaves or enters the cell and intersects only *once* the boundary of the receptive field)
2-cell - the trajectory crosses the receptive field (it intersect *twice* the boundary of the receptive field).
*-cell - a cell where more than one trajectory meet, a junction cell (the boundaries of the receptive field are intersected *more than twice*).
1-edge - trajectory intersects the connected segment boundary of the receptive field.
0-edge - no trajectory intersect the boundary of the receptive field.

It is assumed that: 1) the cells are consistent, i.e. if a trajectory crosses a boundary both cells adjacent to this boundary are in correct classes, and 2) all trajectories are well distinguishable in the base, e.g. there are no more than one single curve in one single cell of the base (except at *-cells).

3.4 Selection of Contraction Kernels

Contraction should be done along the trajectory, like in curve pyramids in 2D [11, 5]. In order to undertake the contraction process, the contraction kernels must be selected. The selection rules are 1-cells and *-cells must always survive. *-cells are not allowed to have children. This prevents the area of unclear information[2] from growing. Branches of contraction kernels follow the trajectory if possible and are selected in following order: 1-cells, 2-cells, 0-cells. Receptive fields are merged as follows:

1. A 1-cell can merge with its adjacent 2-cells, then with any adjacent 0-cell and will become an 1-cell again;
2. a 2-cell can merge with both adjacent 2-cells or with any adjacent 0-cell and remains a 2-cell;
3. a 0-cell can merge with any adjacent cell and remains a 0 cell if it is merged with another 0-cell.

If the rules do not determine the contraction kernels the random selection methods [12, 10, 9] are applied. Applying these rules, the trajectory remains a simply connected curve in spatiotemporal space. At the top level (where no more contraction is possible) we find only 1-cells and *-cells giving on overview of all movements, when and where is started, when and where the cup was grasped, and this is compact for all types.

4 Example

A simple, human language like description of a scene with two cups and a yellow ball is shown in Table 1. Even though the frame numbers are given, they are only for orientation purposes and can be easily eliminated from the description by putting the adverbial for example 'next', 'after that', 'then' etc. The previous description would be represented in the following way (see Figure 5) in the resulting top level of both approaches. The initial configuration contains 3 objects: 2 cups and 1 ball. So we initialize the objects structure with the following: *cup(1)*, *ball* and *cup(2)*. (The numerical ids in parenthesis are present to distinguish the two cups, identification could be done in many other ways. Also in the same interest, vertices are numbered to identify different positions in time.) Vertex(0) in Figure 5 depicts the initial configuration. The next vertices and edges are as follows:

1. action *move*: creates object *hand* and adds *vertex(1)*;
2. action *grasp*: links to objects *hand* and *cup(1)* and adds *vertex(2)*

[2] trajectory may intersect or may be just close to each other.

Table 1. Scene description

cell type	Frame	Description	cell type	Frame	Description
0	16–21:	hand from left	*	91:	grasps the same cup again
*	22:	grasps left cup	*	87–90:	releases it and moves up and down
*	27–30:	moves it over ball	*	85–86:	moves it to the right (but left of the right cup)
*	31:	releases cup	*	84:	grasps it
*	32:	grasp same cup (again)	*	76–77:	moves to the right cup
*	33–36:	shifts it to the left	*	75:	releases it
*	37:	releases cup	*	71–74:	shifts it to the right (but still to the left of the right cup
*	38–40:	moves to right cup	*	70:	grasps it
*	41:	grasps right cup	*	67–69:	moves to the left (most) cup
*	42–58:	shifts right cup in front of left cup (hiding left cup Fr 46–54) to the left of the original cup	*	66:	releases it
*	58:	releases cup	*	63–65:	shifts it to the right
*	59–61:	moves to the other cup	*	62:	grasps it
			...	...	...

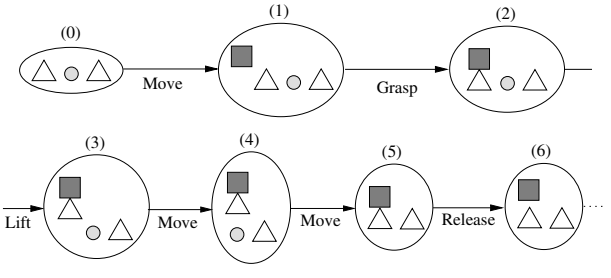


Fig. 5. Example history graph. □ Hand, ◯ Ball, △ Cup

3. action *lift*: links to objects *hand* and *cup(1)* and adds *vertex(3)*
4. action *move*: links to objects *hand* and *cup(1)* and adds *vertex(4)*
5. action *move*: links to objects *hand* and *cup(1)* and adds *vertex(5)*
6. action *release*: links to objects *hand* and *cup(1)* and adds *vertex(6)*

Although the presented approaches would work in a different way (one would first try to identify the important visual entities and then key events, while the other would start with the key events and then continue with key entities), the expected result is the same.

5 Conclusion

This paper presents a concept relating story-board description of video sequences with spatio-temporal hierarchies build by local contraction processes of spatio-temporal relations. Since object trajectories are connected curves we identify their ends and junctions and their connections as the high level descriptions. Junction points happen when two (or more) trajectories touch or cross each other, which we interpret as the 'interaction' of two objects. We propose to derive them similar to curve pyramid in 2D [11, 5], For the implementation we plan to use the concept of combinatorial pyramids in 3D [3, 4].

References

[1] J. Allen. An Interval-based Representation of Temporal Knowledge. In *Proc. 7th Inter. Joint Conf. on AI*, p:221–226, 1981.
[2] N. I. Balder. *Temporal Scene Analysis: Conceptual Descriptions of Object Movements*. PhD thesis, University of Toronto, Canada, 1975.
[3] L. Brun and W. G. Kropatsch. The Construction of Pyramids with Combinatorial Maps. Technical Report PRIP-TR-63, Pattern Recognition and Image Processing Group, TU Wien, Austria, 2000.
[4] L. Brun and W. G. Kropatsch. Introduction to Combinatorial Pyramids. In G. Bertrand, A. Imiya, and R. Klette, editors, *Digital and Image Geometry*, p:108–128. Springer, Berlin, Heidelberg, 2001.
[5] M. Burge and W. G. Kropatsch. A Minimal Line Property Preserving Representation of Line Images. *Comp., Devoted Issue on Im. Proc.*, 62:355–368, 1999.
[6] A. Chella, M. Frixione, and S. Gaglio. Understanding Dynamic Scenes. *Artificial Intelligence*, 123:89–132, 2000.
[7] B. Clarke. A Calculus of Individuals Based on Connection. *Notre Dame J. of Formal Logic*, 23(3):204–218, 1981.
[8] B. Clarke. Individuals and Points. *Notre Dame J. of For. Log.*, 26(1):61–75,1985.
[9] Y. Haxhimusa, R. Glantz, and W. G. Kropatsch. Constructing Stochastic Pyramids by MIDES - Maximal Independent Directed Edge Set. In E. Hancock and M. Vento, eds., *4th IAPR-TC15 Workshop on GbR in PR*, LNCS 2726:35–46, York, UK, 2003. Springer Verlag.
[10] Y. Haxhimusa, R. Glantz, M. Saib, G. Langs, and W. G. Kropatsch. Logarithmic Tapering Graph Pyramid. In L. van Gool, ed., *Proc. of 24th DAGM Symposium*, LNCS 2449:117–124, Swiss, 2002. Springer Verlag.
[11] W. G. Kropatsch. Property Preserving Hiearchical Graph Transformation. In C. Arricelli, L. Cordella, and G. Sanniti di Baja, editors, *Advances in Visual Form Analysis*, p:340–349, Singapore, 1998. World Scientific.
[12] P. Meer. Stochastic image pyramids. *Computer Vision, Graphics, and Image Processing*, 45(3):269–294, 1989.
[13] D. Randell, Z. Cui, and A. Cohn. A Spatial Logic Based on Regions and Connection. In *Proc. 3rd Int. Conf. on Knowledge Representation and Reasoning*, p:165–176. Morgan Kaufmann, 1992.
[14] R. Sablatnig. Increasing flexibility for automatic visual inspection: the general analysis graph. *Mach. Vision Appl.*, 12(4):158–169, 2000.
[15] D. Vernon, editor. *A Reasearch Roadmap of Cognitive Vision (DRAFT Version 3.2)*. ECVision: The European Research Network for Cognitive Computer Vision Systems, 2004.

Approximating the Problem, not the Solution: An Alternative View of Point Set Matching

Tibério S. Caetano and Terry Caelli

National ICT Australia, Canberra ACT 0200, Australia

Abstract. This work discusses the issue of approximation in point set matching problems. In general, one may have two classes of approximations when tackling a matching problem: a representational approximation, which involves a simplified and suboptimal modeling for the original problem, and algorithmic approximation, which consists in using suboptimal algorithms to infer the assignment. Matching techniques in general have relied on the second approach: to keep a complete model of the original problem and use suboptimal techniques to solve it. In this paper, we show how a technique based on using exact inference in simple graphical models, which is an instance of the first class, can significantly outperform instances of techniques from the second class. We give theoretical insights of why this happens, and experimentally compare our approach with the well-known Shapiro and Brady and Christmas et al. methods, which are exemplars of the second class. We perform experiments with synthetic and real-world data sets, which reveal a significant accuracy improvement of the proposed technique both under point position jitter and size increasing of the point sets. The main conclusion is that techniques based on optimal algorithms with appropriate suboptimal representations may lead to better results than their counterparts which consist in using optimal representations, but approximate algorithms.

1 Introduction

The point set matching problem consists in finding correspondences between two point sets, which may be one-to-one or also many-to-one [1, 2], and arises in a variety of real world vision tasks such as stereo vision, registration, model-based object recognition and the like. In any real vision problem we are faced with *inexact* point set matching, or matching under structural corruption, which is known to be an NP-hard problem [3]. As a result, one must rely on approximate techniques do derive the "best" assignment in some suboptimal sense. Several approaches for solving matching problems, and in particular the inexact point set matching problem, have been proposed along the years [4]. Major representatives are *spectral* methods [5, 6] and *relaxation labeling* methods [7, 8]. These families of techniques consist in encoding all the available information into some complete representation of the problem and subsequently using approximate algorithms to derive the assignment. Several limitations have been reported with respect

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 233–242, 2005.

to spectral methods when structural corruption is present and with respect to relaxation methods when matching large point sets [1, 6, 9, 10].

This paper shows how a shift in point of view can lead to an alternative method that overcomes many of the limitations existent in some spectral and relaxation methods. Essentially, instead of using an optimal representation and an approximate algorithm, we do the opposite. We model the structure of points by a sparse representation that deliberately disregards a particular set of relational information that is actually available. In other words, we *approximate the problem*. The reason for that becomes clear in the next step: by taking advantage of this sparsity, we are able to apply an *optimal algorithm* to derive the best assignment. As a result, we advocate in favor of approximating the problem instead of the algorithm for solution.

For performance evaluation, we conducted experiments with synthetic and real world data sets, where we compared the proposed approach with traditional versions of spectral and relaxation methods, namely the Shapiro and Brady spectral method [5] and the Christmas et al. relaxation method [8]. Results indicate that the accuracy of the results obtained with the proposed technique significantly exceeds that obtained by the alternative techniques, either under structural corruption by point position jitter or under augmentation of the point set sizes.

2 Problem Definition

We consider the problem in, $\mathbb{R}^2$, of finding the subset of an S-sized point set (the *codomain pattern*) that best matches another point set (the *domain pattern*) having T points, where $T \leq S$. There may or may not exist distortions due to noise, but if there are, we assume no prior knowledge of the type of noise. We restrict the matching to be invariant up to isometries, so we do not consider scaling. The only constraint enforced in the mapping is that it must be a total function: every point in the domain pattern must map to one point in the codomain pattern (but the opposite may not hold).

3 The Model

The basic idea of the modeling strategy is to consider an undirected probabilistic graphical model (a Markov random field) where the nodes are points in the domain pattern and their possible realizations are points in the codomain pattern. In order to fully specify a graphical model, it is necessary to define (i) the potential functions and (ii) the connectivity of the model [11]. We start by formally specifying the model and introducing the potential functions.

3.1 Potential Functions

The cardinalities of the domain and codomain pattern sets are denoted, respectively, by T and S. Each point d_i in the domain is associated with a vertex of

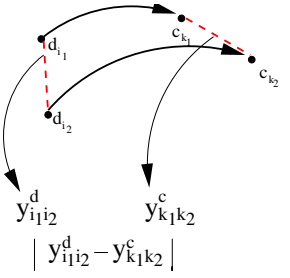


Fig. 1. An example of a pairwise mapping. An appropriate potential function should penalize more severely mappings for which $|y^d_{i_1 i_2} - y^c_{k_1 k_2}|$ is higher

Fig. 2. The kernel structure of the graphical model

a graph G_d, and each point c_k in the codomain is associated with a vertex of a graph G_c. The relative distance between a pair $\{d_{i_1}, d_{i_2}\}$ of points in the domain pattern is denoted as $y^d_{i_1 i_2}$. Analogously for the codomain pattern, we have that $y^c_{k_1 k_2}$ is the distance between points c_{k_1} and c_{k_2}. These distances are seen as *edge weights*. In this formulation, point pattern matching turns out to be a weighted graph matching problem.

The model formulation consists, initially, in defining each of the T vertices in G_d as a random variable that can assume S possible values (discrete states), corresponding to the vertices in G_c. Note that in this formulation the solution to the problem (the best match) corresponds to finding the most likely (the best) realization of the set of random variables.

Figure (1) illustrates a pairwise map and a possible measure which is relevant in order to construct the potential functions ($|y^d_{i_1 i_2} - y^c_{k_1 k_2}|$).

Since each node in the domain graph can map to S different nodes in the codomain graph, each pair of nodes can map to S^2 different pairs in the codomain graph. Figure (2) illustrates the kernel structure of our model: a pairwise clique, where each random variable represents a point in the domain graph which in turn can assume a set of S possible realizations (which themselves correspond to points in the codomain graph).

The sample space for this clique has S^2 elements, corresponding to all possible combinations that a pair of points in the domain graph can map to in the codomain graph. A potential function is a function that associates to each element of the sample space a positive real number. In our case, the only requirement that the potential function must obey is that its value must be as higher as more similar are the distances of the mapped edges, as illustrated in Figure (1).

Formally, we can specify the potential function by

$$\psi_{ij;kl} = p(X_i = x_k | X_j = x_l), \tag{1}$$

or, in matrix form, for each pair $\{X_i, X_j\}$ in G_d, we define

$$\psi_{ij} = \psi_{ij}(X_i, X_j) = \frac{1}{Z} \begin{pmatrix} \mathcal{S}(y^d_{ij}, y^c_{11}) & \dots & \mathcal{S}(y^d_{ij}, y^c_{1S}) \\ \vdots & \ddots & \vdots \\ \mathcal{S}(y^d_{ij}, y^c_{S1}) & \dots & \mathcal{S}(y^d_{ij}, y^c_{SS}) \end{pmatrix}, \tag{2}$$

where y^a_{bc} denotes the edge weight between vertices with indexes b and c in graph G_a. Z is a normalization constant that equals the sum of all elements in the matrix, in order to keep ψ_{ij} compatible with a probability distribution. $\mathcal{S}$ is a similarity function that measures the compatibility of the two arguments. We use here the Gaussian function,

$$\mathcal{S}(y^d_{ij}, y^c_{kl}) = \exp\left(-\frac{1}{2\sigma^2}|y^d_{ij} - y^c_{kl}|^2\right). \tag{3}$$

This "proximity measure" is needed in order to model the uncertainty due to the presence of noise. Obviously, its maximal value must be reached when there is no noise ($y^d_{ij} = y^c_{kl}$).

Having specified the potential functions, it remains to be determined the connectivity of the graphical model: which nodes will be neighbors in the model?

3.2 Connectivity

We have derived elsewhere [2, 4] a particular connectivity of the graphical model (a sparse graph) which has a very unique property: it is provably *optimal* in the particular case of exact matching. By optimal we mean the fact that the optimization problems over this particular sparse graph and over the fully connected graph are one and the same. This is important, because the optimization problem in the sparse graph can be solved in polynomial time, whereas the one over the complete model is NP-complete [4]. This equivalence holds strictly only for exact matching, and it is still an open issue to determine theoretically how close to optimal is the solution for inexact matching problems. Nevertheless, there is experimental evidence that for small to moderate noise the results are impressively good [4].

Here we use this particular subgraph as the connectivity pattern for the Markov random field under consideration. The graph topology is depicted in Figure (3).

The resulting graph is technically a 3-tree [12], which is a graph that arises by adding new nodes that are connected to precisely 3 existent nodes (these 3 nodes must form a clique, i.e. every pair must be connected). In the example in Figure (3), X_1, X_2 and X_3 are the 3 "reference" nodes to which the additional $T - 3$ nodes are connected.

Given the graph connectivity and the pairwise potential functions, the model is defined. The last step then consists in inferring what is the most likely joint

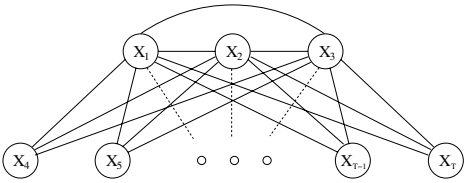


Fig. 3. A 3-tree graphical model. Each of the T nodes is a random variable representing a point in the domain graph. Each variable can assume S possible realizations, corresponding to points in the codomain graph

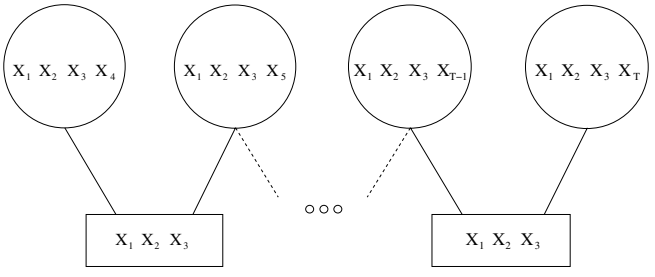


Fig. 4. The Junction Tree obtained from the model in Figure (3)

realization of all the random variables for the given connectivity and set of potentials of the model. This is precisely the MAP inference problem in this model, whose solution represents the best assignment in the point set matching task and is described in what follows.

3.3 MAP Computation

The Junction Tree framework provides a set of deterministic algorithms for *exact* inference in arbitrary graphical models [11, 13]. Here we use an algorithm from this framework in order to find the optimal MAP estimate for the model in Figure (3). A Junction Tree of a graph is another graph where (i) the nodes correspond to the maximal cliques of the former graph (a maximal clique is a clique which is not a proper subset of another clique) and (ii) the *running intersection property* is satisfied. This property states that all the nodes in the path between any two nodes in the Junction Tree must contain the intersection of these two nodes. It is known that the condition for the existence of a Junction Tree is that the graph must be *chordal*, or *triangulated* [13]. A chordal graph is a graph with no chordless cycles[1]. The 3-tree is a chordal graph, and this allows us to use the Junction Tree framework to calculate the MAP estimate of the random variables in the model.

[1] A chord in a cycle is an edge between two non-consecutive nodes in the cycle.

Figure (4) shows a Junction Tree obtained from the model in Figure (3). The nodes of the Junction Tree are denoted by circles in which are listed the nodes of the original graph that correspond to the respective maximal cliques. The rectangles are the so-called *separators*, that contain the intersection of the nodes to which they are linked. Both the nodes and the separators are endowed with "clique potentials", and the optimization process consists in updating these potentials, as explained below.

In this paper we applied the Hugin algorithm [13], an instance of the Junction Tree framework, to accomplish exact inference in the 3-tree model shown in Figure (3). The complexity of Junction Tree using Hugin in our 3-tree model is $O(S^4T)$. As a result, the complexity on S and T is polynomial. The Hugin algorithm essentially works in two steps: initialization and message-passing. During the initialization, the clique potential of each separator (Φ) is set to unity and the clique potential of each node (Ψ) is introduced (see subsection 3.1). These last clique potentials are assembled as an element-by-element product of the pairwise potentials (see Eq. 2) in the respective clique. For example, for the 3-tree model, $\Psi(x_i, x_j, x_k, x_l) = \psi(x_i, x_j)\psi(x_i, x_k)\psi(x_i, x_l)$.

The second step is the message-passing scheme, which involves a transfer of information between two nodes V and W in a systematic way until every pair of nodes in the Junction Tree has participated in the process [11]. This operation is defined by the following equations:

$$\Phi_S^* = \max_{V \setminus S} \Psi_V \qquad\qquad \Psi_W^* = \frac{\Phi_S^*}{\Phi_S} \Psi_W$$

where we used standard notation for the current and updated (*) versions of the separator potentials (Φ) and the clique potentials (Ψ). The first equation is a maximization over all sub-configurations in Ψ_V that do not involve the singleton nodes which are common to Φ_S and Ψ_V. The second is simply a normalization step necessary to keep Ψ_W consistent with the updated version of Φ_S (division and multiplication are performed element-by-element). The above potential update rules must respect the following protocol: a node V can only send a message to a node W when it has already received messages from all its other neighbors. If this protocol is respected and the equations are applied until all clique nodes have been updated, the algorithm assures that the resulting potential in each node and separator of the Junction Tree is proportional to the (global) maximum a posteriori probability distribution of the set of enclosed singleton nodes [11]. The constant of proportionality is guaranteed to be the same for every node, what implies that the mode of the local potentials will correspond to the MAP estimate. In our particular case, we need the maximum probability for each singleton, what can be obtained by maximizing out the remaining 3 singletons in each of the nodes. The indexes for which the final potentials are maximum are considered the vertices in G_c to which the corresponding vertices in G_d must be assigned.

4 Experiments and Results

We have carried out two sets of experiments, one with synthetic point sets and another with real-world data. In both of them, we compare our technique (denoted simply as JT) with the probabilistic relaxation labeling version described in [8], denoted as PRL, as well as with the spectral method of Shapiro and Brady [5], denoted as SB. These two methods encode all the pairwise distances in their model representation, whereas our method only encodes those distances that correspond to the 3-tree topology. On the other hand, our approach uses a non-iterative algorithm which is optimal, whereas the other two are based on approximate and heuristic algorithms. Results show how these different approximation principles affect accuracy in point set matching.

4.1 Synthetic Data

In the experiments with synthetic data, we generated random points according to a bivariate uniform distribution in the interval $x = [0, 1]$, $y = [0, 1]$. We carried out two experiments: one with graphs of equal sizes and another with graphs with different sizes (the SB method is not suited for different graph sizes). In the first experiment, with equal sized graphs, we used graphs of sizes (10,10), (20,20) and (30,30) nodes. Then, for each of these 3 instances, we perturbed the codomain pattern with progressive levels of noise in the position of the nodes (white Gaussian noise with varying standard deviation). This setting allows us to have an idea of the relative performances both under the augmentation of the graph sizes and under progressive structural corruption of the patterns. Figure (5)-left shows the obtained curves under these experimental conditions.

The graphs show that differential noise increasing has a very different impact on SB when compared to JT and PRL. JT and PRL performances are much

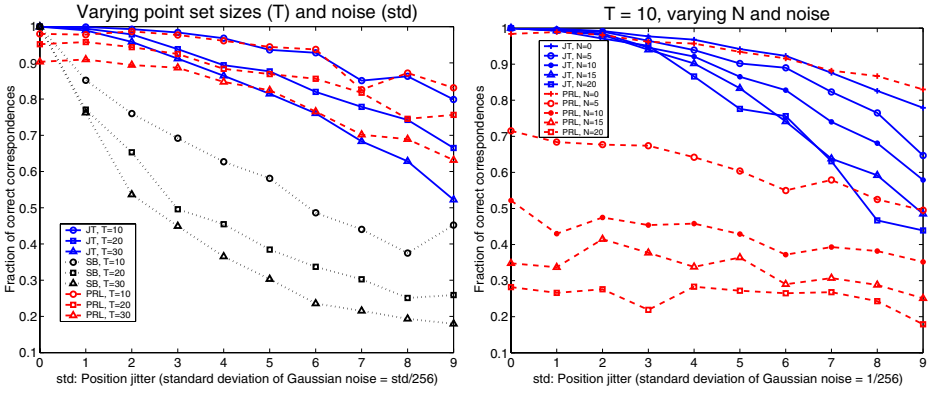


Fig. 5. Left: performances of JT, PRL and SB when the noise (position jitter) increases, for various sizes of the graphs ($T = S$ in this experiment). Right: Performances of JT and PRL when the noise (position jitter) increases, for fixed size of the domain graph (T) and various sizes of the codomain graph ($T + N = S$)

less affected than that of SB for a same amount of incremental noise. Also, it is possible to note that scaling up the sizes of the graphs, for small levels of noise ($std = 0 - 1$), practically does not affect the performance of JT, whereas the performances of PRL and SB are significantly affected. This may suggest that the proposed technique may be a serious alternative to these other approaches in circumstances where the noise involved is not high. For larger, but still moderate amount of noise ($std = 1 - 4$), the proposed technique still dominates the others. It is clear however that PRL has similar or possibly superior performance for extremely high levels of noise. This is probably due to the fact that under severe noise the 3-tree approximation becomes poor. However, note that severe stochastic perturbation is not a common issue in point set matching problems. Usually, in real applications like stereo matching, shape matching or registration, the point sets differ essentially by some isometric, affine or projective transformation (possibly together with a non-linear deformation), but the stochastic perturbation in the position of the points is itself small or at most moderate [4].

In the second experiment with synthetic point sets, we kept constant the size of the domain point set (10 nodes) and varied the size of the codomain point set (from 10 to 30 nodes in steps of 5). The introduction of noise by position jitter was analogous to the equal-size graphs experiment described above. In this experiment, we only compared JT with PRL, since SB is not suited for graphs with different sizes. Figure (5)-right shows the graphs corresponding to this experiment. In this experiment, it is clear that PRL does not perform well when the sizes of the point sets are significantly different. It is also possible to notice the fact that for larger graphs the proposed technique has a very superior performance for small levels of noise. A glimpse of this fact was available in the previous experiment, but it is much more evident in this new one.

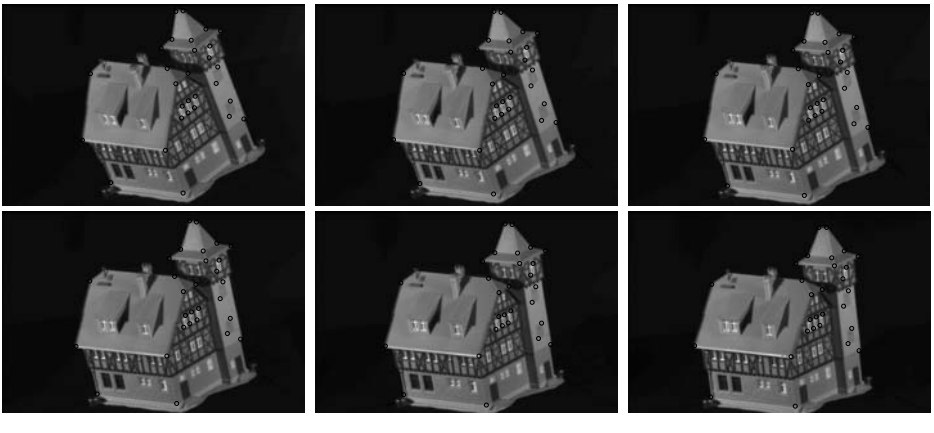


Fig. 6. Images from the CMU house data set (from left to right and top to bottom: images 1, 11, 21, 31, 41 and 51)

Table 1. Matching results (correct correspondences out of 29)

Image pair	1 - 11	11 - 21	21 - 31	31 - 41	41 - 51	1 - 21	11 - 31	21 - 41	31 - 51
JT	29	29	29	29	29	28	28	29	29
PRL	29	29	29	29	29	28	28	28	28
SB	19	16	18	23	23	17	16	15	14

Image pair	1 - 31	11 - 41	21 - 51	1 - 41	11 - 51	1 - 51
JT	28	28	29	27	25	25
PRL	28	26	27	26	26	25
SB	21	14	14	11	13	7

4.2 Real-World Data

In the real-world experiments, we performed comparisons of the algorithms using the CMU house sequence, as done in [1]. Figure (6) shows the images used in the experiments.

A total of 6 images were used in the experiments, as shown in Figure (6). In total, 29 landmark points were manually marked in each of the images. Then we run the three algorithms (JT, PRL and SB) in several pairs of them, in a systematic way, described as follows. First we matched pairs that are consecutive in Figure (6) (1-11, 11-21, 21-31, 31-41, 41-51). Then we matched pairs separated by a single image (1-21, 11-31, 21-41, 31-51), by two images (1-31, 11-41, 21-51), by three images (1-41, 11-51) and finally by four images (1-51). For each of the experiments, the amount of correct correspondences for each technique was recorded and is shown in Table 1.

5 Discussion

A general understanding of the results is possible if we pay attention to a few key observations. PRL is an heuristic iterative optimization procedure that for problems with big search spaces may not converge in a feasible amount of iterations (even when it does, it reaches only a local optimum). In fact, in all experiments we have used 200 iterations for PRL, what is already considered to be a very large number compared to its use in many applications [8]. The SB method is non-iterative and also effective in the noiseless case, but it is clear from the experiments that the eigen-structure of the point proximity matrix does not provide robust features for matching under (even very small) noise. On the other hand, the proposed technique is non-iterative and always finds the optimal solution for a model that itself is optimal in the limit of zero noise. Thus it is reasonable to expect that for small levels of noise the solution will be "close" to the optimal solution (although the precise meaning of "close" is not yet clear, as already mentioned), what is strongly suggested by the virtually perfect performance in the range $std = [0, 1]$. Current efforts are being dedicated to obtain theoretical results on how the 3-tree approximation deteriorates as noise increases.

6 Conclusion

In this work, we have investigated how different sources of approximation affects the performance of point set matching methods. Usual approaches to point set matching, such as spectral and relaxation-based methods, encode all the available information in the model representation, but rely on approximate algorithms for deriving the assignment. Our method, in contrast, consists in approximating the problem itself such that the resulting representation is suitable for the use of optimal algorithms for finding the match. Our method consists in modeling the relational features of a point set in a Markov random field framework where the underlying graph structure is sparse and allows for optimal MAP computation in polynomial time. Experiments were performed both with synthetic and real-world data sets, which indicate that the proposed "approximate model - optimal algorithm" approach is a serious alternative to other "optimal model - approximate algorithm" approaches.

References

1. Wang, H., Hancock, E. R.: A kernel view of spectral point pattern matching. Workshop on S+SSPR (2004). LNCS 3138, 361–369
2. Caetano, T. S., Caelli, T., Barone, D. A. C.: An optimal probabilistic graphical model for point set matching. Workshop on S+SSPR (2004). LNCS 3138, 162–170
3. Akutsu, T., Kanaya, K., Ohyama, A., Fujiyama, A.: Point matching under non-uniform distortions. Discrete Applied Mathematics **127** (2003) 5–21
4. Caetano, T. S.: Graphical models and point set matching. PhD. Thesis, Universidade Federal do Rio Grande do Sul, Brazil (2004)
5. Shapiro, L., Brady, J. M.: Feature-based Correspondence - An Eigenvector Approach. Image and Vision Computing **10** (1992) 283–288
6. Carcassoni, M., Hancock, E. R.: Spectral correspondence for point pattern matching. Pattern Recognition **36** (2003) 193–204
7. Rosenfeld, A., Kak, A. C.: Digital Picture Processing, Vol. 1. Academic Press, New York, NY (1982)
8. Christmas, W. J., Kittler, J., Petrou, M.: Structural Matching in Computer Vision Using Probabilistic Relaxation. IEEE Trans. PAMI **17** n. 8 (1995) 749–764
9. Caelli, T., Kosinov, S.: An eigenspace projection clustering method for inexact graph matching. IEEE Trans. PAMI **26** n. 4 (2004) 515–519
10. Gold, S., Rangarajan, A.: A Graduated Assignment Algorithm for Graph Matching. IEEE Trans. PAMI **18** n. 4 (1996) 377–388
11. Jordan, M. I.: An Introduction to Probabilistic Graphical Models. In preparation
12. West, D. B.: Introduction to Graph Theory, 2nd Edition. Upper Saddle River, NJ. Prentice Hall (2001)
13. Lauritzen, S. L.: Graphical Models. Oxford University Press, New York, NY, (1996)

Defining Consistency to Detect Change Using Inexact Graph Matching

Sidharta Gautama, Werner Goeman, and Johan D'Haeyer

TELIN, Ghent University, St.Pietersnieuwstraat 41, B-9000 Gent, Belgium
sidharta.gautama@ugent.be

Abstract. In this paper, we discuss the notion of consistency in inexact graph matching to be able to correctly determine the optimal solution of the matching problem. Consistency allows us to study the cost function which controls the graph matching process, regardless of the optimization technique that is used. The analysis is performed in the context of change detection in geospatial information. A condition based on the expected graph error is presented which allows to determine the bounds of error tolerance and in this way characterizes acceptable over inacceptable data inconsistencies.

1 Introduction

Graphs are a powerful data structure to represent objects and concepts in various domains. In geographic information systems (GIS), attributed graphs form a natural way to represent spatial objects together with its features and relationships to other objects. Within this field, a major challenge is the continuous assessment and control of the quality of the spatial data. The rapid growing number of sources of geospatial data, ranging from high-resolution satellite and airborne sensors, GPS, and derivative geospatial products, pose severe problems for integrating data. Content providers face the problem of continuously ensuring that the information they produce is reliable, accurate and up-to-date. Integrity constraints are able to resolve certain issues in the data, like valid attribute values or relationships between data objects. The main issue is however the consistency of the data with respect to the current "real-world" situation. Part of this problem is handled using image interpretation from aerial photos and very-high-resolution satellite images. Although this is still mainly a manual process performed by human operators, automated detection of change and anomalies in the existing databases using image information can form an essential tool to support quality control and maintenance of spatial information.

In our work, graph matching is used to find correspondences between the detected image information and the geospatial vector data, like digital road maps. The query process, based on attributed graph matching, is driven by the spatial relations between the features and takes into account different errors that can occur (e.g. spatial inaccuracy, data inconsistencies between image and vector data). Error-tolerant graph matching can be used to find correspondences between the detected image information and the vector data. Spatial constraints

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 243–252, 2005.

between objects are used to find a reliable object-to-object mapping. Spatial relations between objects prove to be more reliable for detecting change compared to local object features which cannot always be detected with high enough reliability. We derive an expression, based on the notion of consistency as introduced in [4], which characterizes the bounds where an image feature is identified as part of the object model or as a noise structure. This condition which maps a feature on the null label is a difficult constraint to model and has been traditionally set using heuristic rules-of-thumb. We show how the expected graph error of the object model can be used to determine this constraint.

The remainder of this paper is organized as follows. Section 2 introduces error-tolerant graph matching and derives the error bound to characterize acceptable over inacceptable inconsistencies. Section 3 gives experimental results on synthetic data which validate the derived bounds. Section 4 concludes the paper.

2 Error-Tolerant Graph Matching

The problem can be represented as finding the correspondence between two sets of features: one set originating from the geographic database and one set originating from the image. Given these features an abstract representation can be built as an attributed graph. The vertices of the graph represent image features and the vertex attributes can contain measurements on these features. The edges of the graph represent relations between features and the edge attributes can contain measurements on spatial relations. A similar graph can be built on the vector data, using data objects as vertices and relations between objects as edges. The problem of registration is represented as a graph matching problem, which seeks the correspondence of similar vertices between two attributed graphs.

In solving the correspondence problem, one should allow tolerance to imprecision and inconsistencies. Errors can occur on the location of the object due to inaccurate detection, differences in spatial resolution of the data and data inconsistency. In addition, false positives can be present in both datasets. Specific to the problem is that the matching technique should be able to distinguish between errors due to severe data inconsistencies and errors due to spatial inaccuracy. As an example of the latter category, the position of the roads can differ but the general structure of the road network is preserved. In the first category, the structure of the road network differs significantly. In this work, we examine continuous relaxation labeling to solve the correspondence problem ([2],[7]).

2.1 Constraint Satisfaction Using Relaxation Labeling

The matching problem can be defined as a graph labeling problem, which consists out of the following elements:

1. a set of objects $i \in \Omega_i$, corresponding to image features;
2. a set of labels $\lambda \in \Omega_\lambda$, corresponding to GIS features;
3. a neighbour relationship over the objects;
4. constraints on possible labels between pairs of neighbouring objects.

Relaxation labeling techniques use an iterative process to determine the probabilities of each object. Different update rules have been proposed. In [4], the relation between different update rules is analytically shown. The problem of finding consistent solutions is shown to be equivalent to solving a variational inequality which is based on the mathematical concept of "consistency". This concept, which is defined below, is interesting because by using it, the labeling process can be redefined as a quadratic optimization process. This offers guidance in determining good compatibility coefficients.

To each object i a probability distribution $\{p_i(\lambda)\}_{\lambda \in \Omega_\lambda}$ is associated that expresses that object i has label λ:

$$0 \leq p_i(\lambda) \leq 1, \quad \sum_{\lambda \in \Omega_\lambda} p_i(\lambda) = 1 \tag{1}$$

A labeling for the problem is specified by $\boldsymbol{p} = \{p_i(\lambda)\}_{i \in \Omega_i, \lambda \in \Omega_\lambda}$. For each pair of neighbouring objects i and j and for each pair of labels λ and λ', a compatibility coefficient $r_{ij}(\lambda, \lambda')$ is defined. These coefficients express the compatibility of assigning label λ to object i in combination with assigning label λ' to object j. Negative values express incompatibility, positive values compatibility. Given these quantities, the support of a label λ for the object i given by the correspondence $\boldsymbol{p}$ is defined as

$$s_i(\lambda) = s_i(\lambda, \boldsymbol{p}) = \sum_{j \in \Omega_i} \sum_{\lambda' \in \Omega_\lambda} r_{ij}(\lambda, \lambda') p_j(\lambda') \tag{2}$$

Given a non-ambiguous solution $\boldsymbol{p}$ (i.e. $p_i(\lambda) = 0$ or 1), with $\lambda_1, ..., \lambda_n$ the labels which are given to the resp. object i,...n, then $\boldsymbol{p}$ is a consistent solution iff

$$s_i(\lambda_i, \boldsymbol{p}) \geq s_i(\lambda, \boldsymbol{p}), \quad \forall \lambda, \quad i = 1...n \tag{3}$$

For a non-ambiguous solution $\boldsymbol{p}$, this can be extended to the weighted sum of the support functions. $\boldsymbol{p}$ is a consistent solution iff

$$\sum_{\lambda \in \Omega_\lambda} p_i(\lambda) s_i(\lambda, \boldsymbol{p}) \geq \sum_{\lambda \in \Omega_\lambda} v_i(\lambda) s_i(\lambda, \boldsymbol{p}), \quad i = 1...n \tag{4}$$

for all labelings $\boldsymbol{v}$.

Eq. 4 defines the solution $\boldsymbol{p}$ through a system of n inequalities. Hummel and Zucker have shown that if the compatibility matrix $r_{ij}(\lambda, \lambda')$ is symmetric, the solution can be calculated as maximizing the average local consistency, given by

$$A(\boldsymbol{p}) = \sum_{i \in \Omega_i} \sum_{\lambda \in \Omega_\lambda} p_i(\lambda) s_i(\lambda, \boldsymbol{p}) \tag{5}$$

This is a quadratic function in the variables $p_i(\lambda)$, which can be optimized using a constrained gradient descent method taking into account the restrictions of Eq.(1).

This quadratic function can be found in various forms in the literature. In graph theory, the problem is known as the maximum clique problem [1]. The

optimal consistent correspondence is equivalent with the maximum clique in the association graph between two graphs. Based on a theory of Motzkin and Strauss, it has been shown how the global optima of a quadratic problem based on the adjacency matrix of a graph are equivalent to the maximum cliques [6]. The quadratic problem is similar to Eq. 5 where the compatibility matrix equals the adjacency matrix.

In computer vision, the graph matching problem has been studied using different techniques. In [2] a Bayesian model is developed which leads to a probabilistic relaxation scheme. In [4] it is shown how the updating equation of probabilistic relaxation can be approximated by continuous relaxation. The underlying model is thus similar to Eq. 5, with the compatibility coefficients related to the conditional probabilities $p_{ij}(A_{ij}|\lambda, \lambda')$. In [5] a Markov random field approach is introduced to solve the matching problem using a relational description of the scene. The problem is modeled using normal distributions, which makes the modeling similar to the one used by [2]. The essential difference lies in the optimization technique which is applied. In the case of MRF, the optimal solution is found using annealing techniques. Probabilistic relaxation uses the relaxation equations, which are a form of gradient descent. After review of the literature, we find that the published techniques differ mostly in the optimization technique that is applied (i.e. search trees, annealing, genetic algorithms). The modeling of the matching problem is often similar albeit in disguised form. In our view, it is important to concentrate on the correct modeling of a problem. If a problem is badly modeled, it will always lead to a bad solution, regardless of the optimization technique that is used.

2.2 Parameter Condition

To guarantee a good solution of the matching problem, the compatibility matrix $r_{ij}(\lambda, \lambda')$ needs to be determined correctly. In most applications, the value of these coefficients are determined using heuristics which basically impose a relative order on the constraints. Strong constraints receive a higher absolute value then weak constraints. The specific ratio between the constraints is usually determined through trial-and-error. For the null assignment it is however difficult to determine a correct value for the compatibility coefficient $r_{ij}(\lambda_\emptyset, \lambda')$. Since each object is a priori a possible null object, every assignment is consistent with the null assignment. The problem is to assess the relative importance of the null assignment with respect to the other constraints. It should be avoided that the null solution is the most consistent solution of the system. On the other hand, false correspondences of spurious points should be less consistent than the null assignment.

The definition of consistency can be used to determine the correct values. The definition not only determines the optimal solution of the labeling problem, it also determines what values the compatibility coefficients should take for an "ideal" solution to become the optimal solution of the system. The ideal solution is the matching we wish to find given the noise properties of the detection. For a correct null assignment, we need to determine when the errors, which occur

in the neighbour structure of a node, are acceptable and when the number of errors becomes too large so that the null label should be assigned. To analyse this, we should look at the support of the different assignments. In the case of the null assignment, the support can be written as:

$$\begin{aligned} s_i(\lambda_\emptyset, \boldsymbol{p}) &= \sum_{j \in \Omega_i} \sum_{\lambda' \in \Omega_\lambda} r_{ij}(\lambda_\emptyset, \lambda') p_j(\lambda') \\ &= w_\emptyset \sum_{j \in \Omega_i} \sum_{\lambda' \in \Omega_\lambda} p_j(\lambda') \\ &= w_\emptyset d(i) \end{aligned} \tag{6}$$

with $d(i)$ the degree of node i (i.e. the number of neighbours). We have simplified $r_{ij}(\lambda_\emptyset, \lambda') = w_\emptyset$ if $j \in \Omega_i$ (else $r_{ij}(\lambda_\emptyset, \lambda') = 0$). The constant factor $w_\emptyset$ is reasonable in the absence of prior knowledge of assignments.

The support for a non-null label can be split up into three classes Ω_i^+, Ω_i^- and Ω_i^0, namely positive coefficients which express compatibility, negative coefficients which express incompatibility and negative coefficients which control the null assignment. If we consider the first two classes of coefficients constant (resp. w_+ and w_-) within the neighbourhood of node i then the support for λ_i can be simplified to

$$\begin{aligned} s_i(\lambda_i) &= \sum_{j \in \Omega_i^+} r_{ij}^+(\lambda_i, \lambda_j) + \sum_{j \in \Omega_i^-} r_{ij}^-(\lambda_i, \lambda_j) + \sum_{j \in \Omega_i^0} r_i^0 \\ &= w_+ n_+ + w_- n_- + w_\emptyset n_0 \end{aligned} \tag{7}$$

Here n_+ is the number of compatible neighbours, n_- the number of incompatible neighbours and n_0 the number of null-neighbours, with $n_+ + n_- + n_0 = d(i)$. Eq.(6) and (7) give the following condition which holds in the optimal solution:

$$w_+ n_+ + w_- n_- + w_\emptyset n_0 > w_\emptyset d(i) \tag{8}$$

or equivalently

$$(1 - f^0) w_\emptyset < f^+ w_+ + f^- w_- \tag{9}$$

where f^+, f^- and $f^\emptyset$ are the fraction of compatible, incompatible and null assignments in the neighbourhood of object i for the ideal mapping.

Eq. 9 can be used to determine the weights for the compatibility matrix given the expected relational graph error. It allows to make a distinction between points showing small distortions, which should find a correspondent in the other dataset, and points showing severe distortions, which should be assigned the null label. As previous research usually relied on rules-of-thumb to determine these weights (e.g. [7]), the importance of this equation is that it allows precise definition of the weights of the graph matching problem with respect to the expected graph error of the system.

3 Experimental Results

A set of experiments has been performed on images containing randomly scattered points. Each image is generated twice: one copy which serves as a reference and one copy which contains perturbations on the scattered points (e.g. noise on the position, added spurious points). The aim is to find the corresponding points between the two copies using graph matching while ignoring the spurious points in the data. The experiment is an abstraction of the correspondence problem between image and GIS data after features like road junctions have been detected in the image.

To apply the technique to matching sets of points, we need to introduce the constraints which define similarity. For road junctions several possibilities exist. However, the quality of detection of road junctions that can be achieved is not of sufficient quality to use object features, like number of incoming roads, as information for the correspondence process [3]. Fragmentation and false detections can frequently occur in the detected road network and are difficult to control. We therefore opt to use geometric invariants between subsets of corresponding junctions. The most simple constraints are binary relations like geometric relations (e.g. angle, distance) between a junction and its neighbours to find correspondences. These are much more stable features, given the detection quality which can realistically be expected from road detection. In these experiments we rely only on the relative angle between pairs of points. Figure 1 shows an illustration of this constraint. The black points show object points i and the grey points show the label points λ. In mapping a pair of points i and j on λ and λ' the relative angle between the lines ij and $\lambda\lambda'$ does not exceed a given $\triangle\alpha$. (e.g. $\pi/4$). If this constraint is violated, the compatibility coefficient $r_{ij}(\lambda, \lambda')$ is assigned a negative weight w_-. The graph representation of the data is of course not restricted to angles and can be readily generalized to incorporate other measurements like connectivity, distance or other topological relations. In our case, the angle between junctions was chosen because it could be reliably measured in the image. Other measurements like connectivity between junctions are more difficult to measure in the image due to the degree of fragmentation in road

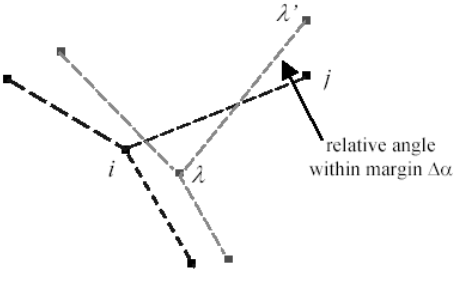


Fig. 1. Illustration of the spatial constraint, based on consistent relative angle between pairs of correspondences (i, λ) and (j, λ')

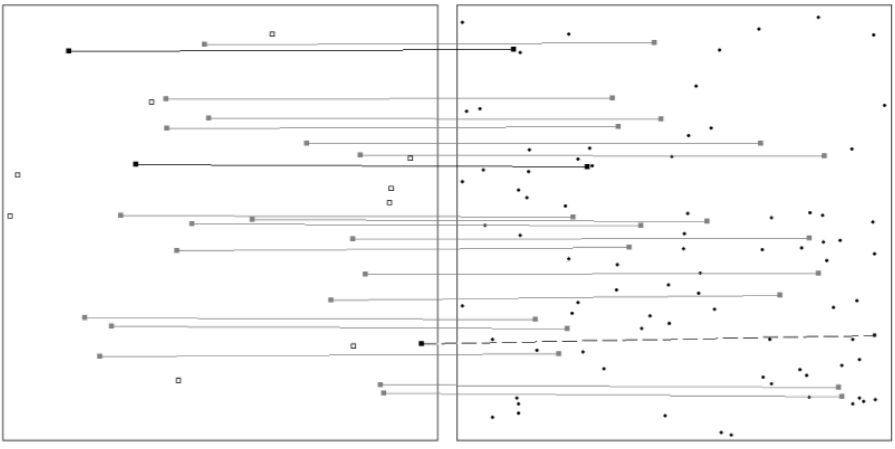


Fig. 2. Example of a 30-to-100 correspondence: (left) original set of points with white boxes showing spurious points, (right) points with added noise $\sigma = 4$ pixels. Correct matches are shown with grey lines; wrong matches with a black line; wrong spurious matches with a dashed line

detection. Nevertheless, the graph matching technique is generic and applicable once image and GIS information are described in terms of attributed graphs. In the experiments, points are randomly scattered within an image of 512×512 pixels. The first set of points contains 30 points and the second set contains 100 points. Both sets have 20 points in common with a perturbation on their position using gaussian noise with a standard deviation between one and eight pixels. The matching result needs to make a distinction between points which are common between the two datasets (so called "real" points) and spurious points. Figure 2 shows an example of this dataset with the first and second set displayed in resp. the left and right frame, and the correspondence computed with graph matching. In this example, the gaussian noise on the position has a standard deviation $\sigma_{noise} = 4$ pixels. The white rectangles in the left frame are added spurious points, which should be assigned the null label. The grey lines show the points which have been correctly associated. The black lines show points which have been incorrectly associated. The dashed black lines show spurious points which have been incorrectly associated.

To determine the optimal weights of the graph matching process, Eq. 9 is used. In these experiments, the parameters of RL have been set at $\triangle\alpha = \pi/16$ and $w_- = -0.5$. Compatible matches are not awarded, meaning that $w_+ = 0$. The data contains a ratio of 10:30 outlier points so that $f^{\emptyset} = 1/3$. Eq. 9 can then be used to determine the weight $w_{\emptyset}$, which varies over the experiments since the graph label error f^- increases as more noise is added to the position. An added difficulty is that the label error f^- is a stochastic variable. To use Eq. 9, we need to determine the value of f^- which optimally makes the distinction between real distorted points and spurious points. This can be done by modeling the

exhibited graph errors of real and spurious points as normal distributions with a certain mean and standard deviation, and taking the maximum likelihood estimate (MLE) as the optimal decision boundary f^-_{opt} (cfr. Figure 3). Label errors f^- below this threshold are then regarded as acceptable errors belonging to real points. Label errors f^- above this threshold are regarded as severe errors belonging to spurious points.

We measured the mean and standard deviation of the graph label error over a selection of 10 image pairs for a given amount of noise σ_{noise}. For real points, the ideal mapping is known and the graph label error for these points can be measured. Table 1 gives a summary of the label error statistics (m_1, σ_1) for the different amounts of noise. For spurious points, we selected the best matching corresponding point in the second dataset. Since this is a combinatorial problem, the match is approximated under the condition of a near ideal mapping, i.e. the real points are mapped on the correct correspondents, the other spurious points are mapped on the null label. Under these conditions, finding the best match for a point is a linear search. For this match, we measure the graph label error that would occur if a spurious point is mapped on his most likely candidate. Measured over the dataset, this gives an mean label error $m_n = 38.5\%$ with standard deviation $\sigma_n = 16\%$. Using MLE on these statistics, the threshold f^-_{opt} can be calculated and consequently $w^{opt}_{\emptyset}$ using Eq. 9.

Table 1 gives the calculated $w^{opt}_{\emptyset}$. These calculated weights are compared to the measured optimal weights $w^{meas}_{\emptyset}$. The weights $w^{meas}_{\emptyset}$ have been determined by plotting the "receiver operating characteristic" (ROC) curve by varying $w_{\emptyset}$. For this curve, sensitivity and specificity are defined as follows:

$$\begin{aligned} sensitivity &= \frac{TP}{TP + FN} \\ specificity &= \frac{TN}{TN + FP} \end{aligned} \tag{10}$$

where $\{TP, FP, TN, FN\}$ stands for true positive, false positive etc. If sensitivity is plotted along the X-axis and specificity along the Y-axis, the optimal performance is defined as the point on the ROC curve closest to the upper right corner $(1, 1)$. The weight associated with this sample is taken as the optimal measured weight $w^{meas}_{\emptyset}$.

Table 1 shows a good correspondence between the predicted optimal weight $w^{opt}_{\emptyset}$ and the measured optimal weight $w^{meas}_{\emptyset}$. There is a slight overestimation with respect to the measured weight $w^{meas}_{\emptyset}$ which becomes more apparent at higher noise levels. This can be due to ambiguous points which are introduced by adding spatial noise. The ground truth which is used to determine true and false positives does not take into account the possibility of optimal point matches changing. Especially at high spatial noise levels this is possible in dense point clouds, since the spatial error can interchange the position of neighbouring points. For the constraint satisfaction problem, this interchange is not detected as the constraints will not be violated, but the ground truth will penalize the found match. Nevertheless, it remains relevant to use Eq. 9 to tune

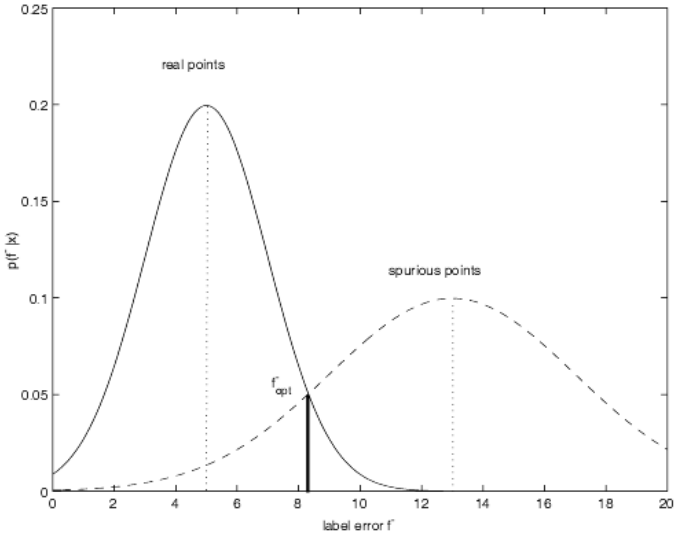


Fig. 3. Maximum likelihood estimate of the optimal decision boundary f^-_{opt}

Table 1. Determining the null weight based on maximal likelihood with respect to the expected graph error. Noise statistics $m_n = 38.5\%$ and $\sigma_n = 16\%$

stdev [pix]	m_1	σ_1	$w_\emptyset^{opt}$	$w_\emptyset^{meas}$
1	0.5%	1.1%	0.05	0.05
2	2.3%	2.8%	0.09	0.07
4	7.8%	6.2%	0.19	0.15
8	21.7%	10.4%	0.32	0.25

the graph matching process based on the expected graph error, as in many applications like road networks such interchanges do not occur often. If it does occur, a minimum point density should be applied to avoid this ambiguous mapping.

4 Conclusion

We have presented a condition based on the expected graph error which allows to determine the bounds of error tolerance in the matching process. The condition allows to characterize acceptable over inacceptable data inconsistencies. The derivation is based on the notion of consistency in inexact graph matching, and is useful to determine the optimal weights of the cost function given the expected graph label error. Experiments on synthetic point sets have shown the relevancy of this condition with respect to the specification of the null weight, which is

typically been determined using rules-of-thumb. Although some problems still need to be solved concerning ambiguous points, the condition allows more control over the desired behaviour of the graph matching problem. This is essential for a reliable use of inexact graph matching in change detection applications.

References

1. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M. (1999) The maximum clique problem. In: Du, D.Z., Pardalos, P.M. (Eds.), Handbook of Combinatorial Optimization, vol. 4. Kluwer Academic Publishers, Boston.
2. Christmas, W., Kittler, J., Petrou, M. 1995. Structural Matching in Computer Vision Using Probabilistic Relaxation. IEEE Trans Pat. Anal. and Mach. Intel. 17(8), 749-764.
3. Gautama, S., Borghgraef, A., Bruyland, I. 2002. Automatic registration of satellite images with GIS databases. In: Proc. Advanced Concepts for Intelligent Vision Systems (ACIVS 2002), Gent, Belgium. 7 pag (on CD-ROM).
4. Hummel, R., Zucker, S. 1983. On the foundations of relaxation labeling processes. IEEE Trans Pat. Anal. and Mach. Intel. 5(3), 742-776.
5. Li S. 1995. Markov Random Field Modeling in Computer Vision. Springer-Verlag, New-York.
6. Pelillo, M., Jagota, A. 1995. Feasible and infeasible maxima in a quadratic program for maximum clique. Journal of Artif. Neural Networks, 2(4), 411-420.
7. Wilson R. 1995. Inexact Graph Matching Using Symbolic Constraints. Ph. D. thesis, Department of Computer Science, University of York.

Asymmetric Inexact Matching of Spatially-Attributed Graphs

Yang Li, Dorothea Blostein, and Purang Abolmaesumi

School of Computing, Queen's University,
Kingston Ontario, Canada, K7L 3N6
{yli, blostein, purang}@cs.queensu.ca

Abstract. This paper discusses inexact matching of graphs that are spatially-attributed and asymmetric. In a spatially-attributed graph, vertex attributes indicate the coordinates of an image feature represented by the vertex. Asymmetry arises when two graphs represent the same data at different resolutions: this causes an edge in the coarse graph to match an entire path in the fine graph. The two graphs may use different coordinate systems, so a coordinate transform must be estimated during the graph matching. We present an edge first graph matching algorithm to solve this problem, and illustrate its application to the registration of satellite images to road maps. In our current implementation, graphs that represent road networks are manually extracted from satellite images and digitized road maps. Most of the existing algorithms are not designed to handle the asymmetry present when matching a coarse graph to a fine graph.

1 Introduction

This paper presents a robust graph matching algorithm that registers spatially-attributed graphs with different resolutions, scales and orientations. This graph-matching problem is motivated by an application in interpretation of geographic data. With widespread availability of both digitized road maps and high-resolution satellite images, there is opportunity to exploit the combination of these two sources of information. A known road map could be used to help interpret a satellite image, providing predictive information for locating features on the satellite image. Alternatively, information extracted from a satellite image could be used to update an outdated road map, or to detect damaged areas after floods or earthquakes [11] [12] [13]. For all of these applications, an important first step is to register the road map to the satellite image.

Figs. 1 and 2 show a digitized road map and the corresponding satellite image, and Fig. 3 shows the correct registration of these two images. We use graph matching to register road maps to satellite images. Other approaches are discussed in [2] [8] [9] [11] [12] [13].

As shown in the white overlays of Figs. 1 and 2, we extract graphs representing road network, with nodes representing intersections, sharp bends, or terminations of roads, and edges representing segments of roads. The graphs are *spatially-attributed* meaning that vertex attributes record the (x, y) coordinates for the road feature

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 253–262, 2005.

represented by the vertex. In our present work, the graphs are created manually, by drawing on a display of the digitized road map or the satellite image. In future work, automatic road-extraction algorithms could be used [3] [6] [10]. In manually extracting graphs from satellite images, we include major roads but omit many small roads. This is meant to approximate the behaviour of road-extraction algorithms, which are likely to obtain a fairly complete road network from the relatively clean and clear road map images, while obtaining a less complete road network from the satellite images.

Several difficulties arise in the graph-matching problem illustrated in Figs. 1 and 2. One difficulty is that graph matching must allow for asymmetry in the level of detail of the two graphs. The more detailed graph (called the *model graph*) contains nodes and edges that are not present in the coarser graph (called the *data graph*). In other words, most of the vertices in the data graph have a corresponding vertex in the model graph, whereas many model graph vertices do not necessarily have a corresponding vertex in the data graph. Therefore, an edge in the data graph needs to be matched to an entire path in the model graph. Another difficulty is that the relative scale, orientation, and alignment of the two images are not known. The graph matching algorithm must find a coordinate transform to relate the positional attributes in the data and model graphs. In this work, we assume that the coordinate transform is affine. Further work is needed to address the problem of radial distortion, which is significant in many aerial photos. Note that the differences in scale and orientation mean that it is not possible to match the entire model graph to the entire data graph. Parts of each graph are unmatchable since they represent geographic regions that are not represented in the other graph.

Most of the existing algorithms for inexact graph matching mainly focus on matching graphs with equal resolution. Therefore, these algorithms are not suitable for solving the asymmetric graph matching problem. We illustrate this by comparing the performance of our algorithm to that of Wilson's algorithm [11] [12] and to that of Shapiro's algorithm [9].

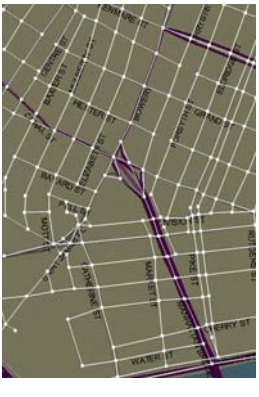


Fig. 1. A road map of Manhattan. In white: the model graph from this image

Fig. 2. Satellite image of Manhattan from spaceimaging . In white: the data graph from this image

Fig. 3. Correct registration of the images in Figs. 1 and 2

The application we discuss in this paper is registration of digitized road maps and satellite images. Other situations in which asymmetric graph matching might be used are when the raw geographic data is not in image form. For example, a vehicle could be equipped with a position sensor and then driven along selected roads. Asymmetric graph matching might also find application in aligning the trajectory of a mobile robot to a given topological or geometric model of the environment. Perhaps images of printed circuit boards could be matched to plans of the circuit board.

2 Description of the Algorithm

In most graph matching methods, vertices are matched first, and then edge mappings are obtained from the mappings of vertices. We call this *vertex-first matching*. In our approach, edges are matched first, and then vertex mappings are derived from the edge mappings. We call this *edge first matching*.

We avoid vertex-first matching because it is difficult to measure the similarity between two vertices when the vertices have different degrees. In our application, vertex degrees are higher in the model graph than in the data graph. Bunke suggests a cost function to use as the matching criterion in vertex-first matching; the computation of the cost function is rather complex, which requires four independent parameters [1]. Furthermore, the prototype of Bunke's cost function does not consider any additional attribute. Myers *et al*. propose a matching criterion based on Levenshtein edit-distance, which does not consider vertex attributes or edge attributes [7]. We need to consider vertex attributes, since these provide positional information that is essential to finding the best graph match. However, the positional attributes cannot be used until the coordinate transform between the model graph and data graph is known. It is difficult to determine this transform during vertex-first graph matching.

We use edge first matching to in our asymmetric graph matching algorithm. The matching is complicated by the need to match a single edge in the data graph to a path in the model graph. As described below, we solve this problem by augmenting the model graph with extra edges, which represent paths in the model graph that are geometrically almost straight. Afterwards, single edges in the augmented model graph can be matched to edges in the data graph. The first edge mapping, between the *seed edge* in the data graph and the *shadow edge* in the model graph, has a special status: this one edge mapping determines the affine transform that relates the coordinate systems used in the model and data graphs. Once this coordinate transform is known, all other edge mappings are found by minimizing the positional differences between the edge endpoints. Fig. 4 illustrates the steps in the algorithm.

The algorithm has two parameters: the augmentation threshold t_{aug} used in Box 1 of Fig. 4, and the standard deviation σ for the Gaussian error function used in Box 6 of Fig. 4. These parameters are discussed further below.

We use $G_D=(V_D, E_D)$ to denote the data graph, where V_D is the set of vertices and E_D is the set of edges. Similarly, the model graph is denoted $G_M=(V_M, E_M)$ and the augmented model graph is denoted $G_{AM}=(V_{AM}, E_{AM})$.

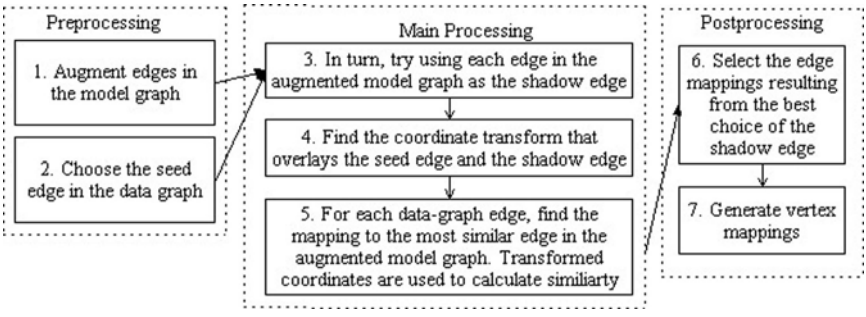


Fig. 4. Flowchart of our asymmetric graph matching algorithm

2.1 Augment Edges in the Model Graph

The first step of the algorithm (Box 1 in Fig. 4) is to augment the model graph with extra edges. This step adds edges to represent paths of the model graph that are geometrically nearly straight. Fig. 5 shows the result of augmenting the model graph in Fig. 1.

To test whether a path in the model graph is nearly straight, we compare the distance between two vertices to the length of the shortest path between these vertices. Distances are calculated as Euclidean distances between the (x, y) coordinates stored as vertex attributes in the graph. We first compute the all-pair shortest paths of the model graph, using Johnson's algorithm [4]. Then we test all pairs of unconnected vertices: add an edge between these vertices if the length of the shortest path is less than t_{aug} times the distance between the vertices.

The augmentation threshold t_{aug} is an important parameter to our algorithm. It must be greater than 1, or the augmentation step does not add edges to the model graph. As reported in Section 3, our experiments use t_{aug} values in the range 1.01 to 1.1. With t_{aug} = 1.01, we observe that in our experimental dataset, the augmented model graph contains approximately 6 times as many edges as the original model graph. With t_{aug} = 1.1, this rises to approximately 50 times as many edges.

2.2 Seed and Shadow Edges Define the Coordinate Transform

Finding the coordinate transform that relates data and model graph is a central step in our algorithm. Since we assume that the coordinate transform is affine, a single mapping between a data-graph edge and a model-graph edge suffices to determine the parameters of the coordinate transform. As shown in boxes 2 to 4 of Fig. 4, the endpoints of a *seed edge* in the data graph are overlaid on the endpoints of a *shadow edge* in the augmented model graph, to determine the coordinate transform. Exhaustive search involves testing $2\,|E_D|\,|E_{AM}|$ possibilities, mapping each of the $|E_D|$ data-graph edges to each of the $|E_{AM}|$ model-graph edges, at two possible orientations. Our algorithm tests only $2\,|E_{AM}|$ possibilities, by fixing one particular edge of the data graph to act as seed edge. The choice of seed edge is discussed further below. (If the seed edge does not have a corresponding edge in the augmented model graph, then our algorithm fails to find the correct coordinate transform, and thus fails to find the

proper graph matching) Once the seed edge has been chosen, we exhaustively search the augmented model graph for an edge that matches the seed edge: we compute $2\,|E_{AM}|$ coordinate transforms, by overlaying every model-graph edge, at two orientations, onto the seed edge. If information about the approximate scale and orientation of the map and satellite images is available, this could immediately eliminate many of the $2\,|E_{EM}|$ coordinate transforms from consideration.

In choosing the seed edge in the data graph, our goal is to choose an edge that is very likely to have a corresponding edge in the augmented model graph. We should avoid choosing the seed edge near the periphery of the data graph, since there can be mismatch between the geographic areas represented by the data and model graphs. Therefore, we choose the seed edge from the *central area* of the data graph. The central area is defined as the central quarter of the area represented by the data graph. If the data graph has extreme coordinates (x_{min}, y_{min}) and (x_{max}, y_{max}), then the central area has extreme coordinates (($3x_{min}+x_{max}$)/4, ($3y_{min}+y_{max}$)/4) and (($x_{min}+3x_{max}$)/4, ($y_{min}+3y_{max}$)/4). The seed edge is chosen as the longest edge that lies within the central area. If no edges lie in the central area, then the seed edge is chosen to be the edge that has the shortest distance to the center point (($x_{min}+x_{max}$)/2, ($y_{min}+y_{max}$)/2). When there is no edge in the central area, the data graph may not work well with partially overlapped model graph.

Once the seed edge in the data graph has been chosen, we iteratively try mapping the seed edge to every possible shadow edge in the augmented model graph. In each iteration, the algorithm transforms the data graph to put it into the Cartesian coordinate system of the model graph. The coordinate attributes in the data graph are recalculated using a translation, scaling, and rotation. This is illustrated in Fig. 6. Mathematical details of the coordinate transform are presented in [5].

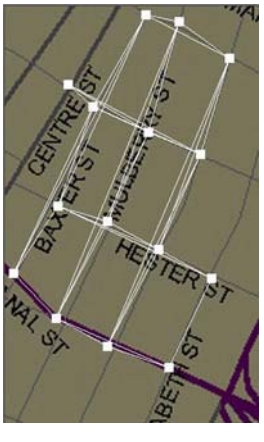


Fig. 5. An augmented model graph. shows a close-up of the graph in Fig. 1, after augmentation

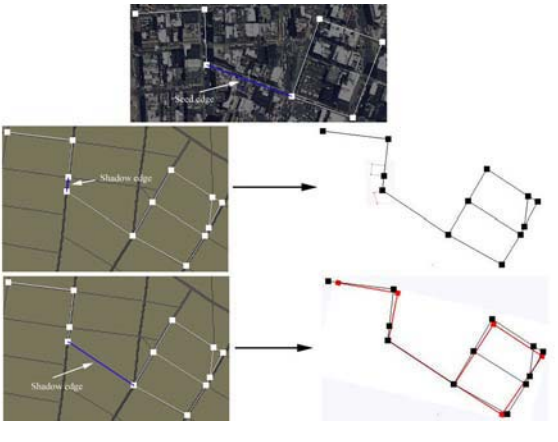


Fig. 6. Coordinate transform resulting from overlaying the seed and shadow edges

2.3 Compute the Rest of the Edge First Mapping

As described above, the mapping from seed edge to shadow edge is used to determine the coordinate transform between data graph and model graph. The next step in the algorithm (Box 5 in Fig. 4) is to consider every edge in the transformed data graph, finding the mapping to the most similar edge in the augmented model graph. The Euclidean distance between vertices is used to calculate similarity scores. The vertex similarity score is

$$P(v_d, v_m) = e^{-\frac{1}{2}\frac{(x_d - x_m)^2 + (y_d - y_m)^2}{\sigma^2}} \quad (1)$$

where data-graph vertex $v_d = (x_d, y_d)$, model-graph vertex $v_m = (x_m, y_m)$, and σ is a parameter that controls the sensitivity to distortions of vertex position. The edge similarity score is computed as the product of the similarities of the edge endpoints. Each data-graph edge is mapped to the edge in the model graph that maximizes the edge similarity. The overall graph similarity score, for this particular coordinate transform, is the average edge similarity score, averaged over all of the mappings of data graph edges.

2.4 Use the Edge Mappings to Find Vertex Mappings

Each data-graph edge is independently mapped to the most similar edge in the augmented model graph. This does not provide an unambiguous mapping of vertices: a data graph vertex acts as endpoint for several data graph edges, and these edges may map that endpoint to different vertices in the augmented model graph. A data-graph vertex v_d has a mapping to a fuzzy set of augmented model graph vertices: the fuzzy membership of vertex v_m is determined by the summed similarity scores of all the edges that map vertex v_d to vertex v_m.

2.5 Computational Complexity

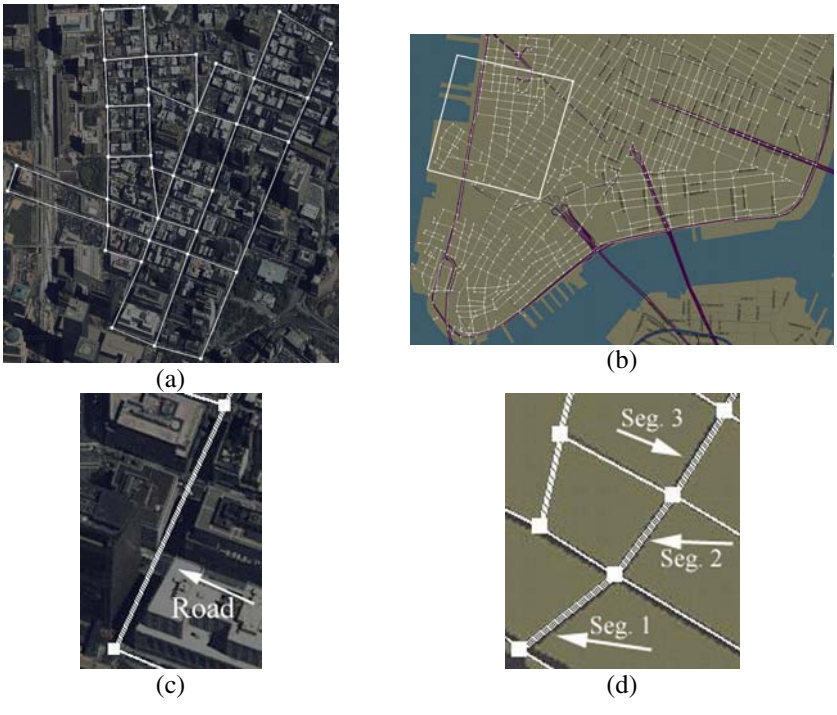
The computational complexity in the worst case is $O((|E_D|+\log|V_M|)|E_{AM}|^2)$. In most cases, the number of edges in the augmented model graph is linear to the number of edges in the model graph. In our experiments, the edge number in the model graph after augmenting edges is increased by 6 times when the augmenting threshold is 1.01. So, the computational complexity in the average case is $O((|E_D|+\log|V_M|)|E_M|^2)$.

3 Experimental Results and Discussions

We conducted a series of 15 tests, using various subimages of a digitized road map and a satellite image of Manhattan, NY. The digitized road map is provided by Rand McNally StreetFinder [16] and the Satellite image is provided by Spaceimaging.com [15]. Data graphs and model graphs are manually extracted from the satellite image and road map. In all 15 test sets, the digitized road map and the satellite image have different alignment, orientation, and scale. For example, in the test set shown in Fig. 7, the road map covers a much larger geographic area than the satellite image.

Sizes of the test sets are as follows: the data graphs contain 15 to 89 vertices, with 19 to 140 edges, and the model graphs contain 567 to 878 vertices, with 909 to 1468 edges. We also tested our algorithm with other digitized road map and satellite images, including a digitized road map of downtown Toronto provided by Microsoft Streets and Trips [14] and a satellite image provided by Spaceimaging.com [15]. The algorithm is implemented in C++ and runs on a Pentium 4-2.6G workstation. Running time for a data graph containing 50 edges and a model graph containing 1500 edges is approximately 20 to 40 seconds.

Each test set is run with various combinations of two parameters t_{aug} (the augmentation threshold) and σ (the standard deviation used in vertex similarity scoring). The effect of changing the t_{aug} is shown in Fig. 8, and the effect of changing σ is shown in Fig. 9.



(a) (b) (c) (d)

Fig. 7. Test set 3. (a) Satellite image with superimposed data graph. (b) Road map with superimposed model graph. The white rectangle indicates the geographic area covered by the satellite image (c) Close-up of a road-edge in the data graph. (d) Close-up of this same stretch of road, represented as three edges in the model graph. The augmented model graph contains an edge from vertex *a* to *d*

We compared our algorithm with Wilson's discrete relaxation graph matching algorithm and Shapiro's feature based algorithm [9] [11] [12]. Our experimental results show that our graph matching algorithm outperforms these algorithms on

matching asymmetric graphs in terms of accuracy rate. As discussed below, Wilson's and Shapiro's algorithms have poor performance when matching asymmetric graphs. This is understandable, since these algorithms are not designed for this task. We are not aware of any existing graph-matching algorithms that are designed to handling asymmetric graphs.

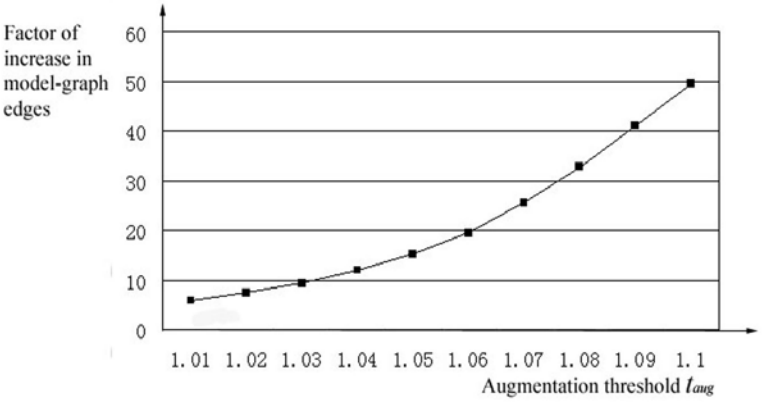


Fig. 8. Size of the augmented model graph, as a function of t_{aug}. When t_{aug}=1.01, the augmented model graph contains approximately 6 times as many edges as the original model graph. When t_{aug}=1.1, this rises to a factor of approximately 50

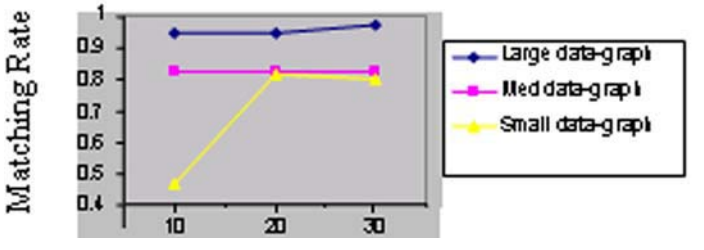


Fig. 9. The effect of changing σ, the standard deviation for matching rate

Under Wilson's discrete symmetric graph matching algorithm, most vertices and edges are mismatched or unmatched on our experimental data. Wilson's algorithm relies on the relationships of vertices between the model graph and the data graph, while these relationships are dissimilar between asymmetric graphs. So, we believe that Wilson's algorithm is not suitable for asymmetric graph matching problems.

Shapiro's algorithm performs well when the data graph and the model graph have no or only little distortion on the features of vertices. However, with the increase of distortion, the correct matching rate decreases dramatically. When we applied Shapiro's algorithm with reasonable values for the parameter σ_x on the test sets used on our algorithm, less than half of the vertices are correctly matched in the best case.

Fig. 10. Successfully matched data graph

Fig. 11. Unmatched data graph

The complete test results are reported in [5]. In these tests we find that small data graphs, with 15 vertices and 19-22 edges, do not provide sufficient context for reliable matching. Medium-sized data graphs, with 35 vertices and 51 to 54 edges, yield more reliable results.

Our algorithm returns perfect results in many cases, including the matching of partially overlapped graphs, as shown in Fig. 7. All edges in the data graph are matched to correct edges in the model graph. Fig. 10 shows another successfully matched data graph which maps to Fig. 7(b).

One cause of the algorithm failure is poor choice of parameter values. It is desirable to keep the augmentation parameter t_{aug} as small as possible, in order to minimize the runtime by minimizing the size of the augmented model graph. In our experiments, the average run time is increased by 30% when we change t_{aug} from 1.01 to 1.02, and is again increased by 155% when we change t_{aug} from 1.02 to 1.05. However, if t_{aug} is too low, edge mismatches result because the augmented model graph is missing needed edges. For example, the road in Fig. 7(c) matches the path from *a* to *d* in Fig. 7(d). When $t_{aug} \geq 1.02$, our augmentation step adds an edge between vertices *a* and *d*. However, if $t_{aug} \leq 1.01$, no such edge is added and therefore no correct match to the edge in Fig. 7(c) can be found. When σ is too small, the matching scores become low and the algorithm may return failure. If σ is too large, some edges may be matched incorrectly.

Periodicity in the road network is another cause of algorithm failure. For example, if the roads are laid out in a grid, as in Fig. 11, then the registration may be shifted by one or more grid locations. Future work to address this problem could include preprocessing roads and intersections, to rank them by their geometric distinctiveness.

4 Conclusion

In this paper, we have formulated and addressed an interesting and practically important graph matching problem. Novel aspects include the asymmetry of the graphs, the use of Cartesian coordinate attributes to match graphs at different scale and orientation, and the use of edge first matching. We have designed a scale, translation, and rotation invariant algorithm for asymmetric graph matching.

In the future, we plan to explore ways to reduce computation time. For large model graphs, we may split them into smaller graphs using an overlapping grid. We can also make use of other vertex and/or edge attributes such as road width and the curve of the road. In addition, we will investigate the use of the algorithm in other applications.

Acknowledgements

This research was financially supported by Canada's Natural Sciences and Engineering Research Council and by a Queen's Graduate Award. We thank Rob Harrap for his helpful comments and suggestions.

References

1. Bunke, H.: "Error Correcting Graph Matching: On the Influence of the Underlying Cost Function," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **21**(9), 917-922, 1999
2. Dare, P.M.: "Automatic registration of multi-source remotely sensed images and other non-image spatial information products," *Proc. of 10th APSPC*, 1150-1157, 2000
3. Geman, D., Jedynak, B.: "An Active Testing Model for Tracking Roads in Satellite Images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **18**(1), 1-13, 1996
4. Johnson, D.B.: "Efficient Algorithms for Shortest Paths in Sparse Networks," *Journal of the ACM*, **24**(1), 1-13, Jan. 1977
5. Li, Y.: *Asymmetric graph Matching for Registering Satellite Images to Road Maps*, M.Sc. Thesis, School of Computing, Queen's University, Kingston, Ontario, Canada, April 2004
6. Merlet, N., Zerubia, J.: "New Prospects in Line Detection by Dynamic Programming," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **18**(4), 426-431, 1996
7. Myers, R, Wilson, R.C., Hancock, E.R.: "Bayesian Graph Edit Distance," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **22**(6), 628-635, 2000
8. Scott, G.L., Longuet-Higgins, H.C.: "An Algorithm for Associating the Features of Two Patterns," *Proc. Royal Society of London*, Vol. B244, 21-26, 1991
9. Shapiro, L.S., Brady, J. M.: "Feature-based Correspondence: an Eigenvector Approach," *Image and Vision Computing*, **10**(5), 283-288, June 1992
10. Shi, W., Zhu, C.: "The Line Segment Match Method for Extracting Road Network for High-Resolution Satellite Images," *IEEE Trans. Geoscience and Remote Sensing*, **40**(2), 511-514, 2002
11. Wilson, R.C., Hancock, E.R.: "Bayesian Compatibility Model for Graph Matching," *Pattern Recognition Letters*, **17**, 263-276, 1996
12. Wilson, R.C., Hancock, E.R.: "Structural Matching by Discrete Relaxation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **19**(6), 634-648, 1997
13. Yanamura, Y., Saji, H.: "Automatic registration of aerial image and digital map for detection of earthquake damaged areas", *Proc. 7th Digital Image Computing: Tech. & App*., 117-126, Dec. 2003
14. Microsoft Streets and Trips: http://www.microsoft.com/streets
15. Spaceimaging.com: http://www.spaceimaging.com
16. Rand McNally StreetFinder: http://www.randmcnally.com

From Exact to Approximate Maximum Common Subgraph

Simone Marini, Michela Spagnuolo, and Bianca Falcidieno

Istituto di Matematica Applicata e Tecnologie Informatiche,
Consiglio Nazionale delle Ricerche,
Via De Marini, 6 - 16149 Genova
{simone.marini, michela.spagnuolo, bianca.falcidieno}@ge.imati.cnr.it
http://www.ge.imati.cnr.it

Abstract. This paper presents an algorithm for the computation of the maximum common subgraph (MCS) between two directed, acyclic graphs with attributes. The core of the contribution resides in the modularity of the proposed algorithm which allows different heuristic techniques to be plugged in, depending on the application domain. Implemented heuristics for robust graph matching with respect to graph structural noise are discussed.

As example of its effectiveness, the algortihm is applied to the problem of 3D shape similarity evaluation through structural shape descriptors.

1 Introduction

Graph matching has been used extensively in a variety of applications and it is particularly useful when the graphs code a description or structure of a shape. By graph matching, or more commonly by subgraph isomorphism, it is possible indeed to assess the similarity among shapes as well as among parts of a shape. Inexact graph matching is also very important for matching structural descriptions of shapes because small features of a shape can cause small differences is the structural descriptions, while similarity should be assessed with stability with respect to noise.

While shape descriptions are usually coded into graph with a relatively small number of nodes, shape databases are composed by hundreds of models of different types and there exists no single similarity measure which optimizes the retrieval results for all shape types. Conversely, heuristics and algorithm flexibility should be left to the user in order to tune the matching to the particular context.

Inexact graph matching has been topic of research since many years and several techniques are available: recently, Demicri et al. in [1] reduce the problem of inexact directed graph matching to the problem of geometric point matching. They use the Earth Mover's Distance as many-to-many matching algorithm among the points. In [2] Messmer and Bunke defined an algorithm for the error correcting subgraph isomorphism (ECSI) detection where the two input graphs

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 263–272, 2005.

are recursively decomposed into smaller subgraphs and the ECSIs with the least edit cost operation are recursively combined to form the complete node matching. Another widely investigated form of inexact graph matching method is the maximum common subgraph (MCS) detection. For example, three algortihms for the exact computation of the MCS are presented in [3]: the first is a state space representation (SSR) algorithm performing a depth-first search in the space of the states, while the other two detect the maximum clique of the association graph built from the two input graphs. In [4] another SSR algorithm for the MCS detection among large graphs is presented. It moves from a generic state to the following one selecting a candidate pair of nodes according to a set of feasibility rules guaranteeing that each state is a common subgraph of the two input graphs.

Many of these techniques have exponential computational complexity and it is therefore necessary to define algorithmical approximation of the optimal solution. The work presented in this paper is aimed at defining a framework for expressing the optimal algorithm in a formalization which makes it straightforward usable for plugging heuristics in it, for achieving different approximations of the optimal solution according to the specific case. The basic idea has been skeched in [5, 6] and here it is fully formalized. An optimal algorithm for the computation of the MCSs is defined on a slight modification of the most naive algorithm: Starting from the list of all mappings among graph nodes, we grow the common subgraphs from each mapping through a process which attempts to add step by step more nodes to the empty initial common subgraph, by expanding at each step two isomorphic subgraphs. This approach may be also seen as a generalization of the state space representation (SSR) algortihm proposed in [7].

In the paper, we will describe in more details how the subgraph expansion works and will we show that using this procedure we get exactly the same results of the naive algorithm, therefore, we get all the MCSs of the input graphs.

2 Basic Definitions

An *attributed graph* is a four-tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu^{\mathcal{V}}, \mu^{\mathcal{E}})$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, $\mu^{\mathcal{V}} : \mathcal{V} \to \mathcal{A}_{\mathcal{V}}$ is a function assigning attributes to the vertices, $\mu^{\mathcal{E}} : \mathcal{E} \to \mathcal{A}_{\mathcal{E}}$ is a function assigning attributes to the edges. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu^{\mathcal{V}}, \mu^{\mathcal{E}})$ be a graph, $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mu^{\mathcal{V}'}, \mu^{\mathcal{E}'})$ is a *subgraph* of $\mathcal{G}$, if $\mathcal{V}' \subseteq \mathcal{V}$, $\mathcal{E}' \subseteq \mathcal{E} \cap (\mathcal{V}' \times \mathcal{V}')$, $\mu^{\mathcal{V}'} : \mathcal{V}' \to \mathcal{A}\mathcal{V}$ and $\mu^{\mathcal{E}'} : \mathcal{E}' \to \mathcal{A}\mathcal{E}$.

A *graph isomorphism* is a bijective function $f : \mathcal{V}_1 \to \mathcal{V}_2$ such that 1) $\mu^{\mathcal{V}_1}(v) = \mu^{\mathcal{V}_2}(f(v)),\ v \in \mathcal{V}_1$. 2) for all the edges $e_1 = (v_1, v_1') \in \mathcal{E}_1$, there exists an edge $e_2 = (f(v_1), f(v_1')) \in \mathcal{E}_2$ such that $\mu^{\mathcal{E}1}(e_1) = \mu^{\mathcal{E}_2}(e_2)$. Moreover, for all the edges $e_2 = (v_2, v_2') \in \mathcal{E}_2$, there exists an edge $e_1 = (f^{-1}(v_2), f^{-1}(v_2')) \in \mathcal{E}_1$ such that $\mu^{\mathcal{E}_1}(e_1) = \mu^{\mathcal{E}_2}(e_2)$. If a graph is not attributed, the first condition and the equality between the edge attributes in the second condition, are not necessary. If $f : \mathcal{V}_1 \to \mathcal{V}_1'$ is a graph isomorphism between $\mathcal{G}_1$ and $\mathcal{G}'$, and $\mathcal{G}'$ is a subgraph of $\mathcal{G}_2$, then f is called a *subgraph isomorphism* from $\mathcal{G}$ to $\mathcal{G}'$. A *common subgraph* of $\mathcal{G}_1$ and $\mathcal{G}_2$ is a graph $\mathcal{G}$ such that there exists a subgraph

isomorphism from $\mathcal{G}_1$ to $\mathcal{G}$ and from $\mathcal{G}_2$ to $\mathcal{G}$. A *maximum common subgraph* of $\mathcal{G}_1$ and $\mathcal{G}_2$, denoted as $\mathcal{MCS}_{\mathcal{G}_1,\mathcal{G}_2}$ is a common subgraph $\mathcal{G}$ such that there exists no other common subgraph having more nodes than $\mathcal{G}$. The $\mathcal{MCS}_{\mathcal{G}_1,\mathcal{G}_2}$ in not necessarily unique.

3 Alternative Formulation of the Naive Solution

A naive algorithm for the computation of the $\mathcal{MCS}_{\mathcal{G}_1,\mathcal{G}_2}$ between the two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is the following: first, enumerate all the possible mappings among $\mathcal{V}_1$ and $\mathcal{V}_2$; second, search those mappings m that satisfy the definition of maximum common subgraph. Of course, m is not necessarily unique.

This algorithm is optimal in the sense that it returns the correct result, but it has an exponential computational complexity. For this reason, heuristic techniques have to be considered in many applications in order to approximate the MCSs of two input graphs. Even if the described algorithm is very simple, it is not easy to define heuristic techniques based on the attributes of edges and nodes, or on reasoning about the graph structure. Also, it is not easy to devise an approximation which makes the structural shape matching robust to structural noise in the graphs. With the aim to introduce such techniques the second point of the algorithm can be modified in the following way: for each listed mapping m, compute the common subgraphs of $\mathcal{G}_1$ and $\mathcal{G}_2$ obtainable by expanding the mapping m.The algorithm `CS_from_Mapping(`m`)` (see Figure 1) expands the input mapping m as much as possible while respecting the definition of common subgraph. Since the first point of both algorithms enumerates all the possible mappings, proving that the second formulation is correct reduces to proving that the expansion always produces a correct common subgraph and that it does not alter the structure of a MCS, should this be given as input to the expansion process.

3.1 Pseudocode of the Proposed Formulation

In this section, the alternative algorithm is described by explaining the steps of its pseudocode, as in Figure 1). The main procedure of the algorithm is `MCS()`. It enumerates the set of initial mappings $\mathcal{M}_{\mathcal{G}_1,\mathcal{G}_2}$ and for each $m \in \mathcal{M}_{\mathcal{G}_1,\mathcal{G}_2}$ it generates the set of the common subgraphs obtained expanding m. `CS` returns the set of the common subgraphs computed by the function `CS_from_Mapping(`m`)` on the input mappings. Obviously, each m may generate more than one common subgraph and the expansion procedure `CS_from_Mapping(`m`)` produces all of them. Finally, the MCSs are obtained selecting the common subgraphs with the largest number of nodes. The procedure `CS_from_Mapping()` trasforms the set of pairs in m in a set of candidates, where each pair is a candidate for growing one common subgraph. In this sense, each candidate contributes to generate the set of common subgraphs, but it also contributes to increment the set of candidates itself. This process continues until the set of candidates becomes empty. The addition of a candidate pair to a common subgraph may not satisfy the definition of subgraph isomorphism, and in this case the `Resolve_Conflict()` procedure

```
MCS(G_1, G_2)
{
  M := Mappings_Set(G_1, G_2)
  for each m in M{
    CS := CS_from_Mapping(m)
    Add(CS, MCS)
  }
  return Max(MCS)
}
```

```
CS_from_Mapping(map)
{
  Init_Candidates(CANDIDATES, map)
  Add((CS, CANDIDATES), CS_SET)
  for each ((CS, CANDIDATES):= cs_elem) in CS_SET{
    While not Empty(CANDIDATES){
      p := (v_1, v_2, *, *) := Pop(CANDIDATES)
      if(Mapped(v_1) or Mapped(v_2))
        Resolv_Conflict(p, cs_elem, CS_SET)
       else{
         Add(p, CS)
         Update(p, CANDIDATES)
       }
     }
   }
   return Max(CS_SET)
 }
```

```
Init_Candidates(CANDIDATES, map)
{
  for each (v_1, v_2) in map
    Add((v_1, v_2, NULL, NULL), CANDIDATES)
}
```

```
Update(p, CANDIDATES)
{
 (v_1, v_2, *, *) := p
 for each edge e_1 outcoming from v_1 {
   for each edge e_2 outcoming from v_2 {
     u = Opposite(v_1, e_1)
     v = Opposite(v_2, e_2)
     Add((u,v, e_1, e_2), CANDIDATES)
   }
 }
}
```

```
Resolv_Conflict(p, (CS, CANDIDATES), CS_SET)
{
 (v_1,v_2, *, *) := p
 NEW_CS := CS
 NEW_CANDIDATES := CANDIDATES

 if(Mapped(v_1) and Mapped(v_2))
  Reset(v_1, v_2, NEW_CS, NEW_CANDIDATES)
 else if(Mapped(v_1) and not Mapped(v_2))
       Reset(v_1, NULL, NEW_CS, NEW_CANDIDATES)
      else if(not Mapped(v_1) and Mapped(v_2))
            Reset(NULL, v_2, NEW_CS, NEW_CANDIDATES)
 Add(p, NEW_CS)
 Update(G_1, G_2, p, NEW_CANDIDATES)
}
```

```
Reset(v_1, v_2, CS, CANDIDATES)
  {
   if(v_1 not NULL){
    Delete((v_1,*),CS)
    for each v s.t. Link(v,v_1,CS)
     Delete((v, *), CS)
    for each v s.t. Link(v,v_1,CANDIDATES)
     Delete((v, *),CANDIDATES)
   }
   if(v_2 not NULL){
    Delete((v_2, *),CS)
    for each v s.t. Link(v,v_2,CS)
     Delete((v, *), CS)
    for each v s.t. Link(v,v_2,CANDIDATES)
     Delete((v,*),CANDIDATES)
   }
  }
```

Fig. 1. The pseudocode of the main steps of the proposed algorithm

is executed to correctly handle the situation. The expansion process starts with the call to `Init_Candidates()` (figure 1) which initializes the set of candidate pairs of mapped nodes, `CANDIDATES`, with the input mapping m. An element of type `CANDIDATES` is a four-tuples (v_1, v_2, e_1, e_2) representing the nodes and edges mappings candidate to belong to the common subgraph. `CS_SET` is defined by a set of pairs (`CS`, `CANDIDATES`) where also `CS` is a set of four-tuples (v_1, v_2, e_1, e_2) representing the nodes and edges mappings of the common subgraph. After the initialization, `CS_SET` contains the pair (`CS`, `CANDIDATES`) with `CS` $= \phi$.

The next step consists of adding elements to `CS` by checking if the vertices of the candidate vertices can be mapped correctly. The test consists of checking if the nodes of the candidate extracted through `Pop()` are already mapped or not: if not, that is if they do not belong to any four-tuple in `CS`, they are added to `CS`, and

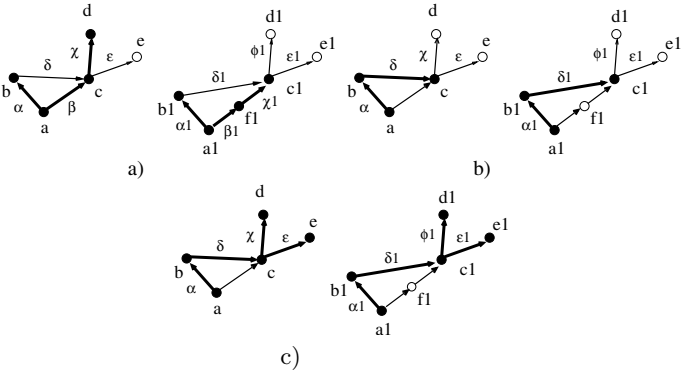


Fig. 2. The conflict between the candidates (`c,c1`,$\delta, \delta 1$) of the two graphs shown in a) is solved and a new common subgraph is produced b). Nodes and edges belonging to the maximum common subgraph c) are: {(`a, a1`, NULL, NULL), (`b, b1`, $\alpha, \alpha 1$), (`c, c1`, $\delta, \delta 1$), (`d, d1`, $\chi, \phi 1$), (`e, e1`, $\varepsilon, \varepsilon 1$)}

new candidates are generated by `Update()`; otherwise, the `Resolve_Conflict()` is called to handle the situation. Intuitively, the conflicts are solved by forking the expansion of the current `CS` into two common subgraphs where the new one is obtained by eliminating the pairs of nodes responsible of the conflict and the subsequent part of the common subgraph.To better understand how the `Resolve_Conflict()` works, let us consider the example shown in figure 2, where only bold edges are mapped ones. Therefore we have `CS_SET` = {(`CS`, `CANDIDATES`)} and `CS`= {(`a,a1`,NULL,NULL), (`c,f1`,$\beta, \beta 1$), (`d,c1`,$\chi, \chi 1$), (`b,b1`, $\alpha, \alpha 1$)} while `CANDIDATES` = {(`c,c1`,$\delta, \delta 1$), (`e,c1`,$\varepsilon, \chi 1$)}. If `Pop()` selects the candidate (`c,c1`,$\delta, \delta 1$), then both nodes `c` and `c1` are already mapped and the statement `Resolve_Conflict()` has to be executed. The new pair generated by `Reset()` is (`CS`′, `CANDIDATES`′), where `CS`′ = {(`a,a1`,NULL,NULL), (`b,b1`, $\alpha, \alpha 1$), (`c,c1`,$\delta, \delta 1$)} and `CANDIDATES`′ = {(`d,d1`, $\chi, \phi 1$), (`d,e1`, $\chi, \varepsilon 1$), (`e,d1`, $\varepsilon, \phi 1$), (`e,e1`,$\varepsilon, \varepsilon 1$)}, as shown in figure 2 b). `CS`′ has been obtained from `CS` deleting the two elements involving `c` and `c1`, and the elements whose nodes are connected to `c` or `c1` with a path contained in `CS`. Analogous considerations are used to obtain `CANDIDATES`′ from `CANDIDATES`.

The proposed algorithm is correct if the statement `CS_from_Mapping(`m`)` produces as output a common subgraph $\mathcal{CS}_m = m$, for each input mapping m corresponding to an MCS. The existence of such m is assured by the procedure `Mappings_Set()`. In order to prove the previous assertion three results have to be shown: first, the nodes involved in $\mathcal{CS}_m$ are an injective function among the nodes of the two graphs; second, $\mathcal{CS}_m$ is a common subgraph of the two input graphs, for each $m \in \mathcal{M}_{\mathcal{G}_1,\mathcal{G}_2}$; third, if m is a MCS, then the node mapping related to $\mathcal{CS}_m$ corresponds to the same MCS. A detailed proof of such three results can be found in [8].

4 Approximate Maximum Common Subgraph

The computation of the MCS of two graphs is a common approach for comparing graphs, but its computational costs make the problem not tractable in many application domains. Most importantly, it is often necessary to insert heuristics in the matching process to be bale to adapt the process to the characteristics of the shapes under examination. Let us recall that the MCS is obtained by providing as input to the procedure `CS_from_Mapping()` all the mappings among the nodes of the two input graphs and by selecting the common subgraph with the largest number of nodes. A first sensible improvement of the matching process can be achieved by relaxing the problem setting and allowing a common subgraph to be accepted also an approximated solution.

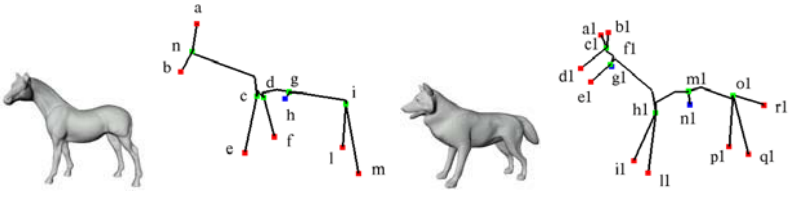
Let us consider the example in Figure 2c): here the optimal solution, that is the MCS, is obtained running the expansion process simply on the pair $m' = \{(\texttt{a},\texttt{a1})\}$. In this case the common subgraph obtained as output from `CS_from_Mapping(`m'`)` corresponds to the MCS of the two graphs, and the process is run on a highly reduced input set of mapping. In general, running the algorithm on a subset of the initial mapping yields to approximations of the MCS, but heuristics or semantic knowledge can be used to select the best candidate initial mappings. It is clear, indeed, that some nodes are more relevant than others, depending on the attributes and on the topology of the graph. A sensible improvement of the computational cost of the process can be obtained by reducing the number of input mappings, at the cost of accepting solutions that are not optimal, that is common subgraphs which might be not maximum. Another important point concerns the type of graphs that are handled. Since the considered input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu^{\mathcal{V}}, \mu^{\mathcal{E}})$ is directed, each node $v \in \mathcal{V}$ identifies a subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mu^{\mathcal{V}'}, \mu^{\mathcal{E}'})$ induced by $\mathcal{V}'$, where $\mathcal{V}'$ is the set of nodes with v as ancestor included v itself. The notion of node relevance can be captured by the subgraph associated to the node: for example, the larger the subgraph associated to the node, the more relevant the node. With reference to Figure 2c), the node `c` is more relevant than `d` and the node `a` is more relevant than `c`. This concept of relevance can be used to drive the selection of the best initial candidates for the process.

Another useful heuristics can be constructed associating the notion of subgraph relevance to the idea of expansion process. For example we might associate to the pair of nodes (v_1, v_2) the information about how much the common subgraph would expand with the addition of that pair to the subgraph. A distance function d between two nodes v_1 and v_2 could be defined which retrieves this information. The distance $d(v_1, v_2)$ can be defined involving node and edge attributes and an approximation of the structure of the subgraph related to v_1 and v_2. Two examples of distances that can be plugged in the algorithm come from [9] and [10]. In [9] a topological signature vector $\chi(v)$ describing the structure of the subgraph related to the node v is defined for each node of the graph. The distance $d(v_1, v_2)$ corresponds to the euclidean distance between $\chi(v_1)$ and $\chi(v_2)$. In [10] the distance value depends mainly by the node attributes. The distance d can be used to reduce the number of elements of `CS_SET`. It acts on the selec-

tion of the $CANDIDATES$ elements and the `Resolve_Conflict()` statement. The simplest way to use d is to extract the best element $(v_1, v_2, \cdot, \cdot)$ (minimum distance between v_1 and v_2) from $CANDIDATES$ and add it to CS if and only if neither v_1 nor v_2 are already mapped. If v_1 or v_2 are mapped, the candidate is discarded and a new one is extracted until $CANDIDATES$ becames empty. In this case the `Resolve_Conflict()` statement is never recalled. Another example of use of d is to add the best candidate (v_1, v_2) to CS even if v_1 and/or v_2 are already mapped if and only if the new mapping has a minor distance than the previous ones. In both the previous cases the `CS_SET` set corresponds to a single pair $(CS, CANDIDATES)$.

5 Experiments

In this section the algorithm has been tested on structural descriptors of 3D objects. A very useful structure for shape description is the Reeb graph, which has been recently used in several application of shape matching. The Reeb graph can be coded into a directed acyclic graph with attributes describing the shape structure of the object [5, 6, 7]. In the experiments presented, nodes having only incoming or outcoming edges describe protrusions of the object, while the remaining nodes describe branching parts. In figure 3 two example of 3D models and graphs are shown, which represent a horse (Figure 3.a) and a wolf (Figure 3.b). The two models are similar but due to small morphological differences the two structural graphs are slightly different: the two subgraphs related to the heads are different and the two front legs of the horse are connected to two different branching nodes, while the two front legs of the wolf are joined to the same branching node. In this case the MCS could give the optimal solution from the point of view of the combinatorial structure, but not an optimal solution with respect to the semantic of the shapes and of their structural descriptors. An approximate solution works better, and different possibilities have been tried. First of all, a distance measure $d(v_1, v_2)$ between two nodes has been experimented in [7]. It roughly describes the differences between the two subgraph related to v_1 and v_2. Each subgraph S is represented by the vector $S(V) = (sub\_n, in(v), out(v), sub\_in(v), sub\_out(v), sub\_s(v))$, where sub_n is the number of



a) b)

Fig. 3. Horse and wolf models together with their graphs

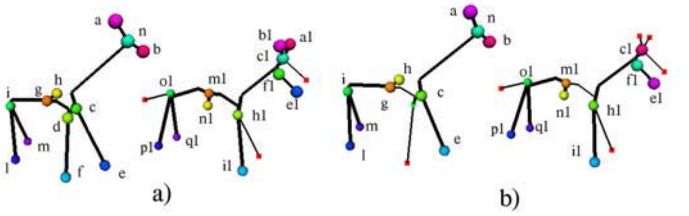


Fig. 4. Two matching between the horse and wolf graphs. a) the common subgraph without grouping is: $\{\mathtt{h} \rightarrow \mathtt{n1}, \mathtt{g} \rightarrow \mathtt{m1}, \mathtt{d} \rightarrow \mathtt{h1}, \mathtt{c} \rightarrow \mathtt{f1}, \mathtt{e} \rightarrow \mathtt{e1}, \mathtt{n} \rightarrow \mathtt{c1}, \mathtt{a} \rightarrow \mathtt{b1}, \mathtt{b} \rightarrow \mathtt{a1}, \mathtt{i} \rightarrow \mathtt{o1}, \mathtt{l} \rightarrow \mathtt{p1}, \mathtt{m} \rightarrow \mathtt{q1}\}$. b) the common subgraph with grouping is: $\{\mathtt{h} \rightarrow \mathtt{n1}, \mathtt{g} \rightarrow \mathtt{m1}, \mathtt{c} \rightarrow \mathtt{h1}, \mathtt{e} \rightarrow \mathtt{i1}, \mathtt{n} \rightarrow \mathtt{f1}, \mathtt{b} \rightarrow \mathtt{c1}, \mathtt{a} \rightarrow \mathtt{e1}, \mathtt{i} \rightarrow \mathtt{o1}, \mathtt{m} \rightarrow \mathtt{q1}, \mathtt{l} \rightarrow \mathtt{p1}\}$

nodes belonging to S, $in(v)$ and $out(v)$ represent the indegree and outdegree of v, $sub\_in(v)$ and $sub\_out(v)$ the indegree and outdegree node sum and $sub\_s(v)$ the sum of the subgraph edge attributes. The distance $d(v_1, v_2)$ is defined as euclidean distance between $S(v_1)$ and $S(v_2)$.

Two methods for the selection of the starting set of node mapping have been experimented using the node distance measure defined above. The resulting common subgraphs are shown in figure 4. Both the methods are based on the concept of node relevance mentioned in section 4, and they select all the nodes whose relevance is bigger than the mean value of the nodes graph relevance. The initial set of candidates of the first method is obtained generating all the possible pairs among the selected nodes, that is: $\{\mathtt{c}, \mathtt{d}, \mathtt{h}, \mathtt{g}\}$ for the horse-graph and $\{\mathtt{f1}, \mathtt{g1}, \mathtt{h1}, \mathtt{m1}, \mathtt{n1}\}$ for the wolf-graph. The approximation of the MCS is shown in Figure 4a). In this case the common subgraph generated by the algorithm correspond to the MCS. Even if this is a good result for graph matching, it is not for 3D object comparison because semantically equivalent object parts are not related between themselves. The mapping $\{\mathtt{c} \rightarrow \mathtt{f1}, \mathtt{e} \rightarrow \mathtt{e1}\}$ (Figure 4a)) associates part of the body and a leg of the horse to part of the head and mouth of the wolf. The second method tested generates the set of initial candidates grouping the relevant nodes with respect to the attributes values, and these candidates are: $\{\mathtt{g} \rightarrow \mathtt{m1}, \mathtt{h} \rightarrow \mathtt{n1}, \mathtt{c} \rightarrow \mathtt{h1}, \mathtt{d} \rightarrow \mathtt{h1}, \mathtt{c} \rightarrow \mathtt{f1}, \mathtt{c} \rightarrow \mathtt{g1}\}$. The common subgraph obtained as output from the algortihm does not correspond to the MCS (figure 4b)) but, due to the initial node grouping, semantically equivalent parts of the two objects have been associated.

The previous experiments are mainly based on graph structural analysis, while node attributes providing geometric information can be considered modifying the distance function between nodes: $d(v_1, v_2) = \frac{w_1 G\_S + w_2 St\_S + w_3 Sz\_S}{w_1 + w_2 + w_3}$, where G_S, St_S, $Sz\_S \in [0, 1]$. G_S and St_S respectively represent the geometric similarity between the node attributes and the structural similarity between the node subgraphs. Sz_S evaluates the similarity between the size of the sub-parts associated to the nodes, where the size corresponds to the sum of the lengths of the subgraph edges. Finally, the three weights $w_1, w_2, w_3 \in [0, 1]$, combine the three components of d. G_S compares the geometric signatures

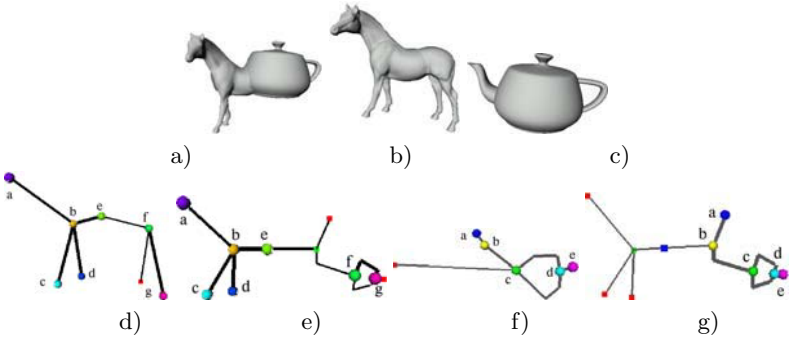


Fig. 5. Sub-part correspondence among a model a) obtained mixing an horse b) and a pot c). The mixed-model compared with the the horse d)-e) and with the pot f)-g). Matched nodes have the same label

associated to v_1 and v_2. In our experiments the signature of each node is obtained decomposing the sub-part surface into a collection of functions defined on concentric spheres and using spherical harmonics to discard orientation information for each one [11]. Then, the similarity between the subgraph structures is defined as: $St\_S = \frac{\overline{in}+\overline{out}+\overline{sub\_n}+\overline{sub\_in}+\overline{sub\_out}}{5}$, where $\overline{X} = \frac{|X(v_1)-X(v_2)|}{max(X(v_1),X(v_2))}$, Finally $Sz\_S = \overline{sub\_s}$, where sub_s is the sum of the edge attributes of the subgraph. In figure 5 sub-part correspondences among semantically equivalent parts has been shown. The algorithm has been used to approximate the common subgraph minimizing geometric differences between the object sub-parts. Also in this case the MCS between the two objects should not represent the best matching.

6 Conclusions

An algorithm for the computation of the MCS between two directed acyclic graphs has been presented. The algorithm is tailored to encapsulate heuristic techniques that may yield to an approximation of the MCS only, but that make the algorithm suitable for applications where graph structural noise may compromise the results and computational costs have to be optimized, both in terms of time consumed and occupied memory space. Different heuristic tecniques have been described and their effectiveness has been shown and discussed with respect to shape structural descriptors of 3D objects. The experiments show that the MCS is not the best solution for the discuussed application domain. The node relevance, the distance between graph nodes and the way they are used to reduce the elements of `CS_SET`, produces a common subgraph approximating the MCS such that it is maximal and that minimizes the geometric and structural graph differences.

References

1. Demirci, M.F., Shokoufandeh, A., Dickinson, S., Keselman, Y., Bretzner, L.: Many-to-many feature matching using spherical codng of directed graphs. In: Proceedings of the European Conference on Computer Vision., Prague (2004)
2. Messmer, B.T., Bunke, H.: A new algorithm for error tolerant subgraph isomorphism detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998) 493–504
3. Conte, D., Guidobaldi, C., C.Sansone: A comparison of three maximum common subgraph algorithms on a large database of labeled graphs. In Hancock, E., Vento, M., eds.: Proc. of IAPR Workshop GbRPR 2003. Volume 2726 of Lecture Notes in Computer Science., Springer-Verlag Berlin Heidelberg (2003) 130–141
4. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence **26** (2004) 1367–1372
5. Biasotti, S., Marini, S.: 3D object comparison based on shape descriptors. International Journal of Computer Applications in Technology (To be Published)
6. Biasotti, S., Marini, S.: 3D object comparison based on shape descriptors. Technical Report 17/04, Istituto di Matematica Applicata e Tecnologie Informatiche, Consiglio Nazionale delle Ricerche, Genova (2004)
7. Biasotti, S., Marini, S., Mortara, M., Patané, G., Spagnuolo, M., Falcidieno, B.: 3D shape matching through topological structures. In Nyström, I., di Baja, G.S., Svennson, S., eds.: Proceedings of the 11^{th} Discrete Geometry for Computer Imagery Conference. Volume 2886 of Lecture Notes in Computer Science., Naples, Springer Verlag (2003) 194–203
8. Marini, S.: Methods for common subgraph approximation. Technical Report 23/2004, Istituto di Matematica Applicata e Tecnologie Informatiche, Consiglio Nazionale delle Ricerche, Genova (2004)
9. Shokoufandeh, A., Dickinson, S.: A unified framework for indexing and matching hierarchical shape structures. In: Proceedings of 4th International Workshop on Visual Form, Capri, Italy (2001)
10. Hilaga, M., Shinagawa, Y., Komura, T., Kunii, T.L.: Topology matching for fully automatic similarity estimation of 3D shapes. In: ACM Computer Graphics, (Proc. of SIGGRAPH 2001), Los Angeles, ACM Press (2001) 203–212
11. Kazdan, M., Funkhouser, T., Rusinkiewicz, S.: Rotation invariant spherical harmonic representation of 3D shape descriptors. In Kobbelt, L., Schröder, P., Hoppe, H., eds.: Proceedings of Symposium in Geometric Processing, Aachen, Germany (2003) 156–165

Automatic Learning of Structural Models of Cartographic Objects

Güray Erus and Nicolas Loménie

Université de Paris 5, Laboratoire SIP-CRIP5,
45 rue des Saints Pères; 75006; Paris; France

Abstract. A model of the target object is required for the recognition of cartographic objects in satellite images. We developed a learning system that constructs the structural models for cartographic objects automatically. Using a database of examples extracted from satellite images, this system constructs the abstract model of the object in each class. The images containing the objects are decomposed into primitive figures and are transformed to Attributed Relational Graphs (ARGs) that are very appropriate for the representation of structured data. We generated the object models applying graph-matching algorithms on these graphs. The quality of a model is evaluated by a specific edit-distance of the examples to the model.

We tested our system on images of bridges and roundabouts. We could obtain object models compatible with manually generated models.

1 Introduction

With the spread of high-resolution satellite images, more sophisticated image processing systems are required for the automatic extraction of information. In the SIP[1] laboratory, two different systems of cartographic object detection in satellite images have been developed in the frame of a research project of CNES[2]. In the terminal step of both systems, a structural model of the target object is used for comparison in order to determine the class of a candidate. This model is defined manually, either by region adjacency constraints [1] or by geometrical constraints [2]. Our aim is to generate the model of cartographic objects automatically using a database of segmented and labeled satellite images. The originality of this work lies in the application of structural learning techniques on cartographic objects which are quite complex and variable.

A survey of model-based object recognition is given in [3]. In most of the existing studies graphs are used to represent the objects. Particularly, Attributed Relational Graphs (ARGs) are preferred to represent structural data. In our system, we used ARGs as the basic data structure. Different techniques have been used to determine the similarity of two images. [4] propose a distance metric

[1] Intelligent Perception Systems.
[2] French space agency.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 273–280, 2005.

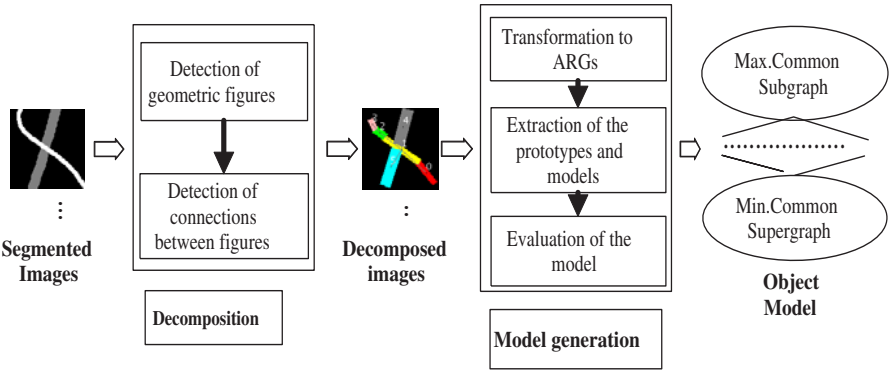


Fig. 1. The system architecture

between two ARGs in order to detect the similar images in an image database. In [5] a projective transformation invariant representation is used on ARGs for matching objects under geometric transformations. [6] propose a method for the automatic learning of a parametric model for modeling a sample distribution represented by ARGs. They use the Expectation-Maximization algorithm to estimate the parameters of likeliness. [7] present a method for learning models of objects represented by ARGs. They introduced a new data structure, Generalized ARG (GARG), which is an ARG with general attribute values. The aim of the system is to generate a set of GARGs that cover all the examples.

To transform an image into a graph, the image is represented either by simple geometric shapes, by its skeleton or by its shock-graph, a representation based on the medial axis of the object. A review of graph matching methods is presented at [8].

We applied in parallel two different decompositions using geometric shapes and the skeleton of the objects. Our system is composed of two modules (Fig. 1). The pre-processing module uses the segmented and labeled images containing the target object. We have a database of SPOT5-THR images with a resolution of 2.5 meters, in which bridges and roundabouts have been extracted and marked by an expert. The aim of this module is the decomposition of these images using basic geometric shapes.

In the second module, the decomposed images are firstly transformed into ARGs. Then, we detect the prototypes, the most frequent representations of the object, between all the graphs. Using the prototypes, we generate the object model by finding the Maximal Common Subgraph (*MaxCSg*) and the Minimal Common Supergraph (*MinCSg*)of the prototypes. The quality of the model is evaluated by calculating the edit distance of the graphs to the model.

The decomposition of images is presented in section 2 and the construction of the model in section 3. The experimental results are given in section 4. Finally we indicate our conclusions and future directions in section 5.

2 Decomposition into Basic Shapes

Using simple geometric shapes, it is possible to obtain a decomposition that preserves the topology and the spatial relationships of the parts of cartographic objects. We proposed two different decompositions in parallel, one using circles and rectangles, and the other using circles and segments. We developed a system with several parameters representing the threshold values used for determining the shape characteristics. We set these parameters empirically in such a way that the final decomposition contains a minimal number of shapes while it is still representative of the target object. The methods used for detecting the basic shapes and the results of the decomposition are briefly presented in the following paragraphs.

The first step of the decomposition is the detection of the circles. Hough Transform is a classical and robust method used for the detection of simple shapes like lines and circles. We detected the circles on roundabout images using this method. The inner regions of the detected circles are erased from the images.

In order to detect the rectangles, we firstly extracted the edges of the object using a morphologic gradient. We then approximated the edge by line segments. We applied the algorithm of the cord for the approximation. To detect the parallel segment pairs, we used a threshold value that limits the angle between two segments. The rectangles are extracted from the parallel segments. We eliminated the rectangles that do not satisfy certain shape criteria. The figures that we obtained were in general non-connected. However, the connections between

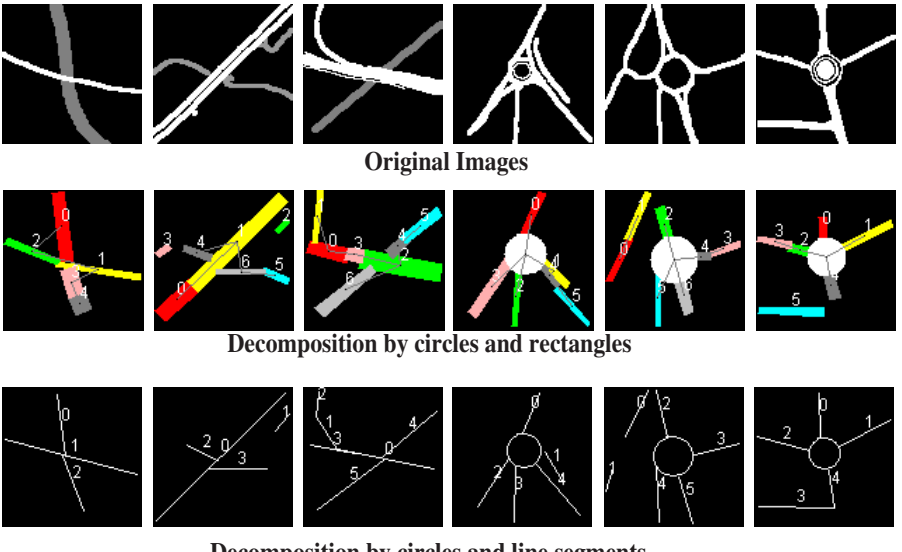


Fig. 2. Decomposition of the images by geometrical figures

objects are crucial in the construction of the graphs. For that reason, we applied a method of elongation to rectangles in order to define their connections.

We used an alternative decomposition using line segments instead of rectangles, in order to obtain a better approximation. To extract the segments we used the thinning algorithm of Zhang-Suen as described in [9]. We converted the skeleton to line segments using the algorithm of the corde. In order to detect the connections between segments, a method similar to the method we used on rectangles is applied .

In Figure 2, we presented some of the images showing the results of the decomposition. In general, we obtained a satisfactory decomposition appropriate for the transformation to ARGs. In some of the images we lost connections between roads and in a few image the result was not representative of the original image.

3 Construction of the Model

The decomposed images are firstly transformed into ARGs. Then we detect the prototypes between the graphs belonging to the same class of objects. The model is defined by the mesh bounded by the ($MaxCSg$) and the ($MinCSg$) of the prototypes. Finally, we calculate the edit distance of each graph to the model in order to evaluate the representative value of the model.

3.1 Construction of ARGs

We represented our objects by ARGs. The characteristics of geometric shapes are stored in the vertices of the graph and the characteristics of connections in the edges. The vertex attributes are:

- The type (t) of the vertex chosen from a finite alphabet $T = \{$*circle, rectangle, segment*$\}$ of possible types.
- The rectangles have the attributes *center, length, width* and *angle with the horizontal.*
- The line segments have the same attributs with rectangles except the *width.*
- The circles have the attributes *center* and *radius.*

The values of these attributes are integer numbers. The edge attributes are:

- The *type of edge* which is a pair of types of the vertices it connects.
- The *direction of connexion* between the components which is discretised by the set $\{$*perpendicular, parallel*$\}$.

3.2 Detection of Prototypes

We proposed a heuristic method in order to reduce the number of graphs used in the construction of the model. In that way, we also eliminate the graphs resulted from a bad decomposition in the first module. We supposed that between all the graphs belonging to an object class, as the number of occurrence of a certain graph increases, this graph becomes more representative of the object. Grouping

the graphs that are isomorphic and sorting these groups by number of elements, we obtained a list of the graphes in the order of frequency. From this list, starting with the most frequent one, we selected a certain number of graphs as prototypes of the object.

We applied an exact graph matching algorithm in order to detect the isomorphic graphs. We did not use all the attributes for the comparison of the graphs, but only a subset of it that does not contain the numerical attributes. The reason of such a simplification is twofolds: the numerical values have a large variation that prevents exact matching between graphs, and also, the values of attributes like the size and the center depend mostly on the global properties of the object (for example the scale of the image) that is not possible to be matched using the structure we defined. The attributes that we used are **the type of the vertex, the type of the edge** and **the direction of the connection**. While the first two attributes provide a matching between figures, the third attribute is used for matching the spatial relation of the connected figures.

The values of the numerical attributes of the detected prototypes are missing. In each group, we detected the mean graph, that is the graph closer to the cumulative mean of the missing attributes, and we used it to set the missing values. In this way, we used some of the attributes to match the graphs and others to find a mean attribute value. Our aim in setting the missing values is to visualize the obtained prototypes.

3.3 Generation of the Model

In order to find the model, we generate the *MaxCSg* and the *MinCSg* of the prototypes. The model is considered to be the mesh bounded by these two graphs, that is the set of all the graphs G, such that $G \subseteq MinCSg$ and $G \supseteq MaxCSg$.

We implemented a recursive algorithm to calculate the *MaxCSg* of two ARGs G_1 and G_2:

$$MaxCSg(G_1, G_2) = \begin{cases} G_2 & if\ \ G_2 \subseteq G_1 \\ \\ max_{\|M_i\|}\{M_i\} & \\ \quad where\ M_i = MaxCSg(G_1, G_i),\ and & \\ \quad G_i = G_2 - v_i, \ \ \forall v_i \in Vertices(G_2) & otherwise \end{cases}$$

We find the *MinCSg* of two graphs G_1 and G_2 using their $MaxCSg$: Let $M = MaxCSg(G_1, G_2)$. The $MinCSg(G_1, G_2)$ is obtained by joining $(G_1 - M)$ and $(G_2 - M)$ to M.

The numerical attributes of the models have been set using the attributes of the prototypes as we have done in the detection of the prototypes.

3.4 The Edit Distance of the Examples to the Model

To evaluate the representative power of the generated model, we calculated the edit distance of each graph to the model. [10] proposed a new distance measure based on the *MaxCSg* of two graphs. The advantage of this metric is that there is no need to define the cost of edit operations.

The distance of two non-empty graphs G1 and G2 is defined as follows:

$$d(G_1, G_2) = 1 - \frac{\|(MaxCSg(G_1, G_2)\|}{max(\|G_1\|, \|G_2\|)}$$

We adapted the metric to the ARGs by calculating the *MaxCSg* for the ARGs. The metric is defined for finding the distance between two graphs. To find the distance of a graph G_1 to the model, we generated G, the set of all graphs covered by the model and found the minimal distance of G_1 to the elements of G.

4 Experimental Results

We tested our system on two classes of segmented images (bridges and roundabouts) and using two different decomposition. We obtained 4 test groups, namely: bridge images decomposed by rectangles (B_R), bridge images decomposed by segments (B_S), roundabout images decomposed by circles and rectangles (R_R), and roundabout images decomposed by circles and segments (B_S).We used 62 bridge images and 54 roundabout images.

The prototypes that we extracted for each different test group is displayed in figure 3. We precised the number of prototypes in such a way that the rate of examples matching with one of the prototypes is higher than a certain threshold. In average 41% of the images for each test group is matching with a prototype.This rate is higher in test groups B_S and R_S. That is the consequence of a more regular decomposition using segments. We obtained similar prototypes of bridges using both decompositions. However this is not the case for the prototypes of roundabouts. This is due to the irregularities in the connections of some of the roads to the central circle. The decomposition using segments missed some of these roads.

Figure 4 presents the models generated from the prototypes. The models are simple and quite similar to manually generated models. They represent the general characteristics of the target objects. However the *MaxCSg* in the R_S test group is incomplete compared with a minimal representation of the roundabouts

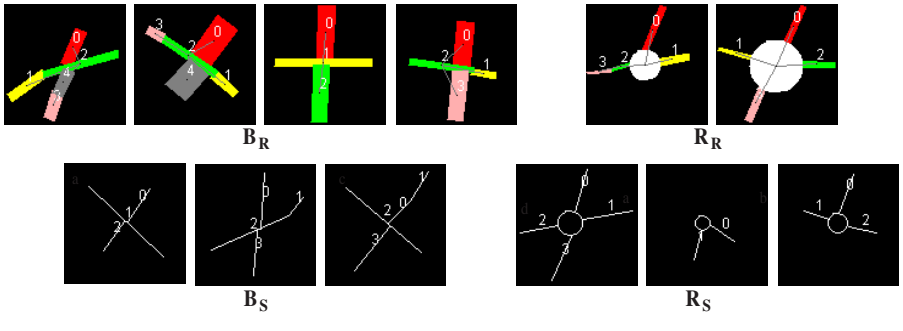


Fig. 3. The prototypes

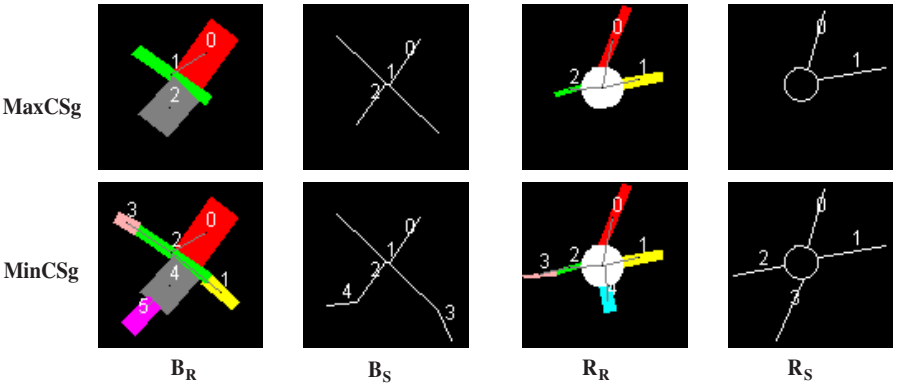


Fig. 4. The models

in the example images. This shows that a systematic deficiency in the decomposition phase may reduce the quality of the model. For the bridges, we obtained quite similar models with the two decompositions.

To evaluate the quality of a model, we calculated the edit distance of each graph to the model of the class it belongs to. According to that criteria, the best model is the B_S with an average edit distance of 0.098. The average distance is 0.1572 for R_S, 0.165 for P_S and 0.1874 for R_S. The cumulative average is quite low and satisfactory, but this metric can not be taken as a direct measure of the quality of the model because it does not take into consideration the quality of the decomposition.

5 Conclusions and Perspectives

This study is a first approach to a difficult problem. Our objective was to develop a coherent and complete system that would be improved with future extensions.

A very important criteria of our work was to obtain a system that is as general as possible. We tried to avoid processings specific to a certain class of object. We could obtain models of bridges and roundabouts similar to manually generated ones.

The selection of the attributes of the ARGs is the central point of the problem. We used a representation that is too direct and sharp. The variances of the geographical objects forced us to limit the number of the attributes we used in the construction of the model. This limitation reduced the representative power of our system.

We can propose several refinements and ameliorations. Considering the numerical attributes, using a fuzzy modelisation of the symbolic concepts and integrating methods issued from qualitative spatial reasoning may enhance the results of our system. A further step will be to check the robustness of our methods on non-segmented satellite images in order to combine it with the existing cartographic object detection systems.

We would like to thank French Space Agency - CNES, and especially Gilbert Pauc, J.C. Favard and Jordi Inglada for their support. We would also like to thank Dr. Z. Hamrouni and Cultural Service of the French Embassy in Ankara.

References

1. Trias R., Loménie L.: Automatic bridge detection in high-resolution satellite images. 3rd International Conference on Computer Vision Systems (ICCVS'03), ser. LNCS. Graz, Austria : Springer (2003) 172-181
2. Loménie N., Trias R., Barbeau J.: Integrating Textural and Geometric Information for an automatic bridge detection. IGARSS'03, Toulouse, France (2003)
3. Pope A.: Model-Based Object Recognition: A Survey of Recent Research. Technical Report 94-04, The University of British Columbia - Department of Computer Science (1994)
4. Petrakis M., Faloutsos C.: Similarity Searching in Large Image Databases. Technical Report 3388, Department of Computer Science,University of Maryland (1995)
5. Shao Z., Kittler J.: Shape representation and recognition based on invariant unary and binary relations. Image Vision Comput. **17** (1999) 429–444.
6. Hong P., Huang T. S.: Spatial pattern discovery by learning a probabilistic parametric model from multiple attributed relational graphs. Discrete Applied Mathematics **139** (2004) 113–135.
7. Cordella L. P., Foggia P., Sansone C., M. Vento: Learning structural shape descriptions from examples. Pattern Recognition Letters. **23** (2002) 1427–1437
8. Bunke H.: Graph matching: Theoretical foundations, algorithms, and applications. in Proc. Vision Interface. (2000) 82–88
9. Lyon, D. A.: Image Processing in Java. Prentice Hall (1999) 297-305.
10. Bunke H.,Shearer K.: A Graph distance metric based on the Maximal Common Subgraph. Pattern Recognition Letters **19** (1998) 255–259

An Experimental Comparison of Fingerprint Classification Methods Using Graphs

Alessandra Serrau[1], Gian Luca Marcialis[1], Horst Bunke[2], and Fabio Roli[1]

[1] Department of Electrical and Electronic Engineering – University of Cagliari,
Piazza D'Armi – I-09123 Cagliari – Italy
{serrau, marcialis, roli}@diee.unica.it
[2] Department of Computer Science – University of Bern,
Neubrueckstrasse 10 – CH -3012 Bern – Switzerland
bunke@iam.unibe.ch

Abstract. In this paper, an experimental comparison among three structural approaches to fingerprint classification is reported. Main pros and cons of such approaches are investigated by experiments and discussed. Moreover, the effectiveness of their measurement-level fusion is analysed. Finally, a comparison among the investigated structural approaches and the well-known statistical approach based on "FingerCodes" is reported.

1 Introduction

The fingerprint classification task consists in associating a given fingerprint to one of the five Henry's classes [1]. The main aim of this task is to decrease the identification time of Automatic Fingerprint Identification Systems (AFIS). These systems are aimed to the criminal identification through the recognition of the fingerprints found in the crime scene. But the identification time can be very long due to the high number of fingerprints in the AFIS data bases (e.g., more than 200 million fingerprints in the FBI data base). Fingerprint classification reduces the identification time by allowing to compare the given fingerprint with the ones associated to the most probable Henry's class, instead of performing a search on the whole data base. As there are five fingerprint classes, a good fingerprint classifier should allow to shorten the identification time to one fifth of the time required without fingerprint classification.

Each one of the above Henry's classes is defined by a particular shape described by the flow of epidermal ridges and valleys of the fingertip. Each shape is characterized by the different position of the "singular points" named "core" and "delta". Figure 1 gives examples of the five Henry's classes and the related position of the singular points. Unfortunately, the fingerprint classification task is made very difficult by several factors. Above the others, the large within-class variability and the small between-class separation [1-2].

In order to address such issues, many approaches to automatic fingerprint classification have been proposed so far. A good survey has been recently reported in [2]. For the purpose of this paper, the approaches can be coarsely subdivided in structural and statistical [3-12]. The former [3-6] is based on the extraction of a set of

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 281–290, 2005.

characteristic measurements, called feature vector, from fingerprint images and used for classification. The latter [6-12] describes fingerprints by production rules or relational graphs, and parsing processes or graph matching algorithms are used for classification. So far, very few attention has been paid to structural approaches to fingerprint classification, mainly due to the lack of simple and effective learning mechanisms, and their high computational complexity of the classification phase. However, recent works have shown the effectiveness of some structural methods [9-11], and their promising performance when they are combined with statistical approaches by measurement-level fusion methods [6, 10]. In particular, structural fingerprint classifiers seem to achieve a good performance for strongly "structured" classes, such as the A and W Henry's classes [10-11] (see figure 1). Moreover, reported results showed that the fusion of statistical and structural fingerprint classifiers can provide the performances of many state-of-the-art fingerprint classification algorithms [3, 6, 10]. However, in our opinion, the role of structural approaches and their effectiveness in fingerprint classification have not yet systematically investigated.

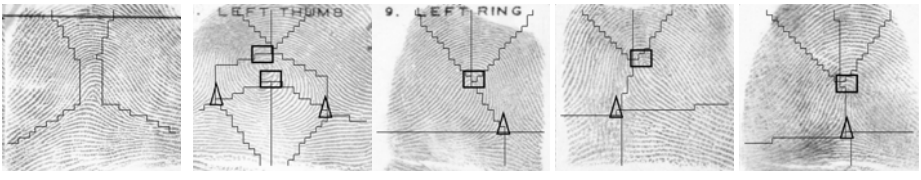


Fig. 1. Examples of the five fingerprint classes of the Henry's classification: (A) Arch (W) Whorl (L) Left Loop (R) Right Loop (T) Tented Arch. The singularity points, named "core" and "delta", are pointed out by squares and triangles, respectively. The "structure" of each class is pointed out by regions with homogeneous orientations which converge around the singularity points

This paper is focused on the experimental comparison of the main approaches to structural fingerprint classification. In particular, we investigated the approaches based on graph-matching [8], dynamic masks [9] and recursive neural networks [6, 10]. Moreover, the measurement-level fusion of the above structural approaches is investigated. Finally, we compared the performance and behavior of the above approaches and their fusion with that of a well-known statistical approach based on the so called "FingerCodes" [3]. The main goal of our investigation is to start the analysis about the advantages and drawbacks of structural methods for fingerprint classification, and also to point out some aspects worthy of further investigations.

The paper is structured as follows. Section 2 describes the graph-based representations and methods used for structural fingerprint classification. Section 3 describes the statistical classifier used for comparison with the structural classifiers. Section 4 describes the experiments performed. Section 5 draws some preliminary conclusions about our investigation.

2 Fingerprint Classification Using Structural Representations

In this section, we briefly describe the rationale behind the use of structural classifiers for fingerprint classification. The use of structural classifiers was introduced by the following observation. Each fingerprint class has a "structure" which can be "defined" through the identification in the fingerprint image of regions characterized by homogeneous direction of the ridges (Figure 1). In particular, such structure is related to the topology of partially overlapped subsets of regions "converging" around the singularity points (the so-called "core" and "delta"), which are differently located from class to class according to the Henry's classification. It easy to see from Figure 1 that the "structure" of each fingerprint class can be extracted by segmenting the images in regions with homogeneous direction of the ridges. Figure 1 also suggests that structural features could be useful to distinguish the A and W classes. In order to identify such a structure, all the approaches we investigated compute and segment the so called "orientation field", which is the map of the ridge and valley orientations of a fingerprint image. In our experiments, we used the special-purpose segmentation algorithm proposed in [12].

The first issue of the investigated structural approaches is to find an appropriate data type to effectively represent the fingerprint structure discussed above. Section 2.1 describes the data types used for the graph-matching and recursive neural networks approaches, which are summarised in sections 2.2 and 2.3, respectively. The dynamic masks approach is described in section 2.4.

2.1 Graph-Based Representations of Fingerprints

One of the most natural structural representation of fingerprint orientation field segmentation (i.e., of the segmentation of the fingerprint image in regions with homogeneous orientation of ridge and valley) is a relational graph. Each graph node can be associated to a segmentation region and the edges join two nodes according to the adjacency relationship of the respective regions [8-12]. The classification algorithm based on this fingerprint representation is described in section 2.2.

Recent works proposed to use directed positional acyclic graphs (DPAGs) to represent the fingerprint orientation field segmentation [6, 10]. This data type also allows using recursive neural networks [6, 10], a machine learning approach explicitly aimed to learn complex data structures [13]. This classification approach is described in section 2.3. To create DPAG representation of the fingerprint image segmented on the basis of their orientation field, the algorithm we proposed in [10] was used. For details about our DPAG generation algorithm, we refer the reader to Ref. 10.

Both graph-based representations are completed by associating to each node a feature vector containing the local characteristics of the regions (area, average directional value, etc) and the geometrical and spectral differences among adjacent regions (relative positions, differences among directional average values, etc) [10].

Figure 2 shows an example of relational graph and DPAG related to the same fingerprint orientation field segmentation. For the DPAG representation, the position of each node is pointed out by an integer labelling each edge.

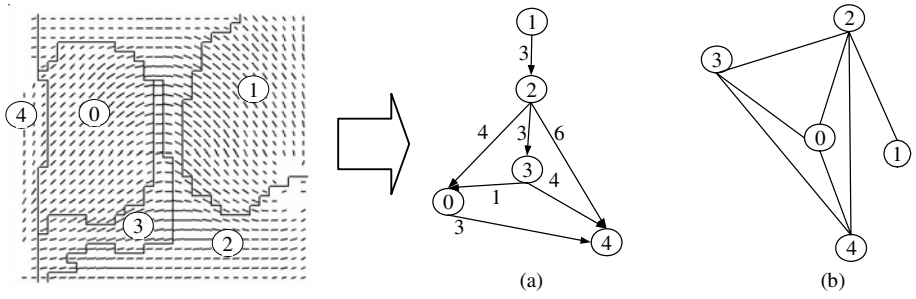


Fig. 2. Graph-based representation obtained from the segmented orientation field of a fingerprint image: (a) DPAG representation obtained from the algorithm described in [10]; (b) relational graph representation. Node labels represent the number associated to each region as shown in the orientation field and the edge labels represent the position of each child-node

2.2 Classification by Graph Matching

The graph matching we used is based on the computation of the graph edit distance between the input graph and a set of prototype graphs representing each class. More than one prototype graph per class is used. The minimum edit distance associated to each class is computed and converted into a posterior probabilities vector. The class associated to the maximum posterior probability is selected.

The edit distance between the input graph and a prototype graph is computed on the basis of a given set of graph edit operations. As an example, the node substitution or the edge deletion can be defined as edit operations [14]. A positive real-value, named "cost", is associated to each edit operation. Given these edit operations, the set of their possible sequences aimed to transform the input graph in the prototype graph is computed. The cost of each sequence is the sum of costs associated to each edit operation of the sequence. The edit distance is defined as the minimum of such costs. The main issue is obviously to find the sequence of edit operations which provide the minimum cost. To this aim, a research tree containing all the possible edit operations sequences is used. The search of the best sequence (i.e. the minimum cost path into the research tree) is performed by an algorithm similar to A* [14]. Further details can be found in [11, 14].

2.3 Classification by Recursive Neural Networks

Recursive neural networks (RNNs) are machine learning architectures which are capable of learning hierarchical data structures [13]. The input to the network is a labeled DPAG $\boldsymbol{U}$, where the label $U(v)$ at each node v is a real-valued feature vector associated with a fingerprint region [6, 10]. A hidden state vector $X(v) \in \Re^n$ is associated with each node v, and this vector contains a distributed representation of the sub-graph dominated by v (i.e., all the nodes that can be reached starting from a directed path from v). The state vector is computed by a state transition function f that combines the state vectors of v's children with a vector encoding the label of v. Computation proceeds recursively from the frontier to the "super-source" (the node

dominating all other nodes). The basic step of such computation is $X(v) = 0$, if v is a missing child. Transition function f is computed by a multi-layer perceptron, that is replicated at each node in the DPAG, sharing weights among replicas. Classification with recurrent neural networks is performed by adding an output function g that takes as input the hidden state vector $X(s)$ associated with the super-source s. Function g is also implemented by a multi-layer perceptron. The output layer uses the *softmax* functions (normalized exponentials), so that $Y = g(X(s))$ can be interpreted as a vector of conditional probabilities of classes given the input graph, i.e., $Y_i = P(C = i \mid U)$, being C a multinomial class variable. Training relies on maximum likelihood and uses a gradient-descent approach to weights optimization of functions f and g called "back-propagation through structure" [13]. Further details can be found in [6, 10, 13].

2.4 The Dynamic Masks Classification Method

This method was introduced by Cappelli et al. to overcome the large variability of segmentations of similar fingerprints [9], which comes out when the segmentation algorithm described in [12] is applied. The basic idea of this approach is to perform a "guided" segmentation of the orientation field of the fingerprint image in order to reduce the variability during the segmentation process [9].

To this end, five filters, called "dynamic masks", one for each class, "guide" the orientation field segmentation, so producing a class-dependent segmentation. Such dynamic masks can be regarded as "prototypes" of images segmented by the orientation field. Using these filters the number of segmentation regions and the coarse region shape are fixed. Each dynamic mask is obtained by the following four steps: (1) for each class, selection of a set of representative fingerprints, (2) computation of the respective orientation fields, (3) application of a genetic algorithm to segment the orientation field and, finally, (4) identification of an "average" ensemble of fixed and mobile vertices and segments that define the mask. Such vertices are located around the singularity points ("core" and "delta"), according to the fingerprint structure showed in figure 1.

To classify fingerprints, the orientation field of an input fingerprint is segmented according to the five dynamic masks (one for each class). For each mask, a "cost" provides a measure of the difficulty of the guided segmentation process. Accordingly, the lowest cost means that the segmentation process can easily produce a segmented image very similar to the used mask. The cost vector is then converted into a posterior probabilities vector. The class associated to the maximum posterior probability is associated to the fingerprint. Further details can be found in [9].

3 The Selected Statistical Approach

In order to analyse how much the structural approach can be useful for fingerprint classification, we compared the performance of the above structural classifiers with that of a well known statistical classifier, using the "FingerCode" as feature vector [3]. In the following, we summarise very briefly the main steps to compute the so called "FingerCode" from a fingerprint image. A circular tessellation is defined

around the "core" point, and subdivided in 48 sectors. A set of 4 pass-band Gabor filters, with orientation selective characteristics, is applied. The absolute deviation of each filter response is computed on each sector, so obtaining a 4x48 real-valued feature vector, called "FingerCode". Further details can be found in [3].

We used the multi-layer perceptron as statistical classifier using FingerCodes [3].

4 Experimental Results

In this section, we report results on the performance of structural classifiers described in section 2.

We firstly compared the performance of structural classifiers, in order to analyse their main pros and cons for fingerprint classification. Then, we investigated their measurement-level fusion. The term "measurement-level fusion" means that the combination rules are applied to the classifiers' outputs, which can be regarded as estimations of the posterior probabilities of each class given the input pattern [15]. The fusion of each vector of probabilities produces a novel posterior probabilities vector which should allow to exploit the complementarity among classifiers and, consequently, to improve the classification performance [15]. We investigated the so-called "fixed" rules, as the mean and the product rule (i.e. the outputs of each classifier are averaged or multiplied), and the "trained" rules. In particular, we investigated the so-called "stacked" approach, where the outputs of each classifier are considered as a novel feature vector. The classification is made using an additional classifier [15]. The additional classifiers we selected are the K-nearest neighbour (KNN) and the multi-layer perceptron (MLP).

Finally, we compared the performance of structural classifiers and their fusion with that of the statistical classifier using FingerCode as feature vector (section 3).

4.1 The Dataset

The well known NIST-4 dataset, commonly used for benchmarking fingerprint classification algorithms, was used for experiments. This data base contains 4,000 fingerprint images subdivided into five fingerprint classes (A, L, R, W, T). In particular, the first 1,800 fingerprints (f0001 through f0900 and s0001 through s0900) were used for individual classifiers training. The next 200 fingerprints were used as validation set, to perform early stopping of the neural classifiers (RNN and MLP) during the learning phase, and to train the classifiers of the stacked approach to classifier fusion. The last 2,000 fingerprints were used as test set.

4.2 Performances of Individual Classifiers

Table 1 reports the class percentage classification accuracies (second to sixth columns) and the overall classification accuracy (seventh column). The second row is related to the dynamic masks method ("masks"), the third one to the recursive neural networks-based approach ("RNN"), and the fourth one to the graph matching approach ("GM").

Table 1. Confusion matrix for the dynamic masks method ("masks"), the recursive neural networks ("RNN"), and the graph matching approach ("GM"). Percentages of the class accuracies and the overall accuracy of the individual classifiers on NIST-4 test set are shown

	A	L	R	T	W	Overall
Masks	48,10	**84,46**	82,14	66,00	78,36	71,45
RNN	**90,71**	79,08	83,30	36,15	81,41	**76,75**
GM	*71,85*	62,32	69,43	52,74	66,25	65,15

The best performance is exhibited by the RNN classifier (table 1, seventh column). This is mainly due to the fact that RNNs do not need of class prototypes, because class representations are automatically learnt by examples. Therefore, RNNs are able to better handle the intrinsic small class-separation of fingerprints, which make difficult to find a representative set of class prototypes to use with GM.

Although the GM classifier performed worse than the RNN on average, their behaviour appears to be similar. Both GM and RNN performed well for the A class, and exhibited the worst performance for the T class of fingerprints. The good performance on class A confirms that structural features could be useful to distinguish strongly structured classes. Accordingly, it can be hypothesised that the performance of the GM approach could be strongly improved if a more robust orientation field segmentation algorithm could be designed, or if effective methods for class prototypes selection would be available.

The dynamic masks classifier performed quite differently with respect to the others. In particular, the performance on the A class is very low. In our opinion, such a low performance can be explained with the absence of singularity points in the A class (figure 1), which make quite difficult to design an appropriate dynamic mask for that class. With regard to this issue, it should be noted that the performance on the other classes, which exhibit at least two singularities, is definitely higher.

4.3 Measurement-Level Fusion of Structural Classifiers

First of all, the complementarity among the investigated structural classifiers was investigated using the so called "oracle", that is, the "ideal" combiner able to select the classifier, if any, that correctly classifies the input pattern. It should be noted that the oracle accuracy is usually a very optimistic estimate of the performance achievable with classifier fusion rules. The performance of the oracle is shown in table 2. The first column shows the "fused" classifiers (the percentage accuracy of the best individual classifier is reported in brackets), the second column shows the performance of the related oracle.

Table 2 points out the strong complementarity between dynamic masks and the other structural classifiers (second and third rows). Such fact could be already hypothesised on the basis of results showed in the previous section (table 1).

It is also worth noting that the highest classification rate can be potentially achieved by combining all three investigated structural classifiers. This means that each classifier can significantly contribute to the performance improvement.

Table 2. Percentage accuracy of the oracle by fusing the classifiers indicated in the first column. The performance of the best classifier is reported in brackets

FUSION of	Oracle
Masks-RNN (76,75)	91,1
Masks-GM (71,45)	89,2
RNN-GM (76,75)	86,1
Masks-RNN-GM (76,75)	94,1

Table 3 shows the performance of individual classifiers and their related measurement-level fusion with different combination rules.

Table 3. Percentage accuracy of the measurement-level fusion of the investigated structural classifiers by the mean rule, the product rule, multi-layer perceptron (MLP) and K-nearest neighbour (KNN). The overall accuracy of the best individual classifier is reported in brackets in the first column

FUSION of	Mean	Product	MLP	KNN
Masks-RNN (76,75)	80,57	79,15	**83,37**	81,74
Masks-GM (71,45)	79,09	77,77	72,58	79,76
RNN-GM (76,75)	76,09	76,70	76,50	76,60
Masks-RNN-GM (76,75)	**82,40**	82,20	83,37	**83,62**

The best performance is achieved by the fusion rules based on KNN when all the three structural classifiers are combined. But the simple mean rule also gives a good performance (table 3, fifth row). Moreover, there is not so much difference between this result and that of Masks-RNN classifiers fusion by MLP (table 3, second and fifth rows). This result points out that the contribution of graph matching classifier is difficult to exploit, probably because of the low performance of this approach. On the other hand, reported results points out the high complementarity between the dynamic masks and RNN classifiers. The improvement of the classification performance is about 7% and, according to the oracle results, there is still room for further improvements (table 2, second row).

4.4 Comparison Between the Structural Classifiers and the Statistical Classifier

Table 4 reports the accuracy on the test set of the statistical classifier mentioned in Section 3, that is, the multi-layer perceptron using FingerCodes. First of all, Table 4 shows that the overall accuracy of the statistical classifier is higher than the one of any structural classifiers and their combination. However, it is evident from tables 1 and 4 that, for the A class, the structural classifiers perform definitely better than the statistical one (except for the dynamic masks method).

In order to investigate the advantages of structural approaches for discriminating classes with a clear structure, for which standard statistical classifiers often perform not well, we analysed in detail the confusion degree between the A and T classes.

Table 4. Percentage class accuracies and overall accuracy of multi-layer perceptron trained with FingerCodes on NIST-4 test set

	A	L	R	T	W	Overall
Statistical classifier	80,51	91,84	89,49	79,13	89,39	86,01

Table 5 shows the confusion degree between the A and T classes (i.e., the percentage of fingerprints of the A class misclassified as T class fingerprints) of the individual structural classifiers, their best fusion, and the statistical classifier. It should be noted that the confusion among such classes is a well-known issue for the state-of-the-art statistical classifiers. Table 5 shows that structural approaches can be useful to recognize strongly structured fingerprint classes, such as the A class. Finally, although the dynamic masks and the GM classifiers do not outperform the statistical classifier individually, Table 5 shows that their fusion definitely improve the performance.

Table 5. Percentage of A-T classes confusion degree of the individual classifiers (masks, RNN, GM), their best fusion, and the statistical classifier. The best fusion has been reported accordingto the best overall accuracy showed in table 3

Classifiers	A – T confusion degree
Masks	19,76
RNN	2,65
GM	19,37
Best fusion Masks-RNN	4,54
Best fusion Masks-GM	5,02
Best fusion RNN-GM	2,88
Best fusion Masks-RNN-GM	5,20
Statistical classifier	16,71

5 Conclusions

In this paper, an experimental comparison among three structural approaches to fingerprint classification was described. Some pros and cons of the use of dynamic masks method, recursive neural networks and graph matching were investigated by experiments and compared with those of a statistical classifier based on FingerCodes.

Experimental results appear to confirm that structural approaches can perform better than statistical ones for the strongly structured fingerprint classes, such as the A class. Moreover, their fusion can help in improving classification performances and, in particular, to reduce the problem of A-T classes confusion degree, which is a well-known issue for currently used statistical classifiers.

Although definitive conclusions cannot be drawn on the basis of the above limited set of experiments, we believe that this paper can contribute to start the discussion about advantages and drawbacks of structural methods for fingerprint classification, and also to indicate some aspects worthy of further investigations.

Acknowledgments

The authors wish to thank Raffaele Cappelli and Davide Maltoni for the results of their image segmentation algorithms, and Anil K. Jain for providing them with the FingerCode representation of NIST-4 data set. Thanks also go to Paolo Frasconi that introduced the authors to the use of recursive neural networks.

References

1. D. Maltoni, D. Maio, A.K. Jain, and S. Prabhakar, Handbook of Fingerprint Recognition, Springer Verlag, 2003.
2. N. Yager and A. Amin, Fingerprint classification: a review, Pattern Analysis and Applications, 7 (1) (2004) 77-93.
3. A.K. Jain, S. Prabhakar, L. Hong, A Multichannel Approach to Fingerprint Classification, IEEE Transactions on PAMI, 21 (4) (1999) 348-358.
4. R. Cappelli, D. Maio, and D. Maltoni, A Multi-Classifier Approach to Fingerprint Classification, Pattern Analysis and Applications, 5 (2) (2002) 136-144.
5. G.T. Candela, P.J. Grother, C.I. Watson, R.A. Wilkinson and C.L. Wilson, PCASYS - A Pattern-Level Classification Automation System for Fingerprints, NIST tech. Report NISTIR 5647, 1995.
6. Y. Yao, G.L. Marcialis, M. Pontil, P. Frasconi and F. Roli, Combining Flat and Structural Representations with Recursive Neural Networks and Support Vector Machines, Pattern Recognition, 36 (2) (2003) 397-406.
7. B. Moayer, K.S. Fu, A syntactic approach to fingerprint pattern recognition, Pattern Recognition, 7 (1975) 1-23.
8. A. Lumini, D. Maio, and D. Maltoni, Inexact graph matching for Fingerprint Classification, Machine Graphics and Vision, 8 (2) (1999) 241-248.
9. R. Cappelli, A. Lumini, D. Maio, and D. Maltoni, Fingerprint Classification by Directional Image Partitioning, IEEE Transactions on PAMI, 21 (5) (1999) 402-421.
10. G.L. Marcialis, F. Roli, and A. Serrau, Fusion of Statistical and Structural Fingerprint Classifiers, Proc. of 4th Int. Conf. on AVBPA'03, (June, 9-11, 2003, Guildford, U.K.), J. Kittler and M.S. Nixon Eds., Springer LNCS 2688, pp. 310-317, 2003.
11. M. Neuhaus, H. Bunke, An Error-Tolerant Approximate Matching Algorithm for Attributed Planar Graphs and Its Application to Fingerprint Classification, Proc. of Int. Conf. S+SSPR'04, (August, 18-20, 2004, Lisbon, Portugal), A. Fred, T. Caelli, R.P.W. Duin, A. Campilho, D. de Ridder, Eds., Springer LNCS 3138, pp. 180-189, 2004.
12. D. Maio, D. Maltoni, A Structural Approach to Fingerprint Classification, Proc. 13th ICPR, Vienna, pp. 578-585, 1996.
13. P. Frasconi, M. Gori, A. Sperduti, A General Framework for Adaptive Processing of Data Structures, IEEE Transactions on Neural Networks, 9 (5) (1998) 768-786.
14. H. Bunke and G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recognition Letters, 1 (1983) 245-253.
15. F. Roli, J. Kittler, and T. Windeatt (Eds.), Multiple Classifiers Systems, Proc. of 5th International Workshop MCS 2004, Cagliari, Italy, June 2004, Springer Verlag LNCS 3077.

Collaboration Between Statistical and Structural Approaches for Old Handwritten Characters Recognition

Denis Arrivault[1,2], Noël Richard[1], Christine Fernandez-Maloigne[1], and Philippe Bouyer[2]

[1] Laboratoire SIC - CNRS - FRE 2731,
SP2MI, Boulevard Marie et Pierre Curie - F-86 962 Futuroscope Cedex,
[2] RC-SOFT , Domaine de la Combe - BP39 - F-16 710 Saint-Yriex
denis.arrivault@etu.univ-poitiers.fr
philippe.bouyer@rcsoft.fr

Abstract. In this article we try to make different kinds of information cooperate in a characters recognition system addressing old Greek and Egyptians documents. We first use a statistical approach based on classical shape descriptors (Zernike, Fourier). Then we use a structural classification method with an attributed graph description of characters and a random graph modeling of classes. The hypothesis, that structural methods bring topological information that statistical methods do not, is validated on Greek characters. A cooperation with a chain of classifiers based on reject management is then proposed. Due to computation cost, the goal of such a chain is to use the structural approach only if the statistical one fails.

1 Introduction

This work can be classified as old characters recognition aided systems. It is a part of an Eureka project named COROC[1], held by the RC-Soft company.

In general, the character recognition systems consist of two steps:(a)features extraction and (b)classification of the feature vectors into a number of class. There are mainly two different types of features, the ones extracting from the whole character image and the others that are describing only parts of it (strokes or primitives). The first feature family is integrated in the statistical recognition approaches, using classifiers as HMM, neural networks or SVM([1]). A good survey of this first feature family can be found in [2]. The second features family are often used with a structural description of the characters like graphs ([3], [4]) or bayesian networks ([5]) for example. Many combinations of those features and classifiers have been made for optimizing the overall recognition rates. Features can be mixed like in [6] and multiple classifiers systems can be used ([7], [8]).

[1] Cognitive Optical Recognition Old Characters.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 291–300, 2005.

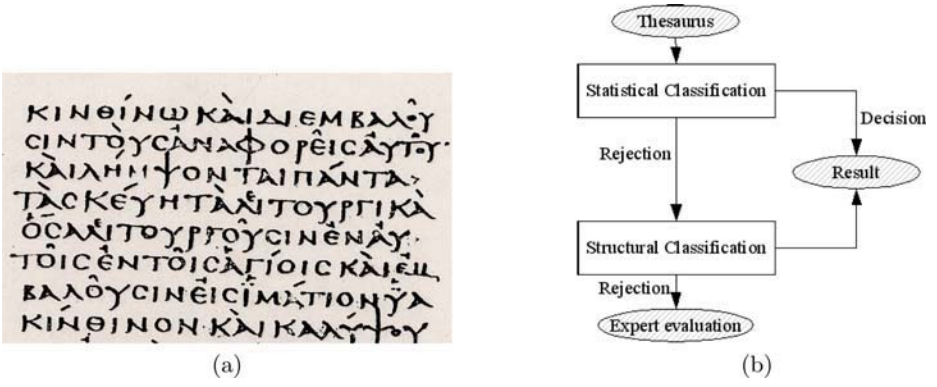


Fig. 1. Greek document example(a), functional chart of cooperation process(b)

The idea , we presents in this article, is to combine both features and classifiers for solving the problem of old greek characters (figure1(a)). Our statistical recognition methods are classical in pattern recognition systems and OCR. They include different shape descriptors (Hu, Zernike, ...) and supervised statistical classifiers (KNN and Bayesian). The second technical process we use is based on a graph extraction which describes the character skeleton. Such an information is at the same time more complex and more complete. That is why we try, by combining both approaches, not only to optimize calculation and decision cost but also to be able to validate the decision hypothesis. Fig.1(b) is a global chart of the cooperation chain, based on a cascaded-representations approach according to Alpaydin & Al([9]), which will be presented in section 4.

In a first section, we develop the statistical level which give the starting hypothesis. In the second part we present the complementary method based on a structural description of the character. Before the results part and the conclusion, we explain in details the cooperation strategy.

2 Statistical Approach

2.1 Principle

Statistical methods are classical in pattern recognition. They describe the geometry of a shape by mathematical descriptors and use statistical tools for classifying the character in the attributes space.

In the numerous existing descriptors, we have chosen to keep Fourier and Zernike moments which describe both contour (Fourier) and region (Zernike). Those two types of descriptors give an attributes vector which is insensitive to the main geometrical transformations (scaling, rotation...). As they are widely used in the pattern recognition community, we will not described here the moments calculation. One can refer to [10], [11] or [12] for further explanations.

2.2 Statistical Classification

We use two types of classification according to the thesaurus size.

For the thesaurus with more than 10 elements in each class, we use a parametric classification after a data reduction made by a principal components analysis. Distributions are supposed to be Gaussian (eq.1) and the Bayes law gives the *a posteriori* probability (eq.2).

$$p(X|\omega_i) = (2\pi)^{-\frac{d}{2}}|\Sigma_i|^{-1}e^{\left(-\frac{1}{2}(X-\bar{X}_i)^T\Sigma_i^{-1}(X-\bar{X}_i)\right)} \tag{1}$$

where $\bar{X}_i$ is the mean vector of the class ω_i and Σ_i is its variance-covariance matrix.

The $p(\omega_k|X)$ estimation is done with the Bayes law :

$$p(\omega_i|X) = \frac{p(\omega_i)p(X|\omega_i)}{p(X)} \tag{2}$$

For the thesaurus with less elements we use the K-Nearest-Neighbors classifier which is a non-parametric classification. The a posteriori probability is then estimated like in eq.3.

$$\hat{p}(\omega_i|X) = \frac{\hat{p}(X,\omega_i)}{\sum_{j=1}^{C}\hat{p}(X,\omega_j)} = \frac{k_i}{k} \tag{3}$$

The decision in both approaches is done by a minimization of the a posteriori probability.

3 The Structural Classification

From a general point of view, the structural classification uses a decomposition of the shape into primitive objects (points and strokes) for describing its relative arrangements. Many types of modelings allow to manipulate those structures. Topological maps for example are interesting tools for describing the spatial organization of the shape [13]. Nevertheless, to our knowledge, there is no robust similarity measurements between topological maps. That is why we have chosen adjacency graphs structures for our modeling. We use attributed graphs with topological information as attributes.

The adjacency graph of a character is built by the description of relations between interest points (figure 2(a)). Interest points in our case are end points, intersection points and inflexion points extracted from the skeleton of the character.

For comparing two graphs we associate attributes to vertices and edges. We detail this structure in the first section of this paragraph. Each class of the thesaurus is then modelized by a random graph. Those random graphs are detailed in the second section. Finally the third section explains the graph matching and the classification used.

3.1 Character Modeling: Attributed Graph

The construction of a graph from a character is made in three steps. First the skeleton is calculated, then the interest points are extracted and finally the attributes are evaluated. The skeleton is a simplification of the shape ([14]) where all strokes have a unit thickness. Among all the skeletonization methods ([15]), only the ones able to give plainly 8-connected skeletons have been considered. Actually, the extraction of end and intersection points from 8-connected skeletons is quite simple. It can be easily done with morphological hit-or-miss operations using suitable masks. The skeletonization used in our work is the Zhang & Wang [16] method which is at the same time perfectly 8-connected and fast.

For obtaining better recognition results we have extended the intersection points class by adding inflexion points. We do not detail the calculation process which is quite complicated and is not the subject of this paper.

The figure2(a) presents some skeletons obtained with the Zhang & Wang method. The detected points are set in black.

Fig. 2. Skeletons and interest points(a), Vertex attributes(b)

The attributed graphs are then built by assimilating the interest points to vertices. The edges represent the lines which connect the points.

The attributes chosen for weighting the edges and the vertices describe the topology of the skeleton. Thus, the attributes associated to vertices are the polar coordinates of the points in a base with the gravity center as reference and the least square line as principal axis. We keep the sines and cosines (rather than the angle for computing reasons) and the distance to gravity center. It gives three attributes between 0 and 1 after normalization (fig.2(b)).

The attributes associated with edges are the curvature (ratio between vertices distance and arc length) and the ratio between the arc length and the longest arc in the skeleton. Those attributes have values between 0 and 1. This modeling allows loops integration which are described as an edge between the same vertex and with null curvature.

Fig. 3. Graph matching

From a formal point of view ([17]), an attributed graph is described as follow. Let us define :

- $\Sigma_v = \{v_k | k = 1, \cdots, n\}$ a set of vertices,
- $\Sigma_e = \{e_{ij} | i, j \in \{1, \cdots, n\}\}$ a set of edges,
- $\gamma_v : \Sigma_v \rightarrow \Pi_v$ a vertex interpreter assigning n_{av} attributes to a vertex,
- $\gamma_e : \Sigma_e \rightarrow \Pi_e$ an edge interpreter assigning n_{ae} attributes to an edge.

Π_v represents the set of legal attribute-value pairs for vertices and Π_e the set of legal attribute-value pairs for edges. In our case $n_{av} = 3$, $n_{ae} = 2$ and all attributes have values in $[0, 1]$.

An attribute graph is defined as a pair (V, A) where $V = (\Sigma_v, \gamma_v)$ is called attributed vertices set and $A = (\Sigma_e, \gamma_e)$ is called attributed edges set.

3.2 Modeling of the Thesaurus: Random Graphs

The random graph is built for modelizing a class.

A random graph is defined in the following way : ([18], [19]):

- $\Sigma_\omega = \{\omega_k | k = 1, \cdots, n\}$ a set of vertices,
- $\Sigma_\epsilon = \{\epsilon_{ij} | i, j \in \{1, \cdots, n\}\}$ a set of edges,
- $\gamma_\omega : \Sigma_\omega \rightarrow \Omega_\omega$ a random vertex interpreter,
- $\gamma_\epsilon : \Sigma_\epsilon \rightarrow \Omega_\epsilon$ a random edges interpreter,
- P : a probability law for random vertices and edges.

Ω_ω and Ω_ϵ are two sets of random variables. In our case those sets are composed by respectively three and two random variables defined in $[0, 1]$.

A random graph is also a triplet (W, B, P) where $W = (\Sigma_\omega, \gamma_\omega)$ and $B = (\Sigma_\epsilon, \gamma_\epsilon)$.

We call the attributed graph $G = (V, A)$ a realization of the random graph $R = (W, B, P)$ if there is a monomorphism $\mu : G \rightarrow R$ such than for ω_i and ϵ_{kl} of R, $a_i = \gamma_v(\mu^{-1}(\omega_i))$ and $b_j = \gamma_e(\mu^{-1}(\epsilon_{kl}))$, then the probability for G to be a realization of R by μ is given by :

$$\begin{aligned} P_R(G|\mu) &= P_R\left(\bigwedge_{i=1}^{n}(\alpha_i = a_i) \wedge \bigwedge_{j=1}^{m}(\beta_j = b_j)\right) \\ &= P(a_1, \cdots, a_n, b_1, \cdots, b_m) \end{aligned} \tag{4}$$

where the $\alpha_i = \gamma_\omega(\omega_i)$ are the random vertices of R and the $\beta_j = \gamma_\epsilon(\epsilon_{kl})$ are the random edges of R.

Practically, it is not easy to estimate the probability P. We have to do an assumption that random variables of vertices and edges are independent with the three following hypothethis :

1. the vertices α_i are independent from each others,
2. a random edge β_j is independent from all vertices except its own two vertices,
3. the conditional distributions $\{\beta_j|\alpha_i = a_i,\ \forall i\}$ are independent from each others.

Then it is demonstrated ([17]) than equation 4 becomes :

$$P_R(G|\mu) = \prod_{i=1}^{n} P_R(\alpha_i = a_i) \times \prod_{j=1}^{m} P_R(\beta_j = b_j|\alpha_k = a_k, \alpha_l = a_l) \qquad (5)$$

where α_k and α_l are the random vertices connected by β_j.

Each thesaurus class is described by a random graph which structure is defined by a model (selected by an expert). The random variables distributions are estimated from the simple graphs attributes of the class and a normality hypothesis.

3.3 Matching and Classification

In the learning phase and in the recognition phase, a monomorphism between a simple graph and the random graph model is needed. In order to simplify this process, we decide to use a graph matching approach. The attributes of the model graph are the mean of all the realizations of the random variables. Thus the problem becomes a classical problem of inexact graph matching.

Many algorithms were developed for solving such a problem. Among those algorithms, the one developed by Chipman & Ranganath ([20]) using a fuzzy relaxation is specially efficient. It first builds a matching graph where vertices represent a matching combination of two vertices (one from each graph) and edges a matching combination of two edges. Each vertex and edge is associated to a weight which is a similarity measurement between the vertices and the edges to match. The weights are reevaluated loop after loop by a classical relaxation rule until they converge. The matching solution is obtained by extracting the maximum clique (with the biggest weight) which conserves the structure of the input graphs.

Nevertheless this algorithm has a main drawbacks. It rather converges on a solution which matches vertices pairs connected by an edge. It is a huge limitation because of the type of characters processed, which are very noisy. And noise often introduces interfering vertices in the middle of an edge. The structural methods of classification are very sensitive to a noisy skeleton, cuttings or spurs. These defaults come from preprocessing and writers characteristics. For solving these problems, we can transform the graphs by applying local transitive closings in order to build what we call "virtual edges". The virtual edges are obtained by adding simple edges (we add the attributes and re-normalize them). Two local transitive closings give virtual edges with a deepness of three. They are built by adding three simple edges. During the matching operation we allow the matching of a simple edge with a virtual edge.

The integration of such a matching with the Chipman & Ranganath's algorithm gives better results. For instance in figure 3, the simple matching give the

following combination of vertices (g1 is the skeleton on the left and g2 the one on the right) : (g1 : 0, g2 : 0), (g1 : 1, g2 : 2), (g1 : 2, g2 : 3), (g1 : 3, g2 : 6). With one local transitive closing (allowing virtual edges with a deepness of two), the combination of vertices becomes : (g1 : 0, g2 : 0), (g1 : 1, g2 : 2), (g1 : 2, g2 : 3), (g1 : 3, g2 : 7), which is more realistic.

After the matching, the recognition is done by calculating the *a priori* probability $P(X|\omega_i) = P(G|\mu)$ with the equation 5. The *a posteriori* probability $P(\omega_i|X)$ comes from the Bayes law (eq.2).

4 Cooperation Process

The process of cooperation is detailed in the chart of figure1(b).

The thesaurus size drives the statistical classifier choice. The parametric method is used with big thesaurus (with at least 10 elements in each class). After the statistical classification, the character is recognized or rejected if the *a posteriori* probability of the most probable hypothesis is too low [21]. If the rejection criteria (eq.6) is satisfied, then the decision is rejected.

$$\delta(X) = max_j\{P(\omega_j|X)\} \leq (1 - a) \tag{6}$$

where a is the error probability obtained by training or by an expert evaluation. We took 0.5.

In the following tests, as the classification is totally supervised, we did not use the distance reject.

If the statistical hypothesis is rejected, a structural recognition is made.

5 Discussion, Results

Two sets of tests were generated. We first have tried to validate our approach with a thesaurus of Greek characters drawn by the hand by four different writers. This thesaurus has been filled with 17641 characters divided into 42 classes. 15641 characters have been used for the training step and 2000 for the recognition step. The results obtained does not show any difference between the statistical approach alone and the cooperative approach ($9,4\%$ of bad recognitions for the first one and $9,2\%$ for the second one). Nevertheless we see an interesting difference in the results of both methods. The figure 4(c) gives the differences ($\Delta\%$ of good recognitions rates) between the cooperative approach and the statistical one ($(\%coop - \%stat)$). If globally the results are the same, we see that, class by class, they are very different. For example for the classes $n13$ and $n30$, the cooperation gives better results in the first case and worst results in the second case.

The figure 4(a) gives two examples ($n30$ and $n4$) of characters skeletons confusing the cooperative method but well recognized by the statistical one (the recognition results are on top, the true characters are on bottom), while the

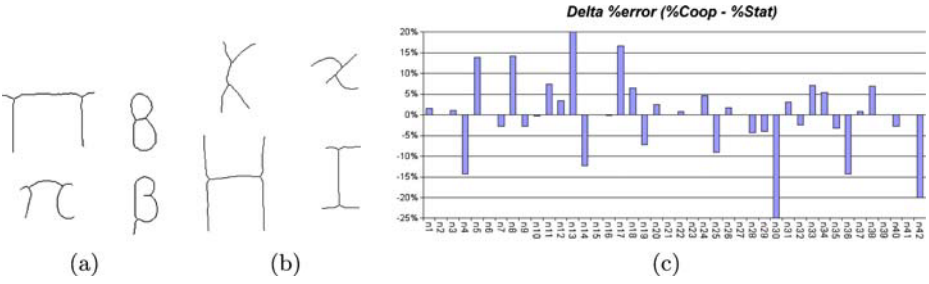


Fig. 4. Skeletons confusing the cooperative approach (a) and the statistical approach (b) (true characters on bottom and confusions on top). (c) represents the detection differences between cooperative and statistical classification with the handwritten thesaurus

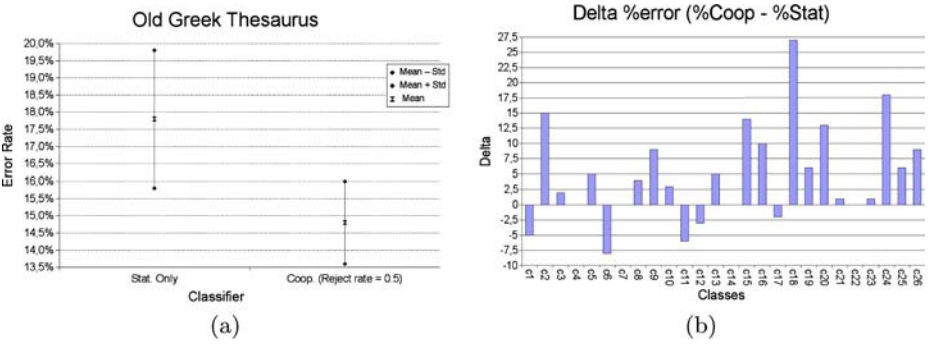


Fig. 5. Results in % of bad recognitions for the old Greek thesaurus(a), good recognitions rates difference between cooperative and statistical classifications for the old Greek thesaurus(b)

figure 4(b) gives two examples ($n13$ and $n17$) of characters skeletons confusing the statistical method but well recognized by the cooperative one.

Secondly, we have realized a set of tests with a thesaurus of old Greek characters extracted from five old handwritten documents of different times. This thesaurus has been filled with 1447 elements divided into 26 significant classes. Regarding to the inhomogeneity of the elements distribution into classes, a cross-validation method has been used for these tests. We have randomly exctracted 5 groups of 200 elements from the thesaurus. Each group has been used alternatively for the recognition while the other groups and the rest of the thesaurus were used for training. This validation method gives a mean recognition error and a standard deviation. The figure 5(a) is a synthesis of the comparative results between the classification methods with and without cooperation. The figure 5(b) gives the differences (Δ% of good recognitions rates) between the cooperative approach and the statistical one ($(\%coop - \%stat)$).

With this set of tests and the first one, we notice that our cooperative approach is at least non-redundant comparing to the statistical one. The global performances are not significatively different but the results are. Nevertheless, the structural classification is not good enough for being complementary with a statistical approach. First the Gaussian modeling for the attributes distributions in the random graphs is too limitative with small thesaurus. Second, a work has to be done on interest points and attributes choices in order to have less noise dependent graphs and more topological information.

6 Conclusions and Perspectives

In this paper a classification of old characters by a cooperation between structural and statistical approaches has been introduced. The specificity of the COROC project, which is working with big and small thesaurus, has forced us to think about this sort of complementary methods.

The results obtained show that our cooperation is interesting. It brings new information but it is not discriminant enough. It is mainly due to two reasons.

First, the performances of the structural classifier alone are not good enough. We can explain it by the independence approximations between random variables or by the lack of robustness of the matching algorithm regarding to noise and attributes choice. That is why we are working on a fuzzy approach which could complement the probabilistic one currently used, and on a new structure for the graphs which will better represent the topological information, and decrease the noise influence. We actually want to introduce a hierarchy between vertices and arcs by defining an influence measurement between them which will weight the matching step.

Secondly, the cooperation can be optimized. We want the algorithm to be able to adapt its own parameters (like thresholds or matching parameters) according to the thesaurus used.

References

1. Bellili, A., Gilloux, M., Gallinari, P.: An mlp-svm combination architecture for offline handwritten digit recognition: Reduction of recognition errors by support vector machines rejection mechanisms. IJDAR **5** (2003) 244–252
2. Trier, O., Jain, A., Taxt, T.: Feature-extraction methods for character-recognition: A survey. PR **29** (1996) 641–662
3. Kang, K.W., Kim, J.H.: Handwritten hangul character recognition with hierarchical stochastic character representation. In: ICDAR. (2003)
4. Chan, K., Cheung, Y.: Fuzzy-attribute graph with application to chinese character recognition. SMC **22** (1992) 402–410
5. Cho, S.Y., Kim, J.H.: Bayesian network modeling of hangul characters for on-line handwriting recognition. In: ICDAR. (2003)
6. Foggia, P., Sansone, C., Tortorella, F., Vento, M.: Combining statistical and structural approaches for handwritten character description. IVC **17** (1999) 701–711

7. Rahman, A., Fairhurst, M.: Multiple classifier decision combination strategies for character recognition: A review. IJDAR **5** (2003) 166–194
8. Kuncheva, L.I., Bezdek, J.C., Duin, R.P.: Decision templates for multiple classifier fusion : An experimental comparison. PR **34** (2001) 299–314
9. Alpaydin, E., Kaynak, C., F., A.: Cascading multiple classifiers and representations for optical and pen-based handwritten digit recognition. In: IWFHR. (2000)
10. Zhang, D., Lu, G.: A comparative study of curvature scale space and fourier descriptors for shape-based image retrieval. JVCIR **14** (2002) 39–57
11. Khotanzad, A., Hong, Y.: Invariant image recognition by zernike moments. PAMI **12** (1990) 489–497
12. Chong, C., Raveendran, P., Mukundan, R.: A comparative analysis of algorithms for fast computation of zernike moments. PR **36** (2003) 731–742
13. Damiand, G., Bertrand, Y., Fiorio, C.: Topological model for 2d image representation: Definition and optimal extraction algorithm. Computer Vision and Image Understanding **93** (2004) 111–154
14. Blum, H.: A transformation for extracting new descriptions of shape. In: Models for the Perception of Speech and Visual Form, MIT Press (1967) 362–380
15. Lam, L., Lee, S.W., Suen, C.Y.: Thinning methodologies - a comprehensive survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **14** (1992) 869–885
16. Zhang, Y.Y., Wang, P.S.P.: A new parallel thinning methodology. PRAI **8** (1994) 999–1011
17. Lu, S., Ren, Y., Suen, C.: Hierarchical attributed graph representation and recognition of handwritten chinese characters. PR **24** (1991) 617–632
18. Serratosa, F., Alquézar, R., A., S.: Function-described graphs for structural pattern recognition. Technical report, Universitat Rovira i Virgili, Tarragona, Spain (1999)
19. Kim, H., Kim, J.: Hierarchical random graph representation of handwritten characters and its application to hangul recognition. PR **34** (2001) 187–201
20. Ranganath, H., Chipman, L.: A fuzzy relaxation algorithm for matching imperfectly segmented images to models. In: IEEE Southeastcon'92. Volume 1. (1992) 128–136
21. Dubuisson, B.: Diagnostic et reconnaissance des formes. In: Diagnostic, Intelligence Artificielle et Reconnaissance des Formes. Traité ic2 diagnostic edn. Hermès Science (2001) 107–140

Decision Trees for Error-Tolerant Graph Database Filtering

Christophe Irniger and Horst Bunke

Department of Computer Science, University of Bern,
Neubrückstrasse 10, CH-3012 Bern, Switzerland
{bunke, irniger}@iam.unibe.ch

Abstract. An important topic in pattern recognition is retrieval of candidate patterns from a database according to a given sample input pattern. Using graphs, the database retrieval problem is turned into a graph matching problem. In this paper we propose a method based on decision trees to filter a database of graphs according to a given input graph. The present paper extends previous work concerned with graph and subgraph isomorphism to the case of error-tolerant graph matching.

1 Introduction

Graphs play an important role in structural pattern recognition. Besides comparing two given patterns, it is often required to match an input pattern against a database of known patterns. If graphs are used to represent structural data, the task of matching patterns is turned into a problem of graph matching. Graph matching is used in a variety of applications, for example document processing [1], image analysis [2] and video analysis [3]. Graph matching is attractive for pattern recognition problems since graphs are a universal representation formalism. It is however a computationally expensive approach. If databases of model graphs are used, an additional factor proportional to the size of the database is introduced in the overall complexity of the matching process. A variety of mechanisms have been proposed to reduce this factor when large databases are involved [4, 5, 6, 7]. In this paper we propose an approach based on machine learning techniques.

In the proposed approach graphs are represented by features which can efficiently be extracted. These features are used to perform a filtering on the database. A filtering procedure is a method which performs a quick and inexpensive reduction of the initial graph database size with respect to a given input graph. The aim of database filtering is to reduce the number of graph candidates in the database that need to undergo an expensive, full fledged graph matching process. A graphical illustration is shown in Figure 1.

The work documented in this paper extends previous studies on graph matching performance and graph database filtering (see [8, 9]). While the works reported in [8, 9] were restricted to the problem of graph- and subgraph-isomorphism, the approach presented in this paper is concerned with error-

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 301–311, 2005.

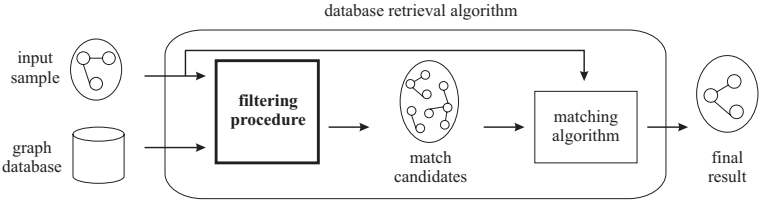


Fig. 1. Illustration of the filtering procedure in the database matching process

tolerant graph matching. In the remainder of this paper, the term 'graph matching' will be referred to in a general context, describing exact matching (such as graph and subgraph isomorphism) as well as inexact matching (graph distance computation and error-tolerant matching).

In the next section, we briefly introduce terminology and graph features used in this study. Then, we show how the concept of graph representation by means of feature vectors can be combined with the decision tree filtering approach. Experimental results will be presented in Section 4, and conclusions drawn in Section 5.

2 Terminology and Graph Features

In this work, structural data or patterns are represented as graphs. A graph is defined as a four-tuple $g = (V, E, \alpha, \beta)$, where V denotes a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges, $\alpha : V \rightarrow L_V$ is a node labelling function, and $\beta : E \rightarrow L_E$ is an edge labelling function. L_V and L_E are finite or infinite sets of node and edge labels, respectively. (Note that this work focuses only on directed graphs; however, the same ideas can be applied to undirected graphs as well.) A subgraph $g_s = (V_s, E_s, \alpha_s, \beta_s)$ of a graph g is a subset of its nodes and edges, such that $V_s \subseteq V$, $E_s = E \cap (V_s \times V_s)$, $\alpha_s(v) = \alpha(v)$ and $\beta_s(e) = \beta(e)$. Two graphs g and g' are isomorphic to each other if there exists a bijective mapping u from the nodes of g to the nodes of g', such that the structure of the edges as well as all node and edge labels are preserved under u. An isomorphism between a graph g and a subgraph g'_s of a graph g' is called a subgraph-isomorphism from g to g'. Given two graphs g_1 and g_2, a common subgraph is defined as a graph g such that a subgraph isomorphism exists from g to g_1 as well as from g to g_2. A common subgraph g of g_1 and g_2 is called a maximum common subgraph, $\text{mcs}(g_1, g_2)$, if no other common subgraph of g_1 and g_2 exists with more nodes than g. (In the remainder of this paper we will use the abbreviation mcs for the maximum common subgraph $\text{mcs}(g_1, g_2)$ of two graphs g_1, g_2 if the context is clear.) Note that the mcs is not necessarily unique for two given graphs g_1, g_2.

In graph matching, it is often required to specify the distance δ between two given graphs. There exist a wide variety of distance functions on graphs (see [10, 11, 12, 13]). The work presented in this paper is based on a distance function which uses the maximum common subgraph of two graphs. It is defined as shown below:

$$\delta(g_1, g_2) = 1 - \frac{|\text{mcs}(g_1, g_2)|}{\max(|g_1|, |g_2|)} \tag{1}$$

Note that this function is known to be a metric (see [10]). Furthermore, in [14, 15] the authors prove that distance δ is equivalent to the well-known graph-edit distance (see [5, 11]) under a certain class of cost functions. Hence, the work presented in the following can be directly related to graph-edit distance as well. In the remainder of this paper, δ will always refer to the distance function defined above.

In this study, feature vectors are used to represent graphs. Motivated by the graph-distance function introduced above, we will focus on features containing information on the number of nodes in the graph. Specifically, we will analyze the number of vertices in a graph with a given label, and use the following notation:

$$f(a, g): \text{ number of vertices in graph } g \text{ with label } a$$

Based on this feature, an estimate on the maximum possible size of the mcs of two graphs g_1, g_2 can be given. The idea is straightforward. Assume that for two graphs g_1, g_2 feature $f(a, g)$ has been extracted for all labels $a \in L_V$. Further assume, without loss of generality, that all nodes in the graphs are labelled. The number of nodes $|\text{mcs}(g_1, g_2)|$ in the maximum common subgraph of g_1, g_2 is given by

$$|\text{mcs}(g_1, g_2)| = \sum_{a \in L_V} f(a, \text{mcs}(g_1, g_2))$$

(Note that by definition a node is only allowed one label.) The sum of the values of feature $f(a, g)$ in the mcs is bounded by:

$$\sum_{a \in L_V} f(a, \text{mcs}(g_1, g_2)) \leq \sum_{a \in L_V} \min\left(f(a, g_1), f(a, g_2)\right)$$

Intuitively speaking this means that per node label 'a' the value of feature $f(a, \text{mcs})$ cannot be larger than the minimum value of that feature in graphs g_1 and g_2. Hence, using feature $f(a, g)$ an estimation $|\text{mcs}_{\max}|$ on the maximum size of the mcs of two graphs (g_1, g_2) can be given:

$$|\text{mcs}(g_1, g_2)| \leq \sum_{a \in L_V} \min\left(f(a, g_1), f(a, g_2)\right) = |\text{mcs}_{\max}(g_1, g_2)| \tag{2}$$

A lower bound on the distance δ (see Equation (1)) is therefore given by:

$$\delta_{\min}(g_1, g_2) = 1 - \frac{|\text{mcs}_{\max}(g_1, g_2)|}{\max(|g_1|, |g_2|)} \tag{3}$$

Using this relation, one can make an approximation on the similarity of two graphs based on the feature $f(a, g)$ introduced before. Assume we need to determine whether or not the distance $\delta(g_1, g_2)$ between a pair of given graphs g_1, g_2

is larger than a specified threshold distance δ_t. If that is the case, the relation $\delta(g_1, g_2) > \delta_t$ will hold where δ_t is defined as

$$\delta_t = 1 - \frac{c}{\max(|g_1|, |g_2|)}.$$

Obviously, the threshold distance δ_t can also be specified in terms of a required minimum size of the mcs. This size is given by variable c. Applying Equations (2) and (3), the following relation can be derived: $\delta_{\min}(g_1, g_2) > \delta_t \quad \Rightarrow \quad \delta(g_1, g_2) > \delta_t$. This simply means that if the approximate distance $\delta_{\min}(g_1, g_2)$ between the graphs g_1, g_2 is larger than the threshold distance δ_t, then the real distance $\delta(g_1, g_2)$ will also certainly be larger than the threshold distance. In that case, it is obsolete to calculate the computationally expensive exact mcs-distance $\delta(g_1, g_2)$. Moreover, by substituting and simplifying the above relation we obtain:

$$c > |\mathrm{mcs}_{\max}(g_1, g_2)|. \tag{4}$$

This means that if the number of nodes of the upper bound estimate $\mathrm{mcs}_{\max}(g_1, g_2)$ is smaller than the threshold number of nodes c, then the two graphs g_1, g_2 will have a distance value larger than the specified threshold distance δ_t. As a consequence, by calculating the estimation of the upper bound of the mcs' size it is possible to decide if the distance between two graphs is larger than a given threshold distance. (Note that Equation (4) is a necessary as well as a sufficient condition to determine whether the distance between two given graphs is larger than a given threshold. However, it is not a sufficient condition to determine if the distance between the two graphs is smaller.) Obviously, the threshold distance can either be specified via the distance δ_t itself or indirectly by specifying the number of nodes c required as a minimum size of the mcs. From this observation we can conclude that a quick estimate based on feature vector comparison between two graphs can be made on the distance without the actual computationally expensive calculation of the mcs.

3 Decision Trees

In this section, we will show how the concept of feature-vector comparison for maximum common subgraph estimation can be used in combination with decision trees. The idea is that, in a preprocessing step, the feature vectors are extracted from the graph database. Then, a decision tree is built to classify graphs according to their feature vectors. At runtime, all the features needed are extracted from the sample graph and then the decision tree is traversed to retrieve suitable graph candidates from the database. The decision tree approach optimizes the number of features tested in order to eliminate the maximum number of non-candidates from the database.

The decision tree induction procedure itself is analogous to standard decision tree methods (see [16], for example) with the difference that, whereas ordinary

decision tree methods try to generalize from a training set of objects, the approach presented here tries to 'overfit' the data in the sense that all leaf nodes in the tree represent just a single graph, in the ideal case. In general, the smaller the number of graphs in a leaf node is, the smaller is the number of full-fledged graph matchings that need to be computed. Decision tree traversal however needs to be modified such that several branches are followed concurrently while keeping track of the assignable as well as the non-assignable feature values. A detailed explanation will be given below.

In Section 3.1 a brief explanation will be given on how to induce decision trees useful for error-tolerant graph matching. A detailed description on how the tree structure can be used for error-tolerant filtering follows in Section 3.2.

3.1 Error-Tolerant Decision Tree Induction

The tree induction procedure is straightforward. First, all graphs in the database are represented as feature vectors. Then, the decision tree induction algorithm, which is derived from [16], classifies the graphs in the database based on their feature values. As an initializing step, the tree's root node is constructed and it is assigned the entire graph set of the database. Then, each available feature is tested and its suitability is evaluated according to a given split criterion (see [8, 16]). Amongst all features, the best one is chosen and the current root's graph set is split into subsets according to the best feature. For each feature value, a child node is created and the node is assigned the subset of graphs that correspond to that feature value. The induction procedure is recursively continued with the child nodes until one of the following termination conditions holds: a) the graph set in a node contains only one graph; b) no features are left to divide a subset; c) the features left cannot distinguish the remaining graphs in the set. In the experimental section, we will refer to this type of tree structure as the *simple tree structure* (see Section 4).

During traversal, an input sample may contain additional information not available in the graph database. Hence, features not capable of distinguishing database graphs may be useful filtering unknown sample graphs. To account for this case, an extended tree structure is introduced. After the induction of the simple tree structure as describe above, the remaining features are induced even though they do not contain any split information for the database graphs. This guarantees that these features are tested during traversal and the maximum information given in database and sample graph is evaluated. This tree structure will be referred to as the *extensive tree structure* (see Section 4).

3.2 Decision Tree Traversal

Decision trees induced as described above can be used to retrieve possible graph candidates differing no more than a specified threshold distance δ_t from the input sample graph. Assume all graphs in the database are equal in size. (If they are not, this can easily achieved by an initial feature test in the decision tree.) Furthermore, assume the size c of the requested mcs as described in Section 2 is known and all features of the specified type have been extracted from the input

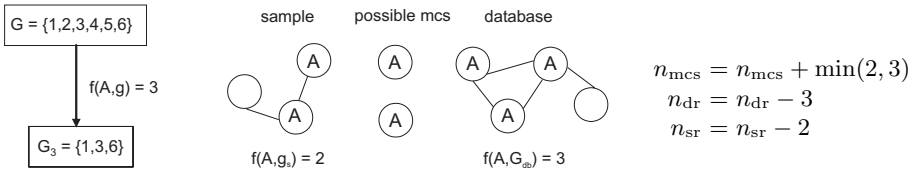


Fig. 2. Illustration of a traversal step and corresponding counter udpates

graph. The general idea is to keep track of the number of nodes assigned to a possible mcs belonging to the traversed tree branch. In order to do this, three counters are needed:

- n_{mcs}: The number of nodes assigned to a possible mcs in the current branch. This counter is initialized with zero (i.e. at the beginning of tree traversal, no nodes are assigned to the mcs).
- n_{dr}: The number of nodes remaining in the database graphs of the current branch. This counter is initialized with the size of the graphs in the database (i.e. all nodes remain to be assigned to a possible mcs).
- n_{sr}: The number of nodes remaining in the sample graph in the current branch. This counters is initialized with the size of the sample graph, analogously to n_{dr}.

The traversal algorithm follows all possible tree branches concurrently. During the traversal of a branch, feature values are compared between database and input sample. Suppose the feature tested at a specific branch is denoted by $f(a, g)$, the input sample is denoted by g_s and similarly, the branch's graph set by G_{db}. The counters are updated in the following way: $n_{\text{mcs}} = n_{\text{mcs}} + \min(f(a, g_s), f(a, G_{db}))$, $n_{\text{dr}} = n_{\text{dr}} - f(a, G_{db})$ and $n_{\text{sr}} = n_{\text{sr}} - f(a, g_s)$.

Figure 2 illustrates a traversal step and the corresponding counter assignments. The feature tested on the illustrated branch is the number of nodes assigned a label 'A'. The sample contains two such nodes whereas the database graphs contain three nodes. It is clear to see that two nodes may be assigned to a possible mcs, hence, n_{mcs} can be incremented by two. Also, three nodes have been evaluated for the database graphs and can be removed from the nodes remaining to be assigned and similarly two nodes can be removed from the remaining nodes of the sample graph. This procedure is recursively continued until one of the following terminating conditions is met: 1) *failure*: $(n_{\text{mcs}}+n_{\text{dr}} < c) \vee (n_{\text{mcs}}+n_{\text{sr}} < c)$ (i.e. not enough nodes can be assigned to a possible mcs); 2) *success*: $n_{\text{mcs}} \geq c$ (i.e. enough nodes have been assigned); 3) *success*: a leaf node has been reached (no other terminating condition occurred). All nodes reached with a successful termination condition are assigned to the overall result set. The graphs contained in these nodes are possible candidates and as a consequence need to be tested with a standard graph distance measure.

4 Experimental Results

To demonstrate the efficiency and feasibility of the approach, we tested it on several different types of graphs (see [17]). The experiments were conducted on two sets of databases. The first set was a collection of small databases (100 graphs per database) of random graphs of varying graph size and label alphabet. These datasets were used to determine the general behavior of the method. The second set was a collection of graph databases of two graph types, namely random graphs and mesh/hyper-cuboid graphs. The random graph database consisted of connected graphs of varying size whereas the mesh database consisted of hyper-cuboids of varying dimension n ($n = 2, 3, 4, 5$). For both graph types different node/edge label alphabet sizes were used. Overall, for each of the above parameter settings a database of 1,000 graphs was created in the second set of experiments. During creation of the database, it was made sure that each graph is isomorphic only to itself.

The primary objective of the proposed filtering method is to reduce the number of candidates which have to undergo a full-fledged graph-distance calculation. Hence, the quality of the approach can be expressed by measuring the number of graphs returned by the traversal method. (We will also refer to this number as the *cluster size* of the decision tree.) The cluster size determines the number of full-fledged distance calculations that need to be executed. The other important measure is the average number of nodes visited during traversal (which is equivalent to the number of tests made during traversal). This value directly affects traversal and therefore filter time.

In order to measure the average cluster size, graphs were randomly picked from the database and were then used as an input sample for the decision tree filter. This procedure was repeated for various allowed distances, ranging from 0.0 to 0.2, which is enough to determine the general behavior of the proposed method. In order to get an average value for the cluster size, the procedure was repeated 100 times for each database and distance setting. First, we will discuss the results obtained on the smaller databases before considering the larger ones.

Figure 3 shows the cluster size for random graphs (20 nodes in size) obtained with the simple as well as the extensive tree structure. (The results for the other graphs sizes are similar to the values illustrated here and are therefore not depicted.) For both approaches it can be seen that, as expected, the cluster size increases with an increasing threshold distance. For a maximum allowed distance of 0.0, the approach constantly returns just one graph from the database. (Since in each database every graph is only isomorphic to itself, this suggests that the method is very well suited for graph isomorphism filtering.) For the simple tree structure, it can be seen that cluster size increases with the number of labels in the label alphabet. This is also expected. With a larger label-alphabet and the labels being uniformly distributed over the graphs in the database, there are overall fewer nodes with equal labels between the graphs. Since the trees induced have a low tree depth, only a limited number of features can be tested in a branch. As a consequence, although n_{mcs} is low, there are enough nodes remaining in the database as well as in the sample to reach a leaf node. (This

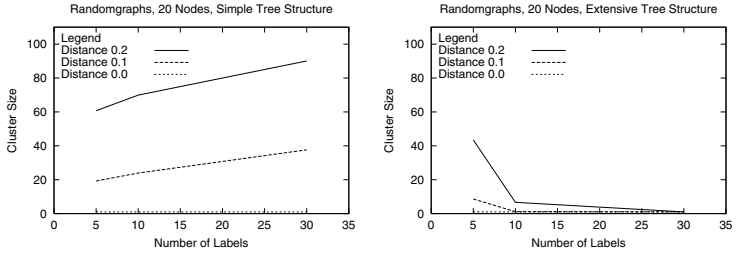


Fig. 3. Cluster size for simple and extensive tree

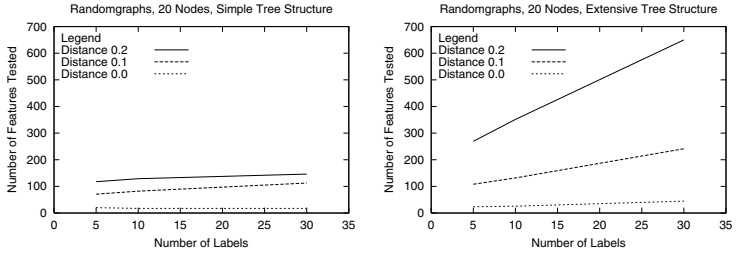


Fig. 4. Average number of visited nodes for simple and extensive tree

means that none of the explicit failure- or success-conditions (conditions 1 and 2) are met during tree traversal. Success is merely achieved by fulfilling condition 3, reaching a leaf node before anything else happens.) Therefore, the graphs in the leaf reached are valid candidates and added to the result set.

To minimize cluster size, the tree structure implementing extensive feature testing has been introduced. The cluster sizes for this traversal type are illustrated on the right of Figure 3. The nodes not accounted for during regular tree traversal are now being consulted and serve to further decrease the cluster size. It is clear to see that with an increasing label-size the graphs contain more information and therefore are much better distinguishable. It can be seen that for a threshold $\delta_t = 0.1$, the cluster size becomes 1 once the label-alphabet is of size 10. This means that finally only a single graph distance needs to be computed. Similarly for $\delta_t = 0.2$ and a label-alphabet of size 10, on average only 4.7 graph-distance computations need to be performed. If the size of the label alphabet is further increased, even less distance evaluations need to be done.

In addition to cluster size, the number of tests made during traversal influences the filtering performance. Since in our approach each test is assigned a node in the tree, this number is reflected in the number of nodes visited during traversal. An illustration for the same databases considered before is given in Figure 4. On the left of Figure 4 the simple approach is illustrated. It can be seen that with an increasing alphabet size the number of features tested (slowly) increases. This can be explained by the fact that with more non-overlapping labels (see above), more features are needed to split the graph set. Hence, the

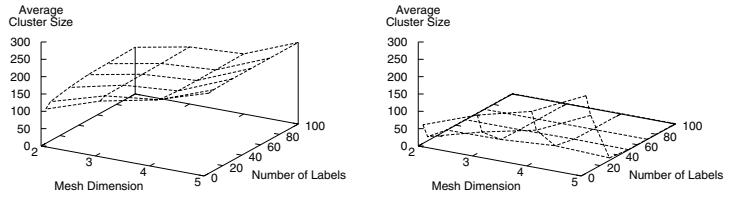


Fig. 5. Cluster size for simple (left) and extensive (right) tree on mesh graphs of varying dimension for a threshold distance of 0.2

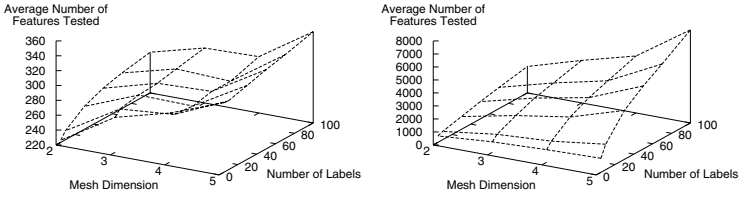


Fig. 6. Average number of features tested for simple (left) and extensive (right) tree on mesh graphs of varying dimension for a threshold distance of 0.2

depth of the tree increases which results in more features being tested during traversal. The right side illustrates the number of tests made during extensive traversal. Clearly, more tests need to be made when the extensive tree structure is adopted. However, considering the significant reduction in cluster size, this additional effort gets easily compensated.

In the second set of experiments we have verified the results found in the first set on a larger variety of graph types and larger databases. Figure 5 shows the results concerning cluster size on mesh graphs. It is clear to see that the extensive tree structure significantly outperforms the simple approach. The simple tree structure does not induce enough features to stop traversal within the decision tree, hence its cluster size is larger than the cluster sizes obtained by the extensive tree structure. The extensive method behaves especially well if the size of the label alphabet is increased and thus the graphs contain more information. The results for the random graph database are similar. Due to increased diversity in the individual graphs in the random graphs database, tree depth is generally lower which results in a larger cluster size when the simple tree structure is traversed. The extensive tree structure on the other hand benefits from the increased diversity which means that the cluster sizes returned are similar to the cluster sizes for mesh graphs.

In Figure 6 it can be seen that the performance gain of the extensive method in cluster size comes at the price of an increased number of features tested during traversal. The additional number of features tested during extensive tree struc-

ture traversal seems quite significant at first sight. However, considering the high computational cost of graph distance computation, there is still a considerable performance gain by further ruling out database graphs, as shown in the right part of Figure 5.

5 Conclusions

In this paper an approach to graph database filtering using machine learning techniques has been presented. The approach addresses the problem of eliminating all graphs g_{db} from a database whose distance $\delta(g_s, g_{db})$ to an input graph g_s is larger than a specified threshold distance δ_t. The method is based on a decision tree data structure. It reduces the number of candidate graphs in a database to be tested by a graph distance measure. Based on simple graph features, a decision tree is built that classifies the graphs. At runtime, possible matching candidates are retrieved from the database by traversing the decision tree.

Considering the complexity of the proposed method we notice that the method is divided into two stages: a) tree induction and b) tree traversal. Tree induction is considered to be an off-line step, hence its complexity is not of primary interest. Tree traversal, however, is the main objective concerning complexity. It is only dependent upon the number of nodes visited during tree traversal. The overall complexity of filtering-based database retrieval is determined by the decision tree traversal complexity, the number of final matchings to be performed (this number is identical to the cluster size) and the complexity of computing the final matchings, which is depending upon graph type and graph distance algorithm. In this paper, a number of experiments investigating cluster size and number of visited nodes have been conducted. The results indicate that the cluster size can be significantly reduced by few tests, resulting in a small number of nodes to be visited as well as a small number of final matchings to be performed.

The main contribution of the present paper is an extension of the methods proposed in [8, 9] from graph and subgraph isomorphism to the problem of error-tolerant graph retrieval. Future work is planned on extending the approach to a wider variety of features, further study of the tradeoff between number of features tested and cluster size as well as determining the method's efficiency on real world data.

Acknowledgment

This research was supported by the Swiss National Science Foundation (Nr. 2100-066700). The authors thank the Foundation for the support.

References

1. Llados, J., Marti, E., Villanueva, J.: Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 23 – 10. (2001) 1137 – 1143

2. Luo, B., Hancock, E.: Structural graph matching using the em algorithm and singular value decomposition. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 23 – 10. (2001) 1120 – 1136
3. Chen, H., Lin, H., Liu, T.: Multi-object tracking using dynamical graph matching. In: Proc. of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2001) 210 – 217
4. Guigno, R., Sasha, D.: Graphgrep: A fast and universal method for querying graphs. In: Proceeding of the International Conference in Pattern Recognition (ICPR). (2002) 112 – 115
5. Messmer, B., Bunke, H.: A new algorithm for error–tolerant subgraph isomorphism detection. In: IEEE Trans. Pattern Analysis and Machine Intelligence. Volume 20. (1998) 493 – 505
6. Shapiro, L., Haralick, R.: Organization of relational models for scene analysis. In: IEEE Trans. Pattern Analysis and Machine Intelligence. Volume 3. (1982) 595 – 602
7. Sengupta, K., Boyer, K.: Organizing large structural modelbases. IEEE Trans. Pattern Analysis and Machine Intelligence **17** (1995) 321–332
8. Irniger, C., Bunke, H.: Graph matching: Filtering large databases of graphs using decision trees. In Jolion, J.M., Kropatsch, W., Vento, M., eds.: Graph-based Representations in Pattern Recognition, Cuen (2001) 239 – 249
9. Irniger, C., Bunke, H.: Graph database filtering using decision tress. In: Proc. 17th Int. Conference on Pattern Recognition. Volume III. (2004) 383 – 388
10. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters **19, Nos 3 - 4** (1998) 255 – 259
11. Sanfeliu, A., Fu, K.: A distance measure between attributed relational graphs for pattern recognition. In: IEEE Trans. Systems, Man, and Cybernetics. Volume 13. (1983) 353 – 363
12. Shapiro, L., Haralick, R.: A metric for comparing relational descriptions. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 7. (1985) 90 – 94
13. Torsello, A., Hidovic, D., Pelillo, M.: Four metrics for efficiently comparing attributed trees. In: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 2, IEEE Computer Society (2004) 467–470
14. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters **18** (1997) 689 – 694
15. Bunke, H.: Error correcting graph matching: On the influence of the underlying cost function. IEEE Trans. Pattern Analysis and Machine Intelligence **21** (1999) 917 – 922
16. Quinlan, J.: C4.5: Programs for Machine Learning. Document Analysis Systems II. Morgan Kaufmann Publishers (1993)
17. Foggia, P., Sansone, C., Vento, M.: A database of graphs for isomorphism and subgraph isomorphism. In Jolion, J.M., Kropatsch, W., Vento, M., eds.: Graph-based Representations in Pattern Recognition, Cuen (2001) 176 – 188

Recovery of Missing Information in Graph Sequences

Horst Bunke[1], Peter Dickinson[2], and Miro Kraetzl[2]

[1] Institut für Informatik und angewandte Mathematik,
Universität Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland
bunke@iam.unibe.ch
[2] Intelligence, Surveillance and Reconnaissance Division,
Defence Science and Technology Organisation,
Edinburgh SA 5111, Australia
{Peter.Dickinson, Miro.Kraetzl}@dsto.defence.gov.au

Abstract. Novel algorithms for the analysis of graph sequences are proposed in this paper. In particular, we study the problem of recovering missing information and predicting the occurrence of nodes and edges in time series of graphs. Our work is motivated by applications in computer network analysis. However, the proposed recovery and prediction schemes are generic and can be applied in other domains as well.

1 Introduction

The aim of graph matching is to find an assignment of the nodes and edges of two given graphs such that some optimality criterion is satisfied. Special instances of the graph matching problem allow us to compute, among other quantities, distance measures between two given graphs. A large body of work on graph matching, dealing with both theory and applications, has been published in the literature. For representative collections of recent work see [1, 2, 3, 4]. However, almost all papers address the case of only two graphs being matched with each other. In this paper, we provide an extension and address the analysis of graph sequences.

The work described in this paper is motivated by applications in computer network monitoring. The basic idea is to represent a computer network by a graph, where the clients and servers are modelled by nodes and the physical connections correspond to edges. If the state of the network is captured at regular points in time and represented as a graph, a time series of graphs is obtained that formally represents the network. In our previous work we have developed various procedures for the detection of anomalous events and network behaviour [5]. These procedures are based on the observation that abnormal network behaviour corresponds to large distance between two consecutive graphs in a time series. In the current paper we address a different problem, viz. the recovery of incomplete network knowledge. Due to various reasons it may happen that the state of a network node or a network link can't be properly captured during network

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 312–321, 2005.

monitoring. This means that it is not known whether a certain node or edge is present in the graph sequence at a certain point in time. In this paper we describe procedures that are able to recover missing information of this kind. These procedures are capable of making a decision as to the presence or absence of such network nodes and edges. Information recovery procedures of this kind can also be used to predict, at time t, whether a certain computer or a certain link will be present, i.e. active, in the network at the next point in time, $t+1$. Such procedures are useful in computer network monitoring in situations where one or more network probes have failed. Here the presence, or absence, of certain nodes and edges is not known. In these instances, the network management system would be unable to compute an accurate measurement of network change. The techniques described in this paper can be used to determine the likely status of this missing data and hence reduce false alarms of abnormal change.

Analysis of time series and prediction is a field that has been intensively studied in the literature [6, 7, 8, 9]. The novel contribution of the current paper is the introduction of a prediction scheme that operates on sequences of graphs rather than symbols, numbers, or vectors.

The paper is organized as follows. Basic terminology and notation will be introduced in the next section. Then, in Sections 3 and 4, we will describe two novel information recovery and prediction procedures. Results of a number of experiments with these new schemes will be presented in Section 5. Finally, conclusions will be drawn in Section 6.

2 Basic Concepts and Notation

A labeled graph is a 4-tuple, $g = (V, E, \alpha, \beta)$, where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, $\alpha : V \rightarrow L$ is the node labeling function, and $\beta : E \rightarrow L^{'}$ is the edge labeling function, with L and $L^{'}$ being the set of node and edge labels, respectively. In this paper we focus our attention on a special class of graphs that are characterized by unique node labels. That is, for any two nodes, $x, y \in V$, if $x \neq y$ then $\alpha(x) \neq \alpha(y)$. Properties of this class of graphs have been studied in [10, 11]. In particular it has been shown that problems such as graph isomorphism, subgraph isomorphism, maximum common subgraph, and graph edit distance computation can be solved in time that is only quadratic in the number of nodes of the larger of the two graphs involved.

To represent graphs with unique node labels in a convenient way, we drop set V and define each node in terms of its unique label. Hence a graph with unique node labels can be represented by a 3-tuple, $g = (L, E, \beta)$ where L is the set of node labels occurring in g, $E \subseteq L \times L$ is the set of edges, and $\beta : E \rightarrow L^{'}$ is the edge labeling function [10, 11]. The terms "node label" and "node" will be used synonymously in the remainder of this paper.

In this paper we will consider time series of graphs, i.e. graph sequences, $s = g_1, g_2, \ldots, g_N$. The notation $g_i = (L_i, E_i, \beta_i)$ will be used to represent individual

graph g_i in sequence s; $i = 1, \ldots, N$. Motivated by the computer network analysis application considered in this paper, we assume the existence of a universal set of node labels, or nodes, $\mathcal{L}$, from which all node labels that occur in a sequence s are drawn. That is, $L_i \subseteq \mathcal{L}$ for $i = 1, \ldots, N$ and $\mathcal{L} = \bigcup_{i=1}^{N} L_i$.[1]

Given a time series of graphs, $s = g_1, g_2, \ldots, g_N$, and its corresponding universal set of node labels, $\mathcal{L}$, we can represent each graph, $g_i = (L_i, E_i, \beta_i)$, in this series as a 3-tuple $(\gamma_i, \delta_i, \widehat{\beta}_i)$ where

- $\gamma_i : \mathcal{L} \rightarrow \{0, 1\}$ is a mapping that indicates whether node l is present in g_i or not. If l is present in g_i, then $\gamma_i(l) = 1$; otherwise $\gamma_i(l) = 0$.[2]
- $\delta_i : \mathcal{L}' \times \mathcal{L}' \rightarrow \{0, 1\}$ is a mapping that indicates whether edge (l_1, l_2) is present in g_i or not; here we choose $\mathcal{L}' = \{l \mid \gamma_i(l) = 1\}$, i.e. $\mathcal{L}'$ is the set of nodes that are actually present in g_i.
- $\widehat{\beta}_i : \mathcal{L}' \times \mathcal{L}' \rightarrow L'$ is a mapping that is defined as follows:
  $\widehat{\beta}_i(e) = \begin{cases} \beta_i(e), & \text{if } e \in \{(l_1, l_2) \mid \delta_i(l_1, l_2) = 1\} \\ \text{undefined}, & \text{otherwise} \end{cases}$

The definition of $\widehat{\beta}_i(e)$ means that each edge e that is present in g_i will have label $\beta_i(e)$. The 3-tuple $(\gamma_i, \delta_i, \widehat{\beta}_i)$ that is constructed from $g_i = (L_i, E_i, \beta_i)$ will be called the *characteristic representation* of g_i, and denoted by $\chi(g_i)$. Clearly, for any given graph sequence $s = g_1, g_2, \ldots, g_N$ the corresponding sequence $\chi(s) = \chi(g_1), \chi(g_2), \ldots, \chi(g_N)$ can be easily constructed and is uniquely defined. Conversely, given $\chi(s) = \chi(g_1), \chi(g_2), \ldots, \chi(g_N)$ we can uniquely reconstruct $s = g_1, g_2, \ldots, g_N$.

In the current paper we'll pay particular attention to graph sequences with missing information. There are two possible cases of interest. First it may not be known whether node l is present in graph g_i or not. In other words, in $\chi(g_i)$ it is not known whether $\gamma_i(l) = 1$ or $\gamma_i(l) = 0$. Secondly, it may not be known whether edge (l_1, l_2) is present in g_i, which is equivalent to not knowing, in $\chi(g_i)$, whether $\delta_i(l_1, l_2) = 1$ or $\delta_i(l_1, l_2) = 0$. To cope with the problem of missing information and in order to make our notation more convenient, we extend functions γ and δ in the characteristic representation, $\chi(g)$, of graph $g = (L, E, \beta)$ by including the special symbol ? in the range of values of each function to indicate the case of missing information. That is, we write $\gamma(l) =?$ if it is unknown whether node l is present in g or not. Similarly, the notation $\delta(l_1, l_2) =?$ will be used to indicate that it is not known whether edge (l_1, l_2) is present in g or not. As any existing edge, $(l_1, l_2) \in E$, requires the existence of both incident nodes, $l_1, l_2 \in L$, we always assume $\gamma(l_1) = \gamma(l_2) = 1$ if $\delta(l_1, l_2) = 1$ to ensure the consistency of our graph representation.

[1] In the computer network analysis application $\mathcal{L}$ will be, for example, the set of all unique IP host addresses in the network. Note that in one particular graph, g_i, usually only a subset is actually present. In general, $\mathcal{L}$ may be any finite or infinite set.

[2] One can easily verify that $\{l \mid \gamma_i(l) = 1\} = L_i$.

3 Recovery of Missing Information Using a Voting Procedure

Consider graph sequence $s = g_1, g_2, \ldots, g_t$ and let $\mathcal{L}$ denote the underlying universal set of node labels. Furthermore, consider graph $g_t = (L_t, E_t, \beta_t)$ with characteristic representation $\chi(g_t) = (\gamma_t, \delta_t, \widehat{\beta}_t)$ and assume that $\gamma_t(l) =?$ for some $l \in \mathcal{L}$. The task to be solved is to make a decision as to $\gamma_t(l) = 1$ or $\gamma_t(l) = 0$. From the formal point of view this problem is cast as a classification problem in this paper. That is, given $\gamma_t(l) =?$ we want to assign l to one of two classes, Ω_0 or Ω_1. Class Ω_1 is equivalent to saying that l is present in g_t, i.e. $\gamma_t(l) = 1$, while class Ω_0 means that l is not included in g_t, i.e. $\gamma_t(l) = 0$.

To classify l as either belonging to class Ω_0 or Ω_1, we consider a subsequence, or time window, $s' = g_{t-M}, \ldots, g_{t-1}$ of length M. The basic idea is to utilize information about node l in the graphs belonging to subsequence s', in order to decide about the presence or absence of l in graph g_t. A simple approach consists in computing the relative frequency of occurrence of node l in subsequence s' and using this value for the decision to be made. Let k_1 be the number of graphs in subsequence s' in which node l is actually present, i.e. $\gamma(l) = 1$. Similarly, assume that k_0 is the number of graphs in subsequence s' for which $\gamma(l) = 0$. Obviously, there are $0 \leq M - (k_0 + k_1) \leq M$ graphs in subsequence s' where $\gamma(l) =?$. Given parameters k_0 and k_1, we can use the following simple rule to make a decision as to the presence of node l in g_t:

$$\gamma_t(l) = \begin{cases} 0 \text{ if } k_0 > k_1 \\ 1 \text{ if } k_1 > k_0 \end{cases} \tag{3.1}$$

In case $k_0 = k_1$, a random decision is in order. A potential problem with the decision rule according to (3.1) occurs if $k_0 + k_1 = 0$, i.e. if in each graph, g, of subsequence s', we have $\gamma_t(l) =?$. In this case one has to resort to making a random decision, or possibly enlarge the length of subsequence s', i.e. increase the value of parameter M. The second possibility is motivated by the expectation to find graphs in subsequence $g_1, \ldots, g_{t-M-1}$ in which $\gamma(l) = 1$ or $\gamma(l) = 0$, resulting in a value $k_0 > 0$ and/or $k_1 > 0$.

Similar decision procedures can be derived for edges e in graph g_t. That is, given edge e with $\delta(e) =?$, we count the number of graphs, g, in subsequence s' with $\delta(e) = 0$ and the number of graphs, g, with $\delta(e) = 1$ and decide, based on these two numbers, whether $\delta_t(e) = 0$ or $\delta_t(e) = 1$.

In case information about more than one node, l, or one edge, e, is missing in graph g_t, we can apply the procedures described above to all affected nodes and edges of g_t in parallel. In the extreme case, these procedures can even be applied when information about the complete graph, g_t, is missing, i.e. when $\gamma_t(l) =?$ for all nodes, l, of g_t and $\delta_t(e) =?$ for all edges, e, of g_t.

4 Recovery of Missing Information Using Reference Patterns

The method described in Section 3 is based on a simple voting scheme that computes the number of graphs in subsequence s' where $\gamma(l) = 0$ and compares this number with the number of graphs where $\gamma(l) = 1$. The particular order of the values $\gamma(l) = 0$ and $\gamma(l) = 1$ in subsequence s' is not relevant. In Section 4 we develop a more refined decision rule where this order is taken into account. We assume the existence of a reference set, R, of graph subsequences of length M, i.e. $R = \{s_1, \ldots, s_n\}$ where $s_j = g_{j,1}, \ldots, g_{j,M}$ for $j = 1, \ldots, n$. Each element, s_j, of the reference set is a sequence of graphs of length M. These sequences are used to represent information about the "typical behaviour" of the nodes and edges in a graph sequence of length M. This information will be used to make a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$ whenever $\gamma_t(l) =?$ occurs.

To generate reference set R, we can utilize graph sequence $g_1, \ldots, g_{t-1}$. Each sequence in R is of length M, by definition. Let's assume that $M \leq t-1$. Then we can extract all subsequences of length M from sequence $g_1, \ldots, g_{t-1}$, and include them in reference set R. This results in

$$R = \{s_1 = g_1, \ldots, g_M \ ; \ s_2 = g_2, \ldots, g_{M+1} \ ; \ s_{t-M} = g_{t-M}, \ldots, g_{t-1}\}$$

From each sequence, $s_i = g_i, \ldots, g_{i+M-1}$, in set R we can furthermore extract, for each node $l \in \mathcal{L}$, the sequence $\gamma_i(l), \ldots, \gamma_{i+M-1}(l)$. Assume for the moment that $\gamma_i(l), \ldots, \gamma_{i+M-1}(l) \in \{0,1\}$, which means that none of the elements $\gamma_i(l), \ldots, \gamma_{i+M-1}(l)$ is equal to ?. Then $(\gamma_i(l), \ldots, \ \gamma_{i+M-1}(l))$ is a sequence of binary numbers, 0 or 1, that indicate whether or not node l occurs in a particular graph in sequence s_i. Such a sequence of binary numbers will be called a *reference pattern*. Obviously $(\gamma_i(l), \ldots, \gamma_{i+M-1}(l)) \in \{0,1\}^M$. Because there are 2^M different binary sequences of length M, there exist at most 2^M different reference patterns for each node $l \in \mathcal{L}$. Note that a particular reference pattern, $x = (x_1, \ldots, x_M) \in \{0,1\}^M$, may have multiple occurrences in set R.

In order to make a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, given $\gamma_t(l) =?$, the following procedure can be adopted. First, we extract from graph sequence $s = g_1, \ldots, g_t$ the sequence $(\gamma_{t-M+1}(l), \ldots, \gamma_t(l))$ where, according to our assumption, $\gamma_t(l) =?$. Assume furthermore that $\gamma_{t-M+1}(l), \ldots, \gamma_t(l) \in \{0,1\}$, i.e. none of the elements in sequence $(\gamma_{t-M+1}(l), \ldots, \gamma_t(l))$, except $\gamma_t(l)$, is equal to ?. Sequence $(\gamma_{t-M+1}(l), \ldots, \gamma_t(l))$ will be called the *query pattern*. Given the query pattern, we retrieve from the reference set, R, all reference patterns $x = (x_1, \ldots, x_M)$ where $x_1 = \gamma_{t-M+1}(l), x_2 = \gamma_{t-M+2}(l), \ldots, x_{M-1} = \gamma_{t-1}(l)$. Any reference pattern, x, with this property is called a *matching reference pattern*. Clearly, a reference pattern that matches the query pattern is a sequence of 0's and 1's of length M, where the first $M-1$ elements are identical to corresponding elements in the query pattern. The last element in the query pattern is equal to ?, by definition, while the last element in any matching reference pattern is either 0 or 1. Let k be the number of reference patterns that match the query pattern. Furthermore, let k_0 be the number of matching reference patterns with $x_M = 0$, and k_1 be the number of matching reference

patterns with $x_M = 1$; note that $k = k_0 + k_1$. Now we can apply decision rule (3.1) again. Intuitively, under this decision rule we consider the history of node l over a time window of length M and retrieve all cases recorded in set R that match the current history. Then a decision is made as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, depending on which case occurs more frequently in the reference set.

The method described above is based on the assumption that none of the reference patterns for node l, extracted from set R, contains the symbol ?. For a generalization where this restriction is no longer imposed, neither in the reference nor in the query patterns, see [12].

5 Experimental Results

The methods described in Sections 3 and 4 of this paper have been implemented and experimentally evaluated on graph sequences derived from real computer networks. For the experiments four time series of graphs, S_1, S_2, S_3 and S_4, acquired from two computer networks have been used. Some quantitative features of these graph sequences are shown in Table 1, where the size of a graph is defined as the number of its nodes. All four series represent logical communications on the network. Series S_1, S_2 and S_4 were derived from data collected from a large enterprise data network, while S_3 was collected from a wireless LAN used by delegates during the World Congress for Information Technology (WCIT2002). The nodes in each graph of S_1 and S_2 represent business domains in the network, while in S_3 and S_4 they represent individual IP addresses.

Note that all graph sequences are complete, i.e. there are no missing nodes and edges in these sequences.

To test the ability of the methods described in Sections 3 and 4 it was assumed, for a particular graph in a time series, that $\gamma(l)$ is unknown for each node. Then the two prediction scheme were applied and the percentage of correctly predicted nodes in the considered graph was determined. This procedure was repeated for each graph in the time series. Note that both methods work with a time window of length M. Hence prediction starts only at the $(M+1)$-th graph in a sequence. In Fig. 1 the percentage of correctly predicted nodes for each graph of sequence S_1 is shown. Because of space limitations, only the results obtained for sequence S_1 are depicted in this paper. But the results for the other three sequences are very similar. All experiments were also executed for predicting the edges, rather than the nodes, in the graphs. Again the results were similar to Fig. 1.

Table 1. Characterisation of the graph sequences used in the experiments

	S_1	S_2	S_3	S_4
Number of graphs in sequence	102	292	202	99
Size of smallest graph in sequence	38	85	15	572
Size of largest graph in sequence	94	154	329	10704
Average size of graphs in sequence	69.7	118.5	103.9	5657.8

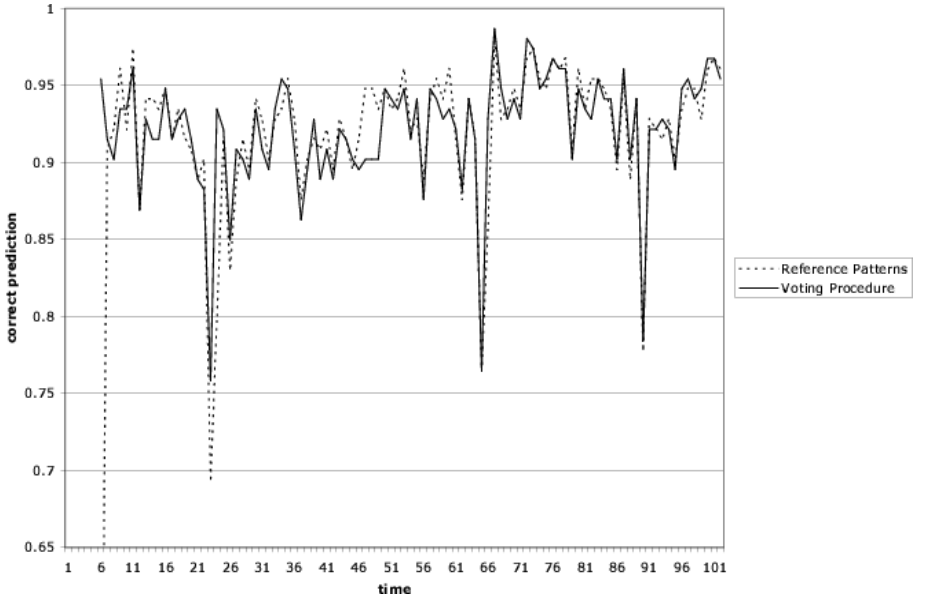


Fig. 1. Percentage of correctly predicted nodes in sequence S_1, using $M = 5$

In Fig. 1 we observe that prediction accuracy for either prediction scheme is over 90% on the average. This is a remarkably high value taking into consideration that for a two-class classification problem, such as the one considered here, random guessing would give us an expected performance of only 50%. There are some obvious drops in prediction performance in Fig. 1, for example at time 22 and at time 65. These drops correspond to abnormal events in the underlying computer network where major changes in network topology take place. In general the topology of the network is changing rather slowly with time. This means that the likelihood of a node or an edge, which is present at time t, being also present at time $t+1$ is quite high. However, abnormal events manifest themselves by sudden change in network topology, i.e. the appearance of many new and/or the disappearance of many existing nodes and edges, which causes the proposed prediction scheme with a higher likelihood to fail. Nevertheless, we note that even at those drop points performance is quite high.

The length, M, of the time window for voting (Section 3) and the reference patterns (Section 4) is a parameter to be selected by the user. In Fig. 1, the value $M = 5$ was used. To study the influence of this parameter on the prediction accuracy, other values of M were experimentally investigated. Fig. 2 shows, for graph sequence S_1, the average node prediction accuracy for values $M = 1, \ldots, 11$. This figure was produced by averaging, for each considered value of M, the prediction accuracy at each individual node, as shown in Fig. 1, over the whole time series of graphs. We notice that the performance of the voting procedure is more or less independent of M, although a slight local maximum can be observed for $M = 5$. The behaviour of the reference pattern scheme is almost

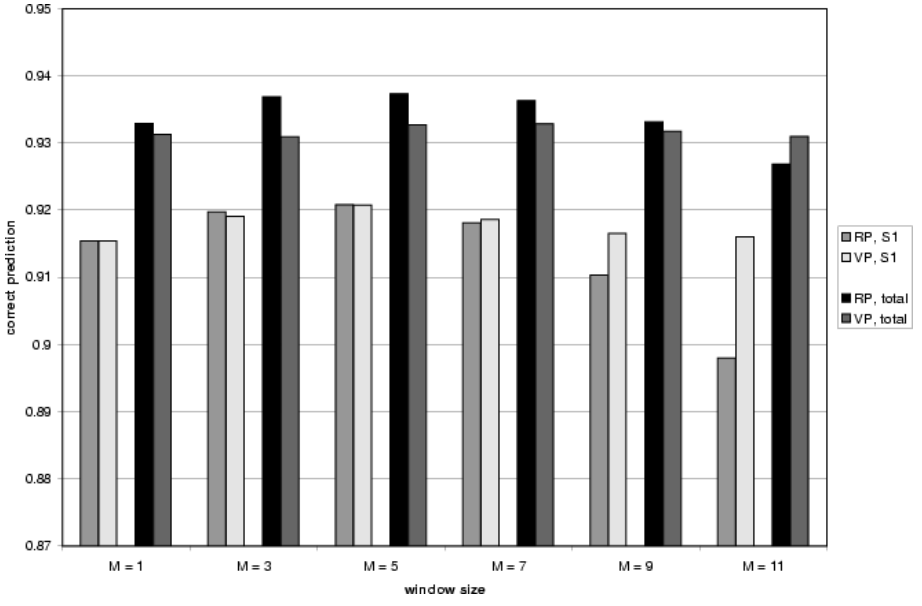


Fig. 2. Percentage of correctly predicted nodes, averaged over whole sequence dependent on parameter M

identical to the voting procedure for $M = 1, \ldots, 7$, but for $M = 9$ and $M = 11$ performance somewhat deteriorates. Also shown in Fig. 2 is the prediction accuracy averaged over all four graph sequences, S_1 to S_4 (total). Except for $M = 11$, the reference pattern scheme has a slightly higher performance than the voting method.

6 Conclusions

The problem of incomplete knowledge recovery and prediction of the behaviour of nodes and edges in time series of graphs was studied in this paper. Formally, this task was formulated as a classification problem where nodes and edges with an unknown status are to be assigned to one of the classes 'present in' or 'absent from' the actual graph. Two procedures for such a classification were proposed. Both use knowledge of the behaviour of a node or an edge in previous graphs of the time series under consideration. The motivation of this work stems from the field of computer network monitoring. However the proposed framework for graph sequence analysis is fairly general and can be applied in other domains as well. In computer network monitoring, prediction procedures, as studied in this paper, are important for patching missing network data in instances where one or more network probes have failed. Without such procedures, the network management system would have diminished capability in detecting abnormal change.

The proposed prediction procedures are conceptually simple and straightforward to implement. Nevertheless they have an excellent performance and achieve correct prediction rates for the nodes and edges of real networks ranging between about 90% and 97%.

The proposed schemes can be extended in a variety of ways. There are several other algorithmic paradigms that seem to be feasible for the considered problem, for example, linear prediction [13]. In a parallel study, the application of decision tree classifiers for predicting the occurrence of nodes and edges is investigated [14]. From the general point of view, the two procedures proposed in the current paper utilize context in time, i.e. knowledge of the past behaviour of a node or an edge is used to predict its future behaviour. As an alternative, one could use within-graph context rather than context in time. Under such a scheme one would base the decision as to the presence or absence of a node or an edge in the network at a particular time on the presence or absence of other nodes or edges at the same time, i.e. in the same graph. Finally, the combination of both schemes, using context in time together with within-graph context seems feasible.

Acknowledgement

The authors are thankful to F. Thalmann, Ch. Irniger and T. Varga for various contributions to this paper.

References

1. IEEE Trans. PAMI. *Special Section on Graph Algorithms and Computer Vision*, Volume 23, 2001.
2. Pattern Recognition Letters. *Special Issue on Graph Based Representation*, Volume 24, Elsevier Science B.V., 2003.
3. Int. Journal of Pattern Recognition and Artificial Intelligence. *Special Issue on Advances in Graph Matching*, Volume 18, No. 3, 2004
4. Hancock, E., Vento, M. (eds.): Graph Based Representations in Pattern Recognition, *Proc. 4th Int. Workshop GBR2003*, Springer, LNCS 2726, 13-23
5. Bunke, H., Kraetzl, M., Shoubridge, P., Wallis, W.: Detection of abnormal change in time series of graphs, *Journal of Interconnection Networks*, Vol. 3, Nos 1,2, 2002, 85-101
6. Weiss, S.M., Indarkhya: Predictive Data Mining: A Practical Guide. Morgan Kaufmann, 1998
7. Fung, G.P.C.F., Yu, d.X., Lam, W.: News sensitive stock trend prediction, in Chen, M.-S., Yu, P.S., Liu, B. (eds.): *Advances in Knowledge Discovery and Data Mining, Proc. 6th Pacific-Asia Conference, PAKDD*, Springer, LNAI 2336, 2002, 481-493
8. Schmidt, R., Gierl, L.: Temporal abstractions and case-based reasoning for medical course data: two prognostic applications, in Perner, P. (ed.): *Machine Learning in Pattern Recognition Proc. 2nd Int. Workshop*, Springer, LNAI 2123, 2001, 23-34
9. Kahveci, T., Singh, K.: Optimizing similarity search for arbitrary length time series queries, IEEE Trans. KDE, Vol 16, No 2, 418-433

10. Dickinson, P., Bunke, H., Dadej, A., Kraetzl, M.: On graphs with unique node labels, in Hancock, E., Vento, M. (eds.): Graph Based Representations in Pattern Recognition, *Proc. 4th Int. Workshop GBR2003*, Springer, LNCS 2726, 13-23
11. Dickinson, P., Bunke, H., Dadej, A., Kraetzl, M.: Matching graphs with unique node labels, accepted for publication in *Pattern Analysis and Applications*
12. Bunke, H., Dickinson, P., Kraetzl, M.: Analysis of Graph Sequences and Applications to Computer Network Monitoring, *Technical Report*, DSTO, Edinburgh, Australia, 2003
13. Makhoul, J.: Linear prediction: a tutorial review, *Proc. of the IEEE*, 63(4), 1975, 561-580
14. Quinland, R.: C4.5: Programs for Machine Learning, Morgen Kaufmann Publ., 1993

Tree-Based Tracking of Temporal Image

Tomoya Sakai, Atsushi Imiya, and Heitoh Zen

Institute of Media and Information Technology, Chiba University,
Yayoi-cho 1-33, Inage-ku, 263-8522, Chiba, Japan
{tsakai, imiya, zen.h}@faculty.chiba-u.jp

Abstract. This paper introduces a tree-based algorithm for the motion tracking of dominant parts in a temporal image sequence. A tree allows us to express hierarchical relations of segments in an image. We first develop a fast tree matching algorithm which is suitable for matching of images. Second, employing the linear-scale-space analysis, we develop an algorithm to extract hierarchical relations of temporal images as a tree. Combining tree matching and tree extraction provides a method to extract moving dominant parts in a sequence of temporal images.

1 Introduction

In the previous paper [13], we introduced hierarchical analysis of temporal images using linear scale space analysis. This paper is a squeal of the previous paper, we show that using the same strategy, it is possible to achieve grouping of image in a temporal sequence. We first show a fast approximate tree matching algorithm [13], which is suitable for time sequence segmentation of temporal images. Since a tree expresses a hierarchical relations in data [13, 14, 16, 15, 10, 11], tree-based expression of segments in an image provides global-hierarchical features for image analysis [13]. For the application of tree-based motion analysis, we are required to develop both fast tree-matching algorithm [14] and tree construction algorithm. Second, we introduce a tree-sequence construction algorithm for a temporal image sequence using linear scale-space analysis [8, 9, 2, 3, 13, 1, 12]. The linear-scale space analysis provides a tree-extraction algorithm from signals [6, 7] and images [2, 3]. We have extended the linear-scale-space-based tree construction method to temporal images [13].

2 Tree Distance and Its Fast Computation

For quantitative analysis of topological changes of a temporal image with temporal trees extracted in the linear scale space, we introduce the distance between trees based on the editing of the tree structure. Since it is possible to transform irregular trees to regular trees by adding special nodes ($*$) to the trees, we assume that our trees are regular. Furthermore, since our trees, extracted using scale space analysis, are rooted trees, we develop a fast computation method for rooted trees which is applicable in time-varying image sequence analysis [16]. Therefore, we assume that trees are m-regular, for $m \geq 2$.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 322–331, 2005.

Setting α to be a k-digit number, each digit of which from 0 to $k-1$, the subtree of the node α is expressed as

$$n_\alpha(T) = t_\alpha[T_{\alpha 1}, T_{\alpha 2}, \cdots, T_{\alpha m}], \tag{1}$$

where t_α is the label of node n_α. For $\alpha = 0$, equation (1) expresses the tree and n_0 is the label of the root. Therefore, the number of digits of α expresses the depth of a subtree.

The operations applied to a tree are the transform of the node label, the permutation of subtrees, the insertion of a subtree to $*$, and the elimination of a subtree, which are mathematically expressed as

$$E_t : t_\alpha[T_{\alpha 1}, \cdots, T_{\alpha m}] = x_\alpha[T_{\alpha 1}, \cdots, T_{\alpha m}], \tag{2}$$

$$E_p : t_\alpha[T_{\alpha 1}, \cdots, T_{\alpha i} \cdots T_{\alpha j} \cdots T_{\alpha m}] = t_\alpha[T_{\alpha 1}, \cdots, T_{\alpha j} \cdots T_{\alpha i} \cdots T_{\alpha m}], \tag{3}$$

$$E_i(S) : t_\alpha[T_{\alpha 1} \cdots * \cdots T_{\alpha m}] = t_\alpha[T_{\alpha 1} \cdots S \cdots T_{\alpha m}], \tag{4}$$

$$E_i(T_{\alpha k}) : t_\alpha[T_{\alpha 1} \cdots T_{\alpha k} \cdots T_{\alpha m}] = t_\alpha[T_{\alpha 1} \cdots * \cdots T_{\alpha m}]. \tag{5}$$

Furthermore, a successive application of E_n derives the transformation of a subtree such as

$$E_t(T_{\alpha,k}, S) : t_\alpha[T_{\alpha 1}, \cdots, T_{\alpha k} \cdots, T_{\alpha m}] = x_\alpha[T_{\alpha 1}, \cdots, S, \cdots, T_{\alpha m}]. \tag{6}$$

We define the lengths of these operations as

$$d(s_\alpha) = \frac{1}{1 + l_k} |s_\alpha|, \tag{7}$$

where

$$l_\alpha = \begin{cases} 0, & \text{if } \alpha = 0, \\ \text{the number of digits of } \alpha, & \text{otherwise}, \end{cases} \tag{8}$$

and

$$|s_\alpha| = \begin{cases} 1, & \text{if } s_\alpha \text{ is the node-label transform}, \\ |P|, & \text{if } s_\alpha \text{ is the permutation}, \\ |S|, & \text{if } s_\alpha \text{ is the insertion}, \\ |S|, & \text{if } s_\alpha \text{ is the elimination}, \end{cases} \tag{9}$$

where $|P|$ is the number of permutations. Using these lengths, we define the distance between trees as

$$D(T, T') = \sum_{\alpha=1}^{n} d(s_\alpha), \tag{10}$$

for the sequence of operations $\{s_1, s_2, \cdots, s_n\}$ which transforms T to T'. This tree distance satisfies the following lemma.

Lemma 1. *For trees of almost the same order, the distance in eq. (10) is metric, that is, it satisfies the conditions of distance.*

3 Linear Scale Space Analysis for Temporal Images

3.1 Scale Space Analysis of Image Sequence

In the two-dimensional Euclidean space $\mathbf{R}^2$, for an orthogonal coordinate system x-y defined in $\mathbf{R}^2$, a vector in $\mathbf{R}^2$ is expressed by $\boldsymbol{x} = (x, y)^\top$, where $\cdot^\top$ is the transpose of a vector. For the temporal function $f(\boldsymbol{x}, t)$, the general function $f(\boldsymbol{x}, \tau, t)$ is the solution of the equation

$$\frac{\partial}{\partial \tau} f(\boldsymbol{x}, \tau, t) = \Delta f(\boldsymbol{x}, \tau, t),\ \tau > 0,\ f(\boldsymbol{x}, 0, t) = f(\boldsymbol{x}, t). \tag{11}$$

The spaces $\mathbf{R}^2 \times (0 \leq \tau < \infty)$ and $\mathbf{R}^2 \times (0 \leq \tau < \infty) \times (-\infty < t < \infty)$ are called the linear scale space and the temporal linear scale space, respectively.

The attention points for the topographical maps $f(\boldsymbol{x}, \tau, t)$ in the temporal scale space are defined as the solutions of the equation $\nabla f(\boldsymbol{x}, \tau, t) = 0$. The stationary curves in the scale space are collections of the attention points. We denote the trajectories of the attention points by $\boldsymbol{x}(\tau, t)$ for each t. Setting $\boldsymbol{H}$ to be the Hessian matrix of $f(\boldsymbol{x}, \tau, t)$, for fixed t, Zhao and Iijima [2] showed that the stationary curves for a function are the solution of

$$\boldsymbol{H} \frac{d\boldsymbol{x}(\tau, t)}{d\tau} = -\nabla \Delta f(\boldsymbol{x}(\tau, t), \tau, t), \tag{12}$$

and clarified the topological properties of the stationary curves for two-dimensional patterns.

This geometrical property guarantees that, as

$$\boldsymbol{x}(t) = \lim_{\tau \to 0} \boldsymbol{x}(\tau, t) \tag{13}$$

the validity of the tracking of the stationary points of the spatio-temporal gradient of the temporal function.

The attention points $\boldsymbol{x}^*(t)$, which satisfy $\nabla f(\boldsymbol{x}, \tau, t) = 0$ expresses the topological structure of image $f(\boldsymbol{x}, \tau, t)$. The signs of the eigenvalues of the Hessian matrix $\boldsymbol{H}$ at attention point $\boldsymbol{x}^*$ clarify the geometrical property of the attention point.

Since the second directional derivation of $f(\boldsymbol{x}, \tau)$ for point $\boldsymbol{x}$ is defined as

$$D^2_{\boldsymbol{x}(,\tau,t)}(\theta) = \frac{d^2}{d\boldsymbol{n}^2} f(\boldsymbol{x}, \tau, t) == \boldsymbol{n} \cdot \nabla (\boldsymbol{n} \cdot \nabla f) = \boldsymbol{n}^\top \boldsymbol{H} \boldsymbol{n}. \tag{14}$$

where $\boldsymbol{n}(\theta) = \boldsymbol{\omega} - \boldsymbol{x}$ for $\boldsymbol{\omega} = (\cos\theta, \sin\theta)^\top$, $0 \leq \theta \leq 2\pi$. $D^2_{\boldsymbol{x}(,\tau,t)}(\theta)$ expresses the geometrical property of the attention point. Equation (14) indicates that the eigenvectors of the Hessian matrix of $f(\boldsymbol{x}, \tau, t)$ give the extrema and saddles of $D^2 f(\boldsymbol{x}, \tau, t)$ and that the extrema and saddles are achieved by the eigenvectors of the Hessian of $f(\boldsymbol{x}, \tau, t)$, since $\alpha_1 \geq \boldsymbol{n}^\top \boldsymbol{H} \boldsymbol{n} \geq \alpha_2$ for $|\boldsymbol{n}| = 1$, where $\alpha_1 \geq \alpha_2$ are two eigenvalues of the 2×2 Hessian matrix $\boldsymbol{H}$. Therefore, we have the relations $\max(D^2_{\boldsymbol{x}(\tau,t)}) = \alpha_1$ and $\min(D^2_{\boldsymbol{x}(\tau,t)}) = \alpha_2$.

In references [2, 3], Zhao and Iijima adopted singular points which satisfy the relation $\frac{d^2}{d\boldsymbol{n}^2} f(\boldsymbol{x}, \tau, t) > 0$ or $\frac{d^2}{d\boldsymbol{n}^2} f(\boldsymbol{x}, \tau, t) < 0$, that is, points which satisfy the relation $\alpha_1 \cdot \alpha_2 > 0$. We denote the signs of the eigenvalues of the minus of the Hessian matrix as $(+,+)$, $(+,-)$ and $(-,-)$, which correspond to the local maximum points, the saddle points, and the local minimum points, respectively. In this paper, we deal with all three classes of extrema. The saddle points in the scale space appear on walls and valleys which connect maximum points and minimum points, respectively. Therefore, the motion of the saddle points in the scale space corresponds to changes in the topology of gray-valued images in the scale space.

3.2 Temporal Tree Construction

Since the stationary curves consist of many curves for $\tau > 0$, we call each curve a branch curve. The point $\boldsymbol{x}_\infty$ for

$$\lim_{\tau \to \infty} \boldsymbol{x}(\tau, t) = \boldsymbol{x}_\infty(t) \tag{15}$$

is uniquely determined for any image. We call a curve on which point $\boldsymbol{x}_\infty$ lies and a curve which is open to the direction of $-\tau$, the trunk and branch, respectively.

Definition 1. *For $S(\boldsymbol{x}, \tau, t) = |\frac{d\boldsymbol{x}(\tau)}{d\tau}|$, the stationary points on the stationary-curves are the points which satisfy $S(\boldsymbol{x}, \tau, t) = 0$ or are isolated points under the conditions*

$$\frac{dS(\boldsymbol{x}, \tau, t)}{d\tau} = 0, \ \frac{d^2 S(\boldsymbol{x}, \tau, t)}{d^2 \tau} = 0. \tag{16}$$

Zhao and Iijima introduced the following procedure for the construction of a tree from an image [2, 3].

Rule 1

1. *The sub-root of a branch is the singular point, such that $\nabla f = 0$, at the top of the branch curve and a sub-root.*
2. *The sub-root is connected to the trunk by a line segment parallel to the x-y plane.*

Definition 2. *For the stationary points on the stationary curves which are merged using* **Rule 1***, the order of the stationary points is defined as*

$$\boldsymbol{x}(\tau, t) \succ \boldsymbol{x}(\tau', t) \ \textit{if} \ \tau > \tau' \ \textit{on a branch for a fixed} \ t. \tag{17}$$

The tree is constructed according to the order of the stationary points on the stationary-curves. Denoting a stationary point on the stationary-curves as $(\boldsymbol{x}_i, \tau_i, t)$, a point $\boldsymbol{x}_i$ and the region of interest $\boldsymbol{R}(\boldsymbol{x}_i \tau_i, t)$

$$\boldsymbol{R}(\boldsymbol{x}_i, \tau_i, t) = \{\boldsymbol{x}(t) | |\boldsymbol{x}(t) - \boldsymbol{x}_i| \leq \sqrt{2\tau_i(t)}\} \tag{18}$$

called the stable attention point and the attention field. Furthermore,

$$f(\boldsymbol{x}, \boldsymbol{x}_i, \tau_i, t) = \exp(-\frac{|\boldsymbol{x} - \boldsymbol{x}_i(t)|^2}{\tau_i}) f(\boldsymbol{x}, t) \tag{19}$$

is called a view-controlled image of the original image, since $f(\boldsymbol{x}, \boldsymbol{x}_i, \tau_i, t)$ approximates an image which is observed by a vision system with a same mechanisms to those of of human beings [1] for the fixed t. **Rule 2** is our rule based on the radii of the attention fields.

Rule 2

1. *On the trunk, if $\tau > \tau'$, we define the order of the stationary points as $\boldsymbol{x}(\tau, t) \succ \boldsymbol{x}(\tau', t)$ for a fixed t.*
2. *On each branch $\boldsymbol{x}_i(\tau, t)$, we express the stationary point $\boldsymbol{x}_i(\tau_{i(j)}, t)$. Assuming that the maximum scale parameter on this branch is $\tau_{i(0)}$, we set $\boldsymbol{x}_{i(j)}(t) = (x_{ij}, y_{ij}, t)^\top$ for point $\boldsymbol{x}_i(\tau_j, t)$ in the scale space.*
3. *We define the order of the stationary points on each branch employing geometric properties of the attention fields.*
   (a) *On each branch curve, for $\tau_{i(m)}(t) > \tau_{i(n)}(t)$, if the relation*

$$|\boldsymbol{x}_{i(m)}(t) - \boldsymbol{x}_{i(n)}(t)| \leq \sqrt{2\tau_{i(m)}(t)}$$

   is satisfied, then we define $\boldsymbol{x}_{i(m)}(t) \succ \boldsymbol{x}_{i(n)}(t)$.
   (b) *For a pair of branch curves $\boldsymbol{x}_i(\tau, t)$ and $\boldsymbol{x}_j(\tau, t)$ and a pair of fixed scales $\tau_{i(m)}(t)$ and $\tau_{j(0)}(t)$, if the relation*

$$|\boldsymbol{x}_{i(m)}(t) - \boldsymbol{x}_{j(0)}(t)| \leq \sqrt{2\tau_{i(m)}(t)}$$

   is satisfied, then we define $\boldsymbol{x}_{i(m)} \succ \boldsymbol{x}_{j(0)}$.

3.3 Transformation from Image to Tree

The property introduced in this section defines the transformation from a temporal image $f(\boldsymbol{x}, t)$ to a temporal tree $T(t)$, such that,

$$\Theta(f(\boldsymbol{x}, t)) = T(t), \ \frac{\partial}{\partial \tau} f(\boldsymbol{x}, \tau, t) = \Delta f(\boldsymbol{x}, \tau, t). \tag{20}$$

For the sampled sequence $f(x, y, 1)$, $f(x, y, 2)$, $\cdots$, $f(x, y, t)$, $f(x, y, t+1)$, $\cdots$, constructing the structure tree $T(t)$ for each image in this sequence, the algorithm yields a sequence of trees from a sequence of images. The transformation from temporal images to trees is algorithmically achieved by the following procedure.

Procedure: Tree Extraction

1. *Compute $f(\boldsymbol{x}, \tau, t)$ from $f(\boldsymbol{x}, t)$ at each t.*
2. *Construct stationary curves applying* **Rule 1** *to the solution of eq. (12).*
3. *Extract stationary points on the stationary curves using* **Definition 1**.
4. *Construct a tree for each t, using* **Rule 2**.

For the generation of a temporal tree sequence, we adopt the following **Rule 3**.

Rule 3

1. *If a pair of successive trees $T(t)$ and $T(t+1)$ is topologically different we affix new labels for nodes, excepting the root.*
2. *For topologically equivalent trees $T(t)$ and $T(t+1)$, if the stationary points of $T(t+1)$ do not remain in the field of view of each node, we consider these two trees to be different and $T(t+1)$ produces new nodes.*
3. *We eliminate old symbols of nodes in $T(t)$ and affix new symbols to new nodes in $T(t+1)$.*

Using these operations, we can extract the motion of stationary points and the trajectory of the attention fields on the image plane $\tau = 0$. This process detects the trajectory of the dominant motion of parts in a sequence of images. Stationary points on the stationary-curves, which are of the centroids of the field of views, are cues for the extraction of the dominant parts of temporal functions. Therefore, by tracking the stationary curves, we can extract the trajectories of the dominant parts of images.

3.4 Optical Flow in Scale Space

Optical flow extracts motion of each point of each image in scale space. In this section, for motion analysis, we compare the deference between optical flow and temporal tree-analysis.

For the total derivation of the general images,

$$\frac{d}{dt}f(\boldsymbol{x},\tau,t) = \nabla f^{\top}\dot{\boldsymbol{x}} + \frac{\partial}{\partial t}f(\boldsymbol{x},\tau,t), \tag{21}$$

where $\dot{\boldsymbol{x}} = \frac{d\boldsymbol{x}}{dt}$ is the optical flow of the temporal function $f(\boldsymbol{x},\tau,t)$, and at the point $\boldsymbol{x}$ [17, 18, 19, 20] we have the relation

$$\frac{d}{dt}f(\boldsymbol{x},\tau,t) = 0. \tag{22}$$

Equations (11) and (22) imply the next property of the flow vectors in the linear scale space.

Theorem 1. *Let $f(\boldsymbol{x},\tau,t)$ be the spatio-temporal general image. We cannot detect optical flow $\boldsymbol{x}$ at the points which satisfy the condition $\nabla f(\boldsymbol{x},\tau,t)$, and we have the relation $f_t(\boldsymbol{x},\tau,t)$, that is, if $f(\boldsymbol{x},\tau,t) = 0$,*

$$(\nabla f(\boldsymbol{x},\tau,t)^{\top}, \frac{\partial}{\partial t}f(\boldsymbol{x},\tau,t))^{\top} = 0. \tag{23}$$

Since, at the points which satisfy the relations $\nabla f = 0$ and $f_t = 0$, we cannot detect the flow vector $\dot{\boldsymbol{x}}$, This theorem implies that for the detection of the topology trajectory in the linear scale space, we are required to develop a method for the detection of the trajectory of the peaks $\nabla f(\boldsymbol{x},\tau,t) = 0$ in the spatio-temporal domain, without using optical flow.

4 Example

Figures 1, 2, and 3 show a sequence of beating heart images with field of attentions, optical flow and trees extracted stationary-curves, respectively.

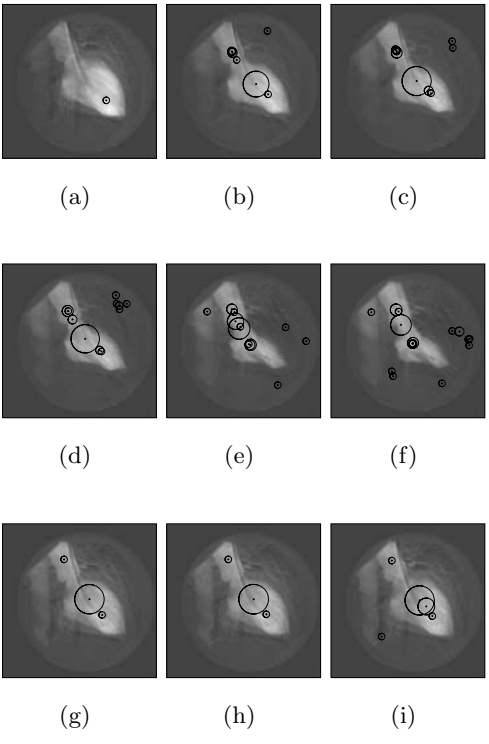


Fig. 1. Sequence of beating heart. 1,6,7,8,11,12,13,31,32,33,38 frames from 1 to 38 frames. From 6 to 7, from 7 to 8, from 11 to 12, from 12 to 13, and from 31 to 32, there is topological changes. From 1 to 6, from 8 to 11, from 13 to 31, and from 33 to 38, there is no topological changes. These changes extracted using tree analysis. Circles and points on images are field of attentions and the centre of the field of attentions, respectively

From these three series of features extracted from an image sequence of a beating heart, we can observe the following mathematical and geometrical properties related to the trajectory of the view-fields and optical flow fields.

Observations

- From 6 to 7, from 7 to 8, from 11 to 12, from 12 to 13, from 31 to 32, and from 32 and 33, gray-values of images are topologically different.
- From 1 to 6, from 8 to 11, from 13 to 31, and from 33 to 38, gray-values of images are topologically equivalent.

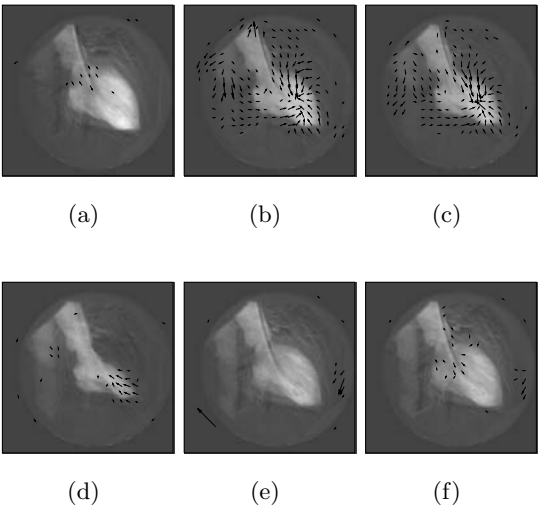


Fig. 2. Sequence of stationary-curves of beating heart. Frames 1 and 2, frames 6 and 7, frames 7 and 8, frames 11 and 12, frames 31 and 32, and frames 37 and 38

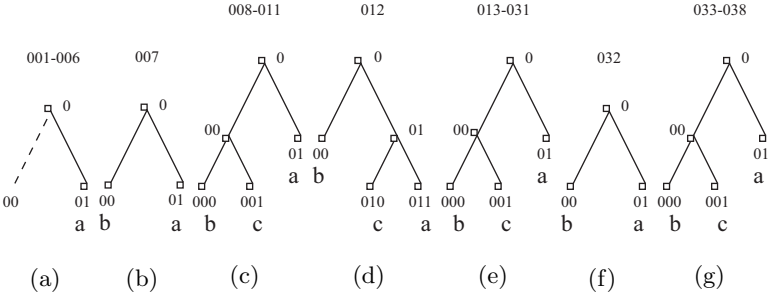


Fig. 3. Temporal trees of a sequence of images. (a) Tree of from 1 to 6. (b) Tree of 7. (c) Tree of from 8 to 11. (d) Tree of 12. (e) Tree of from 13 to 31. (f) Tree of 32. (g) Tree of from 33 to 38

- In from frames 6 to 8, while parts of the images move, the attention points and the attention fields move.
- In frames 11 and 14, the topological changes of structure-curves reveal the topographical changes in the gray values of images in the temporal sequence.
- From frames 13 to 32, when the gray values of parts of image change topographically.

These observations on the trajectory of the attention fields and optical flow of images in a sequence lead to the conclusion that the temporal analysis of the attention fields based on the stationary-curves in the linear scale space is suitable for the extraction of the dominant motions of parts in an image sequence.

The temporal tree analysis found that from frames 6 to 7, from frames 7 to 8, from frames 11 to 12, from frames 12 to 13, from frames 31 to 32, and from frames 32 and 33, gray-values of images are topologically different, and that from frames 1 to 6, from frames 8 to 11, from frames 13 to 31, and from frames 33 to 38, the gray values of images are topologically equivalent. These topological transitions extracted using the tree metric for trees shown in Figure 3, The distances among trees which are topologically different are

$$\begin{aligned} &D(T(6), T(7)) = 1.5, \quad D(T(7), T(8)) = 3, \\ &D(T(11), T(12)) = 3, \quad D(T(12), T(13)) = 1.5, \\ &D(T(31), T(32)) = 1.5, \; D(T(32), T(33)) = 1.5. \end{aligned}$$

These distances achieves the grouping of images in a sequence in to

Frames 0 to 6, Frame 7, Frames 8 to 11, Frame 12,
Frames 13 to 31 , Frame 32, Frames 33 to 38.

Images in a sequence of a beating heart are separated into groups on the basis of the tree-editing distances of the stationary trees of images. This result shows that the edit-distance-based method allows us to derive groups of images in a temporal sequence.

5 Conclusions

As an application of the scale space analysis for time-varying images, we developed a method for the extraction of moving parts from a sequences of images. The method does not assume the scale space transformation with respect to the time variable. Computational examples show that our method is suitable for the spatiotemporal analysis of sequences of low-contrast images such as ultrasonic images.

References

1. Iijima, T., *Pattern Recognition,* Corona-sha, Tokyo, 1974 (in Japanese).
2. Zhao, N.-Y., Iijima, T., Theory on the method of determination of view-point and field of vision during observation and measurement of figure IECE Japan, Trans. D., **J68-D**, 508-514, 1985 (in Japanese).
3. Zhao, N.-Y., Iijima, T., A theory of feature extraction by the tree of stable viewpoints. IECE Japan, Trans. D., **J68-D**, 1125-1135, 1985 (in Japanese).
4. Zhao, N.-Y., *A Study of Feature Extraction by the Tree of Stable View-Points,* Dissertation to Doctor of Engineering, Tokyo Institute of Technology, 1985 (in Japanese).
5. Weickert, J., *Anisotropic Diffusion in Image Processing,* Teubner, 1998.
6. Witkin, A.P., Scale space filtering, Pros. of 8th IJCAI, 1019-1022, 1993.
7. Yuille, A. L., Poggio, T., Scale space theory for zero crossings, IEEE PAMI, **8**, 15-25, 1986.
8. Lindeberg, T., *Scale-Space Theory in Computer Vision,* Kluwer, Boston 1994.

9. Lindeberg, T. Feature detection with automatic selection, International Journal of Computer Vision, **30**, 79-116, 1998.
10. Kuijper, A., Florack, L.M.J., Viergever, M.A., Scale Space Hierarchy, Journal of Mathematical Imaging and Vision **18**, 169-189, 2003.
11. Kuijper, A.; Florack, L.M.J., The hierarchical structure of images, IEEE, Trans. Image Processing **12**, 1067- 1079, 2003.
12. Imiya, A., Katsuta, R. Extraction of a structure feature from three-dimensional objects by scale-space analysis, LNCS, **1252**, 353-356, 1997.
13. Imiya, A., Sugiura, T., Sakai, T, Kato, Y., Temporal structure tree in digital linear scale space, LNCS, **2695**, 356-371, 2003.
14. K. Zhang, A constrained edit distance between unordered labeled trees. Algorithmica **15**, 205-222, 1996.
15. Pelillo, M., Siddiqi, K., Zucker, S.W., Matching hierarchical structures using Association graphs, IEEE, Trans, PAMI **21**, 1105-1120, 1999.
16. Kawashima, T., Imiya, A., Nishida, F., Approximate tree distance, Technical Report of IEICE, PRMU96-36, 81-87, 1996 (In Japanese).
17. Barron, J. L., Fleet, D. J., Beauchemin, S. S., Performance of optical flow techniques, International Journal of Computer Vision, **12**, 1994, 43-77.
18. Beauchemin, S. S. and Barron, J. L., The computation of optical flow, ACM Computer Surveys **26**, 1995, 433-467.
19. Horn, B. K. P. and Schunck, B. G., Determining optical flow, Artificial Intelligence, **17**, 185-204, 1981.
20. Nagel, H.-H., On the estimation of optical flow: Relations between different approaches and some new results. Artificial Intelligence, **33**, 299-324, 1987.
21. http://sampl.eng.ohio-state.edu/ sampl/data/motion/Heart/index.htm

Protein Classification with Kernelized Softassign

Miguel Angel Lozano and Francisco Escolano

Robot Vision Group,
Departamento de Ciencia de la Computación e Inteligencia Artificial,
Universidad de Alicante, Spain
{malozano, sco}@dccia.ua.es
http://rvg.ua.es

Abstract. In this paper we address the problem of comparing and classifying protein surfaces through a kernelized version of the Softassign graph-matching algorithm. Preliminary experiments with random-generated graphs have suggested that weighting the quadratic cost function of Softassign with information coming from the computation of diffusion kernels on graphs attenuate the performance decay with increasing noise levels. Our experimental results show that this approach yields a useful similarity measure to cluster proteins with similar structure, to automatically find prototypical graphs representing families of proteins and also to classify proteins in terms of their distance to these prototypes. We also show that the role of kernel-based information is to smooth the obtained matching fields, which in turn results in noise-free prototype estimation.

1 Introduction

In recent years, there has been a growing interest in exploiting the 3D information derived from the molecular surfaces of proteins in order to infer similarities between them. This is due to the fact that, as molecular function occurs at the protein surface, these similarities contribute to understand their function and, in addition, they reveal interesting evolutionary links between them. Particularly, there are two computational problems in which surface-based methods play a key role: *registration*, that is, determining whether two proteins have a similar shape, and thus, develop a similar function; and *docking*, that is, determining whether two proteins have a complementary shape and thus they can be potentially bounded to form a complex (for antigen-antibody reactions, enzyme catalysis, and so on). In this regard, the two central elements needed to solve both problems are: a proper surface description and an efficient and reliable matching algorithm [1]. Surfaces are often described by the interest points associated to concave, convex or toroidal patches of the so called Connolly surface [2], the part of the van der Waals surface accessible/excluded to/by a probe with a typical radius of 1.4Å. Given such a description, and assuming molecular rigidity, it is possible to find the optimal 3D transformation satisfying the condition of shape coincidence or complementarity. Geometric hashing [3] [4], which infers the most effective transformation needed

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 332–341, 2005.

to make compatible the interest points of both protein surfaces, has proved to be one of the most effective approaches in this context.

Other approaches related to computer vision come from graph theory and usually exploit discrete techniques to find the maximal clique [5] or the maximum common subgraph [6] in graphs with vertices encoding information of parts of the surfaces and edges representing topological or metric relations between these parts. However, the avenue of more efficient, though approximate, continuous methods for matching [7] [8] [9] [10] recommends an exhaustive evaluation of graph-based surface registration and docking under this perspective. One of this methods is the graduated assignment algorithm known as Softassign and proposed by Gold and Rangarajan [7]. Softassign optimizes an energy function, which is quadratic with respect to the assignment variables, through a continuation process (deterministic annealing) which updates the assignment variables while ensuring that a tentative solution represents a feasible match, at least in the continuous case. After such process, a cleanup heuristic translates the continuous solution to a discrete one. However, it has been reported that the performance of this algorithm decays significantly with increasing levels of structural corruption (number of deleted nodes), and also that such a decay can be attenuated by optimizing an alternative non-quadratic cost function [8]. Furthermore, in our preliminary experiments with random graphs, we have reported a similar decay if we weight properly the original quadratic function. Such a weighting relies on distributional information coming from kernel computations on graphs. The key idea is that when working with non-attributed graphs there is a high level of structural ambiguity and such ambiguity is not solved by classical Softassign. However, when using kernels on graphs [12] [13] we obtain structural features associated to the vertices which contribute to remove ambiguities, because they help to choose a proper attractor in contexts of high structural ambiguity. Kernels on graphs are derived from recent results in spectral graph theory [14], particularly the so called *diffusion kernels*, and provide a measure of similarity between pairs of vertices in the same graphs. In the case of diffusion kernels, such a similarity relies on the probability that a lazy random walk (a random walk with a given probability of resting at the current vertex) reaches a given node from another one.

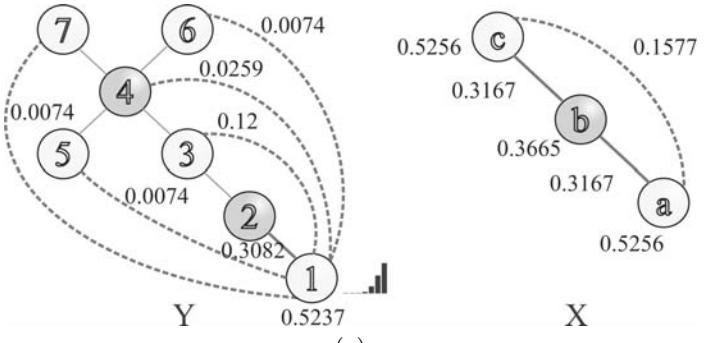
In this paper, we address the problem of protein surface comparison from the surface graphs extracted after labelling interest points/patches as concave or convex. This labelling provides application-driven attributes, and we are interested in knowing the role of structural attributes coming from kernel computation in this context. In section 2, we present the kernelized Softassign with attributes. In section 3 we present some representative experiments showing the application of this algorithm to matching surface graphs, and, finally, in section 4, we present our conclusions and outline our future work in this area.

2 Kernelized Softassign

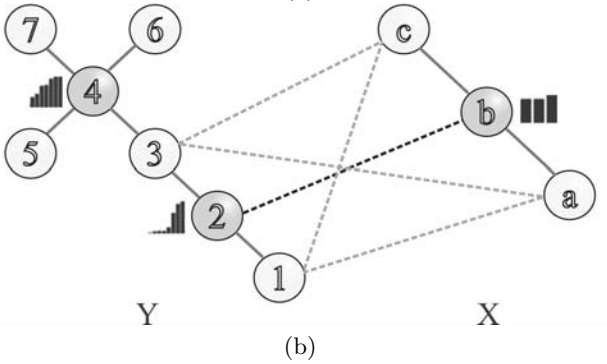
Consider two graphs $G_X = (V_X, E_X)$ and $G_Y = (V_Y, E_Y)$, with $m = |V_X|$ and $n = |V_Y|$, and adjacency matrices X_{ab}, and Y_{ij} ($X_{ab} = 1$ if $(a, b) \in E_X$, and

the same holds for Y). In the Softassign formulation a feasible solution to the matching problem is encoded by a matrix M of size $m \times n$ with $M_{ai} = 1$ if $a \in V_X$ matches $i \in V_Y$ and 0 otherwise, and satisfying the constrains that each node in V_X must match either a unique node in V_Y or none of them, and viceversa. Thus, following the Gold and Rangarajan formulation we are interested in finding the feasible solution M that maximizes the following cost function,

$$F(M) = \frac{1}{2} \sum_{a=1}^{m} \sum_{i=1}^{n} \sum_{b=1}^{m} \sum_{j=1}^{n} M_{ai} M_{bj} C_{aibj} \ , \tag{1}$$



(a)



(b)

$$K_Y = \begin{bmatrix} .524 & .308 & .120 & .026 & .007 & .007 & .007 \\ .308 & .336 & .214 & .065 & .026 & .026 & .026 \\ .120 & .214 & .280 & .137 & .083 & .083 & .083 \\ .026 & .065 & .137 & .191 & .194 & .194 & .194 \\ .007 & .026 & .083 & .194 & .475 & .107 & .107 \\ .007 & .026 & .083 & .194 & .107 & .475 & .107 \\ .007 & .026 & .083 & .194 & .107 & .107 & .475 \end{bmatrix} \quad K_X = \begin{bmatrix} .526 & .317 & .158 \\ .317 & .367 & .317 \\ .158 & .317 & .526 \end{bmatrix}$$

(c)

Fig. 1. Kernelized Softassign. (a) Kernel information for node 1 of graph Y and for all nodes in graph X. (b) Matching with kernelized Softassign. (c) Kernel matrices

where typically $C_{aibj} = X_{ab}Y_{ij}$. The latter cost function means that when $a \in V_X$ matches $i \in V_Y$, it is convenient that nodes b adjacent to a ($X_{ab} \neq 0$) also match nodes j adjacent to i (also $Y_{ij} \neq 0$). This is the well-known rectangle rule (we want to obtain the maximum number of closed rectangles $M_{ai}Y_{ij}M_{jb}X_{ba}$ as possible.

Furthermore, let C^K_{aibj} be a redefinition of C_{aibj} by considering the information contained in the diffusion kernels derived from X and Y. Such kernels are respectively $m \times m$ and $n \times n$ matrices, satisfying the semi-definite condition for kernels, which are derived from the Laplacians of X and Y. The Laplacian of X, is the $m \times m$ matrix $L_X = D_X - X$, where D_X is a diagonal matrix registering the degree of each vertex, and the same holds for L_Y. It turns out that the kernels K_X and K_Y result from the matrix exponentiation of their respective Laplacians:

$$K_X = e^{-\frac{\beta}{m}L_X} \text{ and } K_Y = e^{-\frac{\beta}{n}L_Y} ,$$

where the normalization factors depending on the number of vertices of each graph have been introduced to make both kernels comparable. Then, the definition of C^K_{aibj} involves considering the values of K_X and K_Y as structural attributes for the corresponding vertices (see Fig. 1) because as the Laplacians encode information about the local structure of the graph, its global structure emerges in the kernels. However, we have found that, as edit operations will give different kernels in terms of the diffusion processes, it is not an adequate choice to build attributes in the individual values of the kernels. Particularly, as for a given vertex (row) these values represent probabilities of reaching the rest of the nodes (columns), it can be considered that a given row represents a probability distribution (see Fig. 1) and we may use a characterization of such distribution to build structural attributes. Consequently, we define C^K_{aibj} as follows

$$C^K_{aibj} = X_{ab}Y_{ij}\delta_{abij}K_{abij}, \tag{2}$$

where

$$K_{aibj} = \exp -[(H^{K_X}_a - H^{K_Y}_i)^2 + (H^{K_X}_b - H^{K_Y}_j)^2] , \tag{3}$$

where H^{K_X} and H^{K_Y} are the entropies of the probability distributions associated to the vertices and induced by K_X and K_Y, and $\delta_{aibj} = 1$ if a matches a vertex i with the same curvature, and also b matches a vertex j with a similar curvature; otherwise it returns -1. This definition integrates the surface attributes in the cost function. When curvature compatibility arises, the latter definition of C^K_{aibj} ensures that $C^K_{aibj} \leq C_{aibj}$, being the equality only verified when nodes a and i have similar entropies, and the same for nodes b and j. In practice, this weights the rectangles in such a way that rectangles with compatible entropies in their opposite vertices are preferred, and otherwise they are underweighted and do not attract the continuation process. In such process, the matching variables are updated as follows (before being normalized)

$$M_{ai} = \exp\left[-\frac{1}{T}\frac{\partial F}{\partial M_{ai}}\right] = \exp\left[\frac{1}{2T}\sum_{i=a}^{m} M_{bj}C^K_{aibj}\right] , \tag{4}$$

To see intuitively how the kernelized version works, in Fig. 1 (vertices in dark grey represent convex patches, whereas vertices in light grey represent concave ones) we show the matching preferred by the kernelized Softassign just before performing the clean-up heuristic. Such a matching is the most coherent one in terms of structural subgraph compatibility.

3 Experimental Results and Discussion

The purpose of this paper is to test the reliability and efficiency of the kernelized Softassign described above in protein surface comparison. Before considering several protein families to test whether the approach is useful or not, we have measured the decay of the *normalized cost*, that is, the optimal matching cost between two graphs divided by the minimum of the self-matching costs for each compared graph, as we introduce more and more structural noise. In Fig 2 we show the results obtained for matching the protein $1crn$ with itself in the ideal case (isomorphism) and after artificially removing some atoms. We observe a progressive decay of the normalized cost.

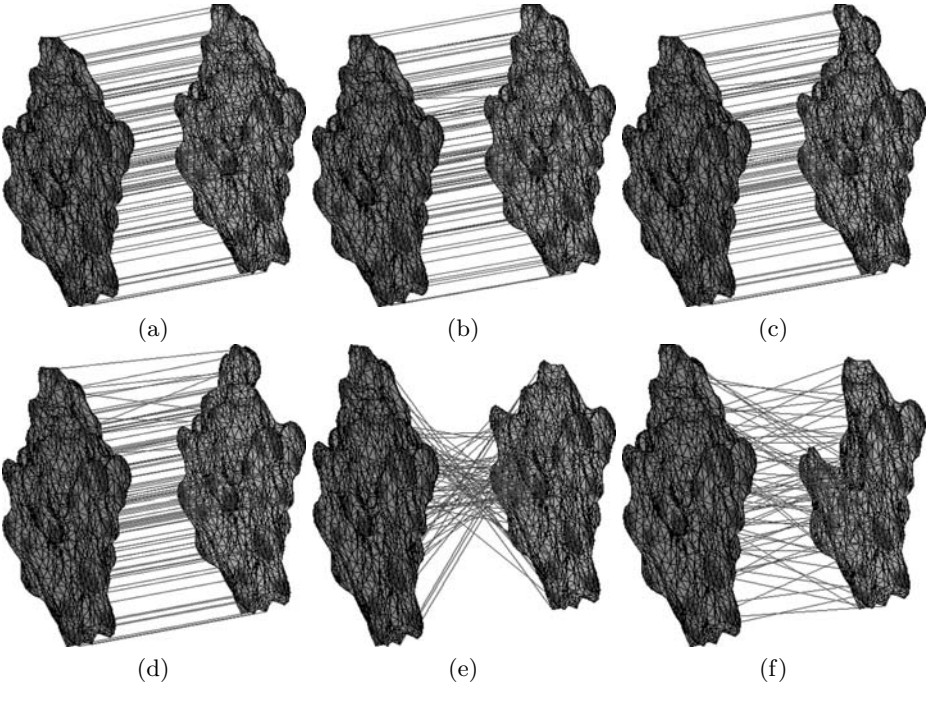


Fig. 2. Matching results for $1crn$ (328 atoms) with artificial noise. (a) Isomorphism ($cost = 1$), (b) removing 10 atoms ($cost = 0.9561$), (c) removing 20 atoms ($cost = 0.9203$), (d) removing 30 atoms ($cost = 0.9027$), (e) removing 100 atoms ($cost = 0.5528$), (f) 150 atoms ($cost = 0.547$)

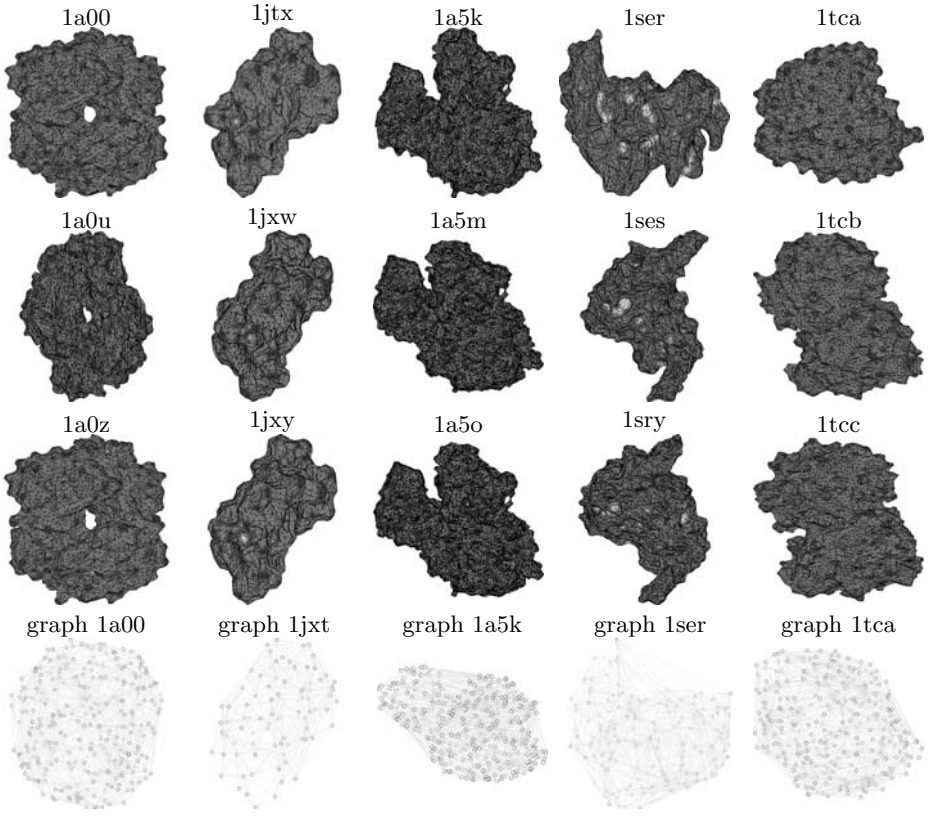
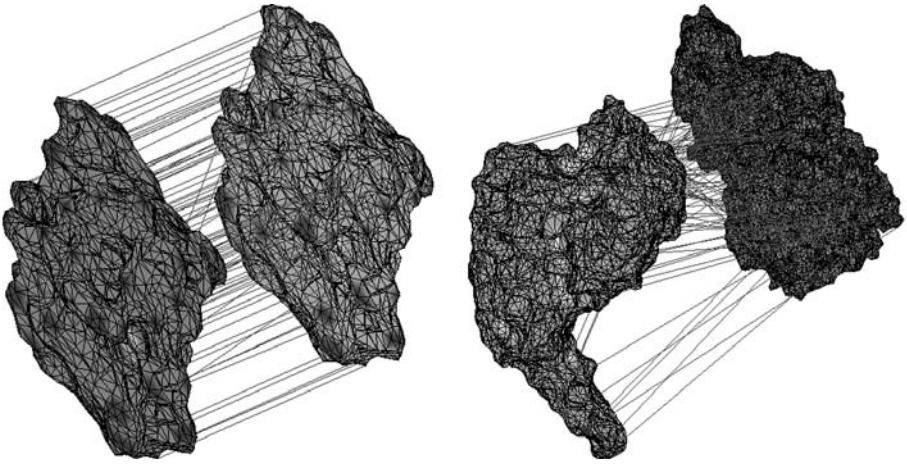


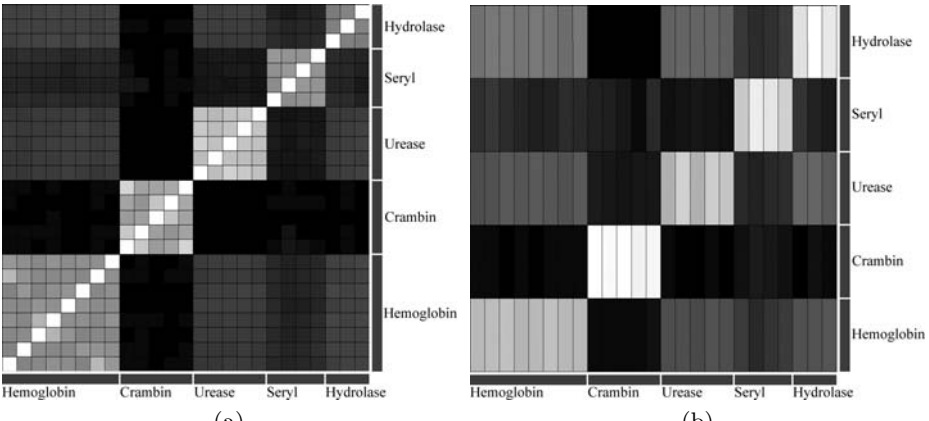
Fig. 3. Examples of the five families used in the experiments. From left-right and top-bottom: Hemoglobins, Crambin (plant proteins), Ureases, Seryl, Hydrolases. In the surfaces, convex patches are represented in dark grey whereas concave patches are showed in light grey. In the bottom row we show the graphs for the surfaces showed in the first row

In order to build a representative experimental set, we have considered proteins of five families extracted from the Protein Data Bank [15] (PDB): Crambin/Plant proteins, Ureases, Seryl, Hemoglobins, and Hydrolases. Given the triangulated Connolly surfaces of each protein, and considering the classification of each surface point as belonging to a concave, convex or toroidal patch, we have clustered the points in patches and we have associated each cluster to a vertex in the surface graph. Retaining only concave and convex patches/vertices we have obtained the edges of the surface graph through 3D triangulation of the centroids of all patches. (see Fig. 3). The number of vertices in the surface graphs range from 79 to 442 nodes, that is, we have heterogeneous families. We want to demonstrate that the normalized cost is a useful distance for predicting whether two proteins belong to the same family or not. Such a normalized cost is bounded by 0 and 1, and the higher the cost the more similar the proteins



(a) (b)

Fig. 4. Some matching results. (a) Correct matching between two proteins of the same family ($1jxt - 1jxu$, with cost 0.7515). (b) Incorrect matching between proteins of different families ($1a5k - 1set$, with cost 0.0701)



(a) (b)

Fig. 5. Matching results for several families: Hemoglobins ($1a00$, $1a01$, $1a0u$, $1a0v$, $1a0w$, $1a0x$, $1a0z$, $1gzx$), Crambin ($1jxt$, $1jxu$, $1jxw$, $1jxx$, $1jxy$), Ureases ($1a5k$, $1a5l$, $1a5m$, $1a5n$, $1a5o$), Seryl ($1ser$, $1ses$, $1set$, $1sry$) and Hydrolases ($1tca$, $1tcb$, $1tcc$). (a) Comparisons all-for-all. (b) Comparisons with all the prototypes

should be. In Fig. 4 (top row) we show some representative matching results. Comparing proteins of the same family results in globally smooth matchings and normalized costs above 0.5 ($1jxt - 1jxu$ in Crambin). However, the cost falls below 0.1 for many proteins of different families (comparing an urease and a seryl, $1a5k - 1set$, we obtain a cost of 0.0701) because in these cases it is not

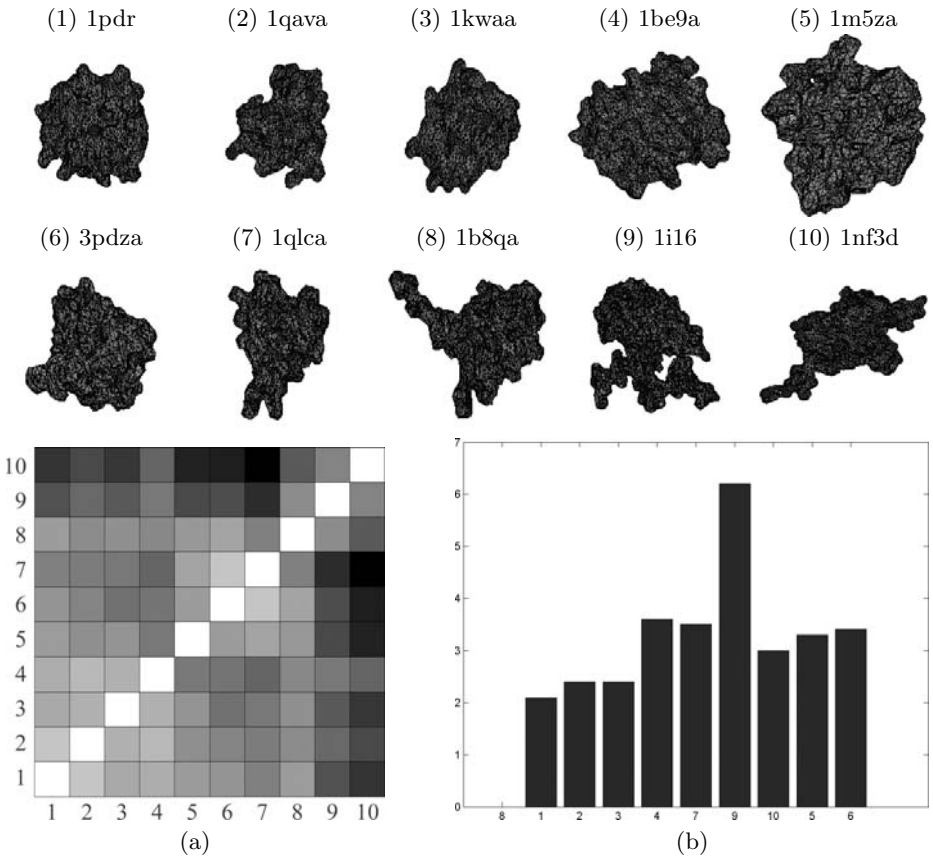


Fig. 6. Comparison with 3D structure. Top: 1*b*8*qa* and the 9 closer proteins following Dali scores but ordered by surface compatibility. Bottom left: Pairwise normalized distances. Bottom right: RMS error given by Dali

possible to find globally consistent matches (only pairs of common subgraphs are matched).

Another important fact to test is whether the kernelized version improves significantly the performance of the classical version of Softassign. In our preliminary experiments for the non-attributed case, where the effect of structural ambiguity is higher, we have found that kernelization yields, in comparison with the classical Softassign, a slower performance decay with increasing structural corruption. However, this does not happen in our surface comparison experiments because the concavity/convexity attributes constrain so much the number of allowable correspondences that there are few pure structural ambiguities. However, although using kernelized or classical Softassign has a negligible impact in the normalized cost obtained (see the bottom row in Fig. 4) we obtain noise-free representative graph prototypes for different families. These prototypical graphs are interesting because an automatic clustering (see our previous

work in [16]) and prototype building relying on the normalized cost yields an useful abstraction for subsequent protein queries. First of all, we test whether the normalized cost yields higher similarities between proteins in the same family than between proteins of different families. In Fig. 5 (left) we show the all-for-all similarities between proteins of different families (8 Hemoglobins, 5 Crambins, 5 Ureases, 4 Seryls, and 3 Hydrolases) which yields good clustering in the diagonal (white/grey level indicates high similarities). In Fig. 5 (right) we show the similarities between each protein and the prototype of all classes which indicates that the prototypical graphs obtained through both kernelized graph-matching and incremental fusion are very representative.

Finally, we compare the similarities due to surface comparison to the similarities due to 3D structure comparison (such information comes from the backbone of the protein but not from its surface). One of the most effective methods for 3D fold comparison is Dali [17] [1]. In Fig. 6 we show the 9 proteins closer, in terms of 3D structure, to $1b8qa$. When analyzing these proteins in terms of their surfaces we obtain three sparse classes: $\{1pdr, 1qava, 1kwaa, 1be9a\}$, $\{1m5za, 3pdza, 1qlca, 1b8qa\}$, and $\{1i16, 1nf3d\}$.

4 Conclusion and Future Work

In this paper we have presented a graph-matching method, relying on discrete kernels on graphs, to solve the problem of protein surface comparison. We extract the graphs describing the topology of the surfaces and perform efficient and reliable graph matching by exploiting both application-driven and structural attributes. Our experimental results with proteins coming from five different families show that the normalized cost derived from the kernelized Softassign algorithm is a useful similarity measure to cluster proteins and also to build structural prototypes for each family. These prototypes may be very useful for simplifying subsequent protein queries. Our future work in this field includes the study of different graph kernels, the proposal of a full kernelized graph-matching algorithm (not only the cost function) and the formalization of graph-edit distances in terms of kernels.

Acknowledgments. This work was partially supported by the research grant $PR2004-0260$ of the Spanish Government.

References

1. Halperin, I., Ma, B., Wolfson, H. & Nussinov, R. (2002) Principles of docking: An overview of search algorithms and a guide to scoring functions. *Proteins* **47**, pp. 409-443
2. Connolly, M. (1983) Analytical molecular surface calculation. *J Appl Crys* **16** pp. 548-558.

[1] http://www.bioinfo.biocenter.helsinki.fi:8080/dali/index.html

3. Wolfson, H.J. & Lamdan, Y. (1988) Geometric hashing; A general and efficient model-based recognition scheme. In *Proc of the IEEE Int Conf on Computer Vision* 238-249.
4. Nussinov, R., Wolfson, H.J. (1991) Efficient detection of three-dimensional motifs in biological macromolecules by computer vision techniques. In *Proc of the Natl Acad Sci USA* **88** pp. 10495-10499.
5. Gardiner, E.J., Willet, P., Artymiuk, P.J. (2000) Graph-theoretic techniques for macromolecular docking. *J Chem Inform Comput Sci* **40** 273-279.
6. Pickering, S.J., Bulpitt, A.J., Efford, N., Gold, N.D. & Westhead, D.R. (2001) AI-based algorithms for protein surface comparisons. *Computers and Chemistry* **26** 79-84.
7. Gold, S. & Rangarajan, A. (1996) A graduated assignment algorithm for graph matching. *IEEE Trans on Patt Anal and Mach Int* **18** (4) 377-388.
8. Finch, A.M., Wilson, R.C. & Hancock, E. (1998) An energy function and continuous edit process for graph matching. *Neural Computation* **10** (7) 1873-1894.
9. Pelillo, M. (1999) Replicator equations, maximal cliques, and graph isomorphism. *Neural Computation* **11** 1933-1955.
10. Luo, B. & Hancock, E.R. (2001) Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Trans on Patt Anal and Mach Int* **23** (10) 1120-1136.
11. Lozano, M.A. & Escolano, F. (2004) A significant improvement of softassign with diffusion kernels. In *Proc of the IAPR Workshop on Synt and Struct Patt Rec* LNCS (Accepted for publication).
12. Kondor, R. & Lafferty, J. (2002) Diffusion kernels on graphs and other discrete input spaces. In *Proc of the Intl Conf on Mach Learn.* Morgan-Kauffman 315-322.
13. Smola, A. & Kondor, R.I.(2003) Kernels and Regularization on Graphs. In *Proc. of Intl Conf on Comp Learn Theo and 7th Kernel Workshop.* LNCS. Springer **2777** 144-158.
14. Chung, F.R.K. (1997) Spectral graph theory. *Conference Board of the Mathematical Sciences(CBMS)* **92**. Americal Mathematical Society.
15. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N. & Bourne, P.E. (2000) The Protein Data Bank. *Nucleic Acids Research* **28** pp. 235-242.
16. Lozano, M.A. & Escolano, F. (2003) EM algorithm for clustering an ensemble of graphs with comb matching. In *Proc of the Intl Workshop on Energ Minim Meth in Comp Vis and Patt Rec.* LNCS **2683** 52-67.
17. Holm, L., Sander, C. (1993) Protein structure comparison by alignment of distance matrices. J Mol Biol 233(1): 123-38.

Local Entropic Graphs for Globally-Consistent Graph Matching

Miguel Angel Lozano and Francisco Escolano

Robot Vision Group,
Departamento de Ciencia de la Computación e Inteligencia Artificial,
Universidad de Alicante, Spain
{malozano, sco}@dccia.ua.es
http://rvg.ua.es

Abstract. In this paper we propose a novel approach to obtain unambiguous and robust node attributes for matching non-attributed graphs. Such approach consists of exploiting the information coming from diffusion kernels to embed the subgraph induced by the neighborhood of each vertex in an Euclidean manifold and then use entropic graphs for measuring the $\alpha-$entropy of the resulting distribution. Our experiments with random-generated graphs with 50 nodes show that at low edge densities, where the effect of structural noise is higher, this approach outperforms the description of the subgraph only in terms of diffusion kernels. Furthermore, our structural recognition experiments show that the approach has a practical application. All experiments were performed by weighting the well-known quadratic cost function used in the Softassign algorithm.

1 Introduction

Considering object recognition, graphs yield useful invariant structural/relational information of patterns [4]. Structural recognition covers both *graph registration* and *graph clustering*. Considerable amount of research has been addressed in the past towards solving the graph registration problem: graph-matching [9][26][18], maximum clique finding [20], graph-edit distance minimization [23][5], and so on. These approaches have to face two main problems: (i) constrain the considerable amount of structural ambiguity arising in such representations; and (ii) tolerate acceptable levels of structural corruption. Such problems must be solved in order to provide globally-consistent registrations, and therefore, good metrics for graph comparison. Considering *graph clustering*, although the registration problem may be eluded if a proper mapping from graphs to a vector space is found [19][1], globally-consistent registration has a deep impact in finding and maintaining consistent graph prototypes [11][24]. For instance, in [14] we address the chicken-and-egg problem of finding the prototypes, the optimal number of classes, and classifying the graphs with an EM algorithm.

In our previous work we have addressed the problems of ambiguity and robustness by proposing sound node attributes for matching non-attributed graphs [15][16][17]. We followed a spectral approach which is closely related to spectral

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 342–351, 2005.

seriation [22]. So far, we have exploited kernels on graphs [12][25][13] in order to weight the quadratic cost function of the classical Softassign algorithm [9] (*kernelized Softassign*). Such weighting relies on characterizing the distribution of kernel values for each vertex, because such values encode the probabilities of reaching the rest of vertices from a given one. Our experimental results, when using diffusion kernels, show that we obtain a matching performance comparable to that obtained by using a non-quadratic cost function [8]. This result is due to the fact that the structural characterization given by the kernel is good enough for driving the graduated assignment method towards proper attractors in contexts of high ambiguity. Using random-generated graphs, we have performed comparative experiments between diffusion kernels and other kernels belonging to the more general class of regularization kernels. Such experiments show that diffusion kernels outperform the rest of kernels, although at low edge densities p-step random walk kernels outperform diffusion ones.

Furthermore, in practice, our structural recognition experiments report a good stability of diffusion kernels for matching unweighted graphs with sizes ranging from 30 to 140 nodes and coming from real images. This allowed us to automatically build graph prototypes and classifying graphs in terms of the distances to the closer prototype. One of the reasons of such success is that the considered graphs were enough dense for removing ambiguity while tolerating some degree of structural corruption. However, our previous results with random-generated graphs predict that in real cases with lower edge densities the noise tolerance will decay significantly. The relatively good behavior of p-step random walk kernel, which is a local kernel for low values of p (typically we set $p = 2$) in these situations focused us to characterize the distributional information more locally, that is, to compute the kernel of the subgraph induced by a given neighborhood around each vertex and use a characterization of such distribution to weight the Softassign quadratic cost function.

In this paper we show that this is not enough to increase the tolerance to noise at low densities. However, mapping the vertices, with the aid of kernel information, to an Euclidean manifold where they are considered as realizations of a probability density, the entropic graphs approach [10][6] allows to characterize such a density through the estimation of the $\alpha-$entropy or Rényi entropy. We report experimental results showing that the latter characterization yields a weighting which is good enough for tolerating high levels of structural noise even at low edge densities. In Section 2 we describe the application of entropic graphs to this context. In Section 3 we present our experiments both with random-generated graphs and with real graphs coming from images. We also show the impact of this improvements in graph classification and we outline our conclusions.

2 Diffusion Kernels and Entropic Graphs

Given a undirected and unweighted graph $G = (V, E)$ with vertex-set V of size n, and edge-set $E = \{(i, j)|(i, j) \in V \times V, i \neq j\}$, its adjacency matrix and degree matrix are respectively defined as

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \text{ and } D_{ij} = \begin{cases} \sum_{j=1}^{n} A_{ij} & \text{if } i = j \\ 0 & \text{otherwise .} \end{cases} \quad (1)$$

Then, the Laplacian of G, $L = D - A$ is a matrix with $L_{ij} = -1$ when $(i,j) \in E$ and with the degree values $L_{ii} = D_{ii}$ in the diagonal. The Laplacian matrix is key to compute the diffusion kernel $K = e^{-\sigma L}$, which is the solution of the diffusion/heat equation [7]

$$\frac{d}{d\sigma} K_\sigma = -L K_\sigma \quad (2)$$

and such a solution arises from exponentiating the spectrum of the Laplacian:

$$K = e^{-\sigma L} = \sum_{k=1}^{n} e^{-\lambda_k \sigma} \phi_k \phi_k^T \text{ that is } K_{ij} = \sum_{k=1}^{n} e^{-\lambda_k \sigma} \phi_k(i) \phi_k(j) \, , \quad (3)$$

where $\{\lambda_k, \phi_k\}$ is the eigensystem of L. As L is symmetric, $K = e^{-\sigma L}$ satisfies the semi-definiteness conditions for kernels, and consequently the elements K_{ij} of such a Gram matrix represent similarities between pairs of vertices.

Furthermore, the similarities K_{ij} given by the kernel have also a probabilistic interpretation in terms of lazy random walks (see [12]): The transition probability associated to the stochastic process $z_0, z_1, \ldots$, being $z_l \in V$, is $p(z_{l+1}|z_l) = \beta : (z_{l+1}, z_l) \in E$ with $\beta \leq 1/D_{z_l z_l}$, where the probability of remaining at each vertex $1 - \beta D_{z_l z_l}$. Then, K_{ij} can be interpreted as the sum of probabilities that a lazy random walk takes each of the paths between vertices i and j, and we have that K satisfies the conditions of a doubly stochastic matrix, that is

$$\sum_{i=1}^{n} K_{ij} = 1, \; j \in \{1, 2, \ldots, n\} \text{ and } \sum_{j=1}^{n} K_{ij} = 1, \; i \in \{1, 2, \ldots, n\} \, , \quad (4)$$

The latter probabilistic interpretation and the fact that K is the solution to the diffusion equation yield a connection with Gaussian distributions in continuous spaces. As noted in [1], in the context of graph embedding, the similarities K_{ij} are closely related to the path length distribution

$$K_{ij} = e^{-\sigma} \sum_{k=1}^{|V|^2} P_k(i,j) \frac{\sigma^k}{k!} \, , \quad (5)$$

being $P_k(i,j)$ the number of $k-$length paths between i and j, and such a distribution is conserved, in terms of geodesic distances, when the graph is embedded on a manifold in the Riemmann space. Furthermore, for a locally-Euclidean manifold we have the following approximation (see also [3])

$$K_{ij} \approx \frac{1}{(4\pi\sigma)^{\frac{d}{2}}} e^{-\frac{1}{4\sigma^2} w(i,j)^2} \text{ that is } w(i,j) \approx 2\sqrt{-\sigma \ln\{(4\pi\sigma)^{\frac{d}{2}} K_{ij}\}} \, , \quad (6)$$

being $w(i,j)$ the distance between vertices i and j on the Euclidean manifold and d is the dimension of such manifold. Thus, the idea that the diffusion kernel

is equivalent to the Gaussian kernel for discrete spaces yields a distance between the embedded vertices. What is more interesting for us is the intuition that the spatial distribution of these vertices yields a robust characterization of the graph structure. Such intuition is supported by an approach, called *entropic graphs* [10][6], which allows to estimate the entropy of the spatial distribution of vertices on the unique basis of the pairwise distances between them.

Given the vertices $i \in V$ in the $d-$dimensional Euclidean manifold, we map them, through a multidimensional scaling process, to points $\mathbf{x}_i \in \mathbb{R}^d$ satisfying $||\mathbf{x}_i - \mathbf{x}_j|| = w(i,j)$. Let $e_{ij} = (\mathbf{x}_i, \mathbf{x}_j)$ the edges of T, the minimum spanning tree (MST) connecting the latter points assuming that the cost of each connection is $||e_{ij}|| = w(i,j)$. It is obvious that the process yielding these distances $w(i,j)$ ensures that $T \subseteq G$, that is, $e_{ij} = (i,j) \in E$. Then, given $\gamma \in (0,d)$, the weighted length of the MST is defined as follows

$$L_{\gamma,d}(\{\mathbf{x}_i\}) = min_{e \in T} \sum_e ||e_{ij}||^\gamma \tag{7}$$

The formal connection between $L_{\gamma,d}$ and the characterization of the distribution of mapped points is set by the *Beardwood-Halton-Hammersley theorem*[2] which states that for $d \geq 2$, $\gamma \in [1,d)$ and $\alpha = (d-\gamma)/d$ we have the following limit

$$\lim_{n \to \infty} \frac{L_{\gamma,d}(\{\mathbf{x}_i\})}{n^\alpha} = \beta_{\gamma,d} \int f^\alpha(\mathbf{x}) d\mathbf{x} \, , \tag{8}$$

being the points $\{\mathbf{x}_i\}$ a realization of a Lebesgue density f with compact support in $\mathbb{R}^d$. Also, for $\gamma \in (0,d)$ the mean normalized weighted length $E(L_{\gamma,d}(\{\mathbf{x}_i\})/n^\alpha)$ converges to the same limit. What is interesting of such limit is that its second factor is a monotonic function of the Rényi entropy (or $\alpha-$entropy):

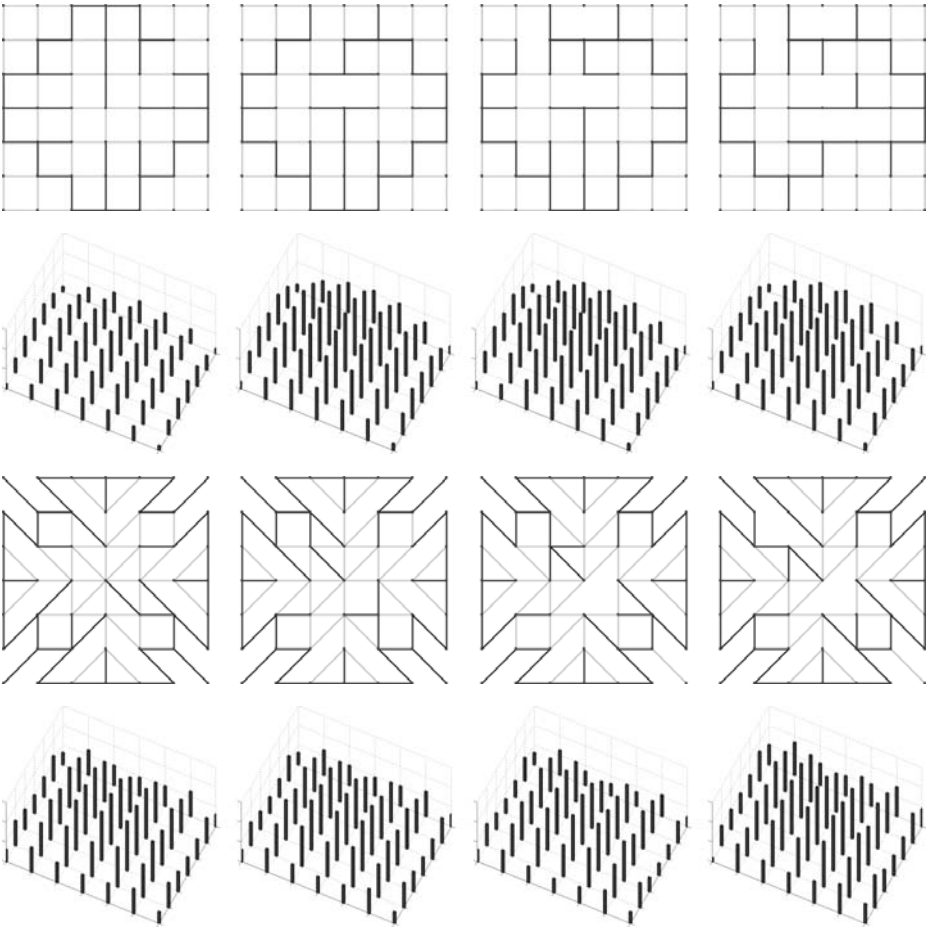
$$H_\alpha(f) = \frac{1}{1-\alpha} \log \int f^\alpha(\mathbf{x}) d\mathbf{x} \quad \text{being} \quad \lim_{\alpha \to 1} H_\alpha(f) = -\int f(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} \, , \tag{9}$$

that is, when $\alpha \to 1$ it converges to the Shannon entropy. Consequently

$$\hat{H}_\alpha(f) = \frac{d}{\gamma} \left[\log \frac{L_{\gamma,d}(\{\mathbf{x}_i\})}{n^{(d-\gamma)/d}} - \log \beta_{\gamma,d} \right] \tag{10}$$

is an asymptotically stable and strongly consistent estimator of the $\alpha-$entropy, being $\beta_{\gamma,d}$ a constant which does not depend on f (we use the approximation $\log \beta_{\gamma,d} \approx \gamma/2 \log(d/2\pi e)$ in this paper).

Our motivation for introducing entropic graphs is that kernel information is not robust enough to characterize even the local structure of the graph. Consider, for instance a reference node $i \in V$ and a neighborhood $N(i)$ with all vertices j satisfying $d(i,j) = \min_\Gamma \{l = |\Gamma| : \Gamma(0) = i, \Gamma(l) = j\} \leq \delta$, being Γ a path between i and j. Such a neighborhood induces a subgraph given by the vertices $j \in N(i)$ and the edges $(i,j) \in E$ between such vertices. Let $g_{ij} = K_{ij}$ the discrete density induced by the kernel, and $g_i = K_i$ the discrete distribution associated to the $i-$th row (vertex). If we measure the discrete Shannon entropy of such distribution $H(g_i) = -\sum_{j=1}^n g_{ij} \log g_{ij}$ we find that such entropy



Noise	Uniform Grid						*Gaussian* Grid					
	Central			Corner			Central			Corner		
	$L_{\gamma,d}$	$H_\alpha(f_i)$	$H(g_i)$	$L_{\gamma,d}$	$H_\alpha(f_i)$	$H(g_i)$	$L_{\gamma,d}$	$H_\alpha(f_i)$	$H(g_i)$	$L_{\gamma,d}$	$H_\alpha(f_i)$	$H(g_i)$
0	67.69	4.97	1.06	26.16	4.11	0.55	90.08	5.27	1.81	31.83	4.30	0.29
1	65.73	5.00	0.83	26.17	4.11	0.55	90.06	5.27	1.66	31.83	4.30	0.29
2	65.70	4.97	0.83	26.10	4.11	0.55	88.13	5.25	1.27	31.83	4.30	0.29
3	59.94	4.89	0.57	26.10	4.11	0.55	88.03	5.25	1.27	20.51	3.89	0.28

Fig. 1. Illustrating Entropic Graphs. First row: Graph from an uniform grid. Third row: Graph from a *Gaussian*-like grid. Second and Fourth rows: $\alpha-$entropies for each vertex. Bottom table: Comparison between H_α and H. In all cases the edges of the graphs are showed in grey and the edges of the MSTs are showed in black

changes significantly with minor edit operations located at such neighborhood . As kernels rely on spectral information, this reasoning is consistent with the fact that spectral seriation is prone to editing vertices or edges in the neighborhood of the leading ones. Analyzing the two graphs in Fig. 1 at different levels of edge noise (removing 0,1,2 and 3 edges) and considering $\delta = 4$ we observe that with increasing levels of noise the structure of the central node is confused with that of the upper-left corner in the uniform grid.

However, when we measure $H_\alpha(f_i)$, the $\alpha-$entropy in the subgraph induced by $N(i)$ we obtain a stronger tolerance to structural noise (we will provide sound experimental results in the following section). What is the contribution of entropic graphs to such increase of robustness? Intuitively, what happens is that although the distances $||\mathbf{x}_i - \mathbf{x}_j||$ between the mapped points rely on the kernel values, when such values are changed the estimation of the $\alpha-$entropy is sound provided that an alternative MST with similar cost exists. Thus, from the point of view of entropic graphs, the impact of edit operations depends on the existence of alternative paths for yielding the same estimation. Considering again Fig. 1, the cost of the MST is quite stable as the structural noise is increased. In the particular case of the second example graph (*Gaussian* Grid) the effect of removing many of the edges of the central node has no impact in the descriptor. It seems that the MST is reorganized effectively from the non-removed edges. In all cases $\alpha = 0.5$ ($\gamma = 1$ and $d = 2$).

3 Experimental Results and Discussion

For our experiments we have compared the weighing of the well-known quadratic cost function used in Softassign depending whether we use H_α relying on MST computation or H relying on kernel computation. In order to study the effect of such descriptors in the global consistency of the algorithm we have performed two sets of experiments: (i) using random-generated graphs (synthetic experiments) and (ii) using real graphs (recognition experiments).

Synthetic Experiments. In Fig. 2 we show some results obtained with graphs of 50 vertices. We report that at low levels of edge density the entropic graph approach (EG) outperforms clearly both the global diffusion kernel (GDK) and the local (subgraph) diffusion kernel (LDK). At high levels of edge density the differences of performance between them is negligible. Furthermore, as we have reported in our previous work such a performance is better than the obtained when using cardinality. These results, where noise is modelled by edge edition, is also consistent with node-edition noise, although the rate of performance decay is higher in all cases (we have not included these experiments in the paper due to space constraints).

Recognition Experiments. We have considered the CMU, MOV1 and Chalet sequences, which are associated to observing different objects (houses) from a smooth range of viewpoints (10 frames). The CMU sequence has graphs of 30 vertices on average, with an averaged corruption of 10% between consecutive

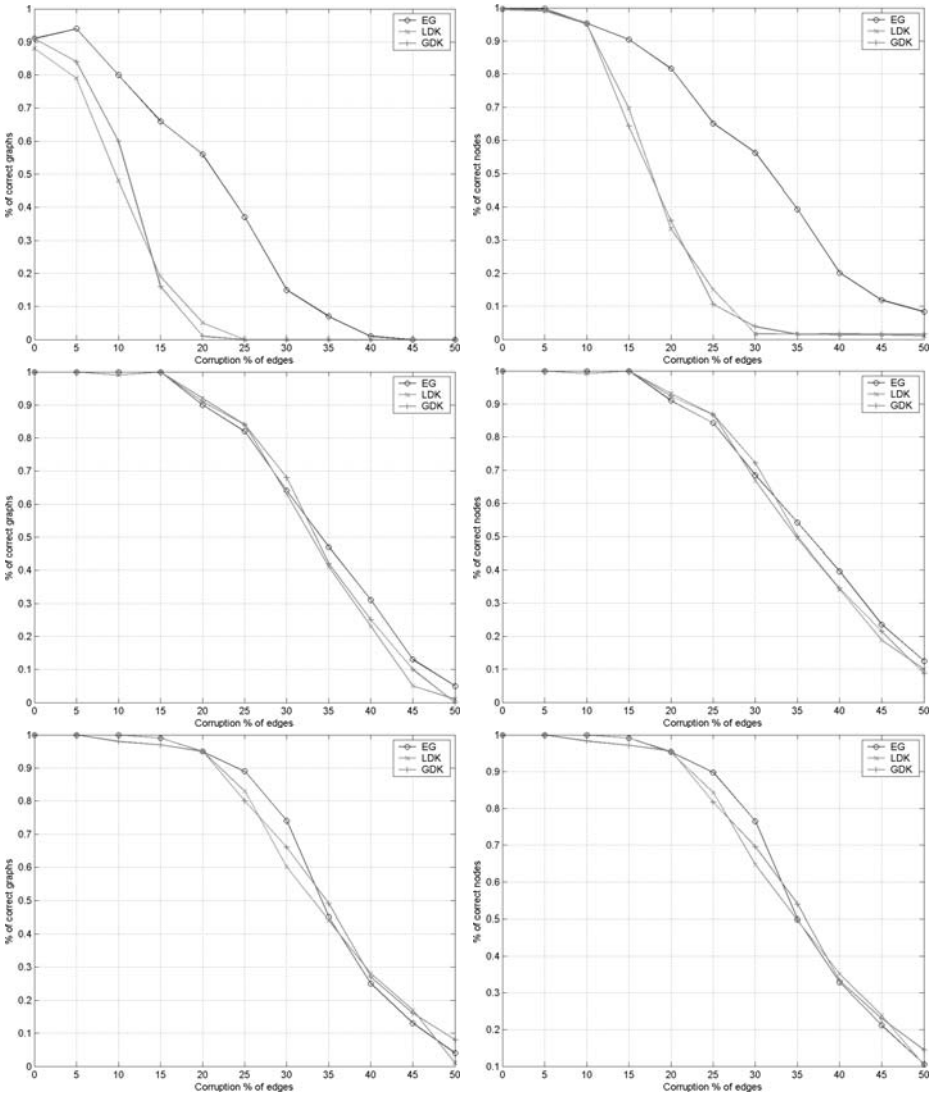


Fig. 2. Synthetic matching results. Graphs (first column) and nodes (second column) successfully matched with an edge density of 10% (first row), 20% (second row) and 30% (third row)

frames. The MOV1 sequence is very sparse (has graphs with 40 to 113 vertices) and the Chalet sequence has graphs of $131 - 140$ vertices. In the first two rows of Fig. 3, we show very good matching results between graphs associated to consecutive frames (*graph tracking*) of the CMU sequence (almost 100% of success). In the third and fourth rows we show also good tracking results even for the Chalet sequence which has very big graphs.

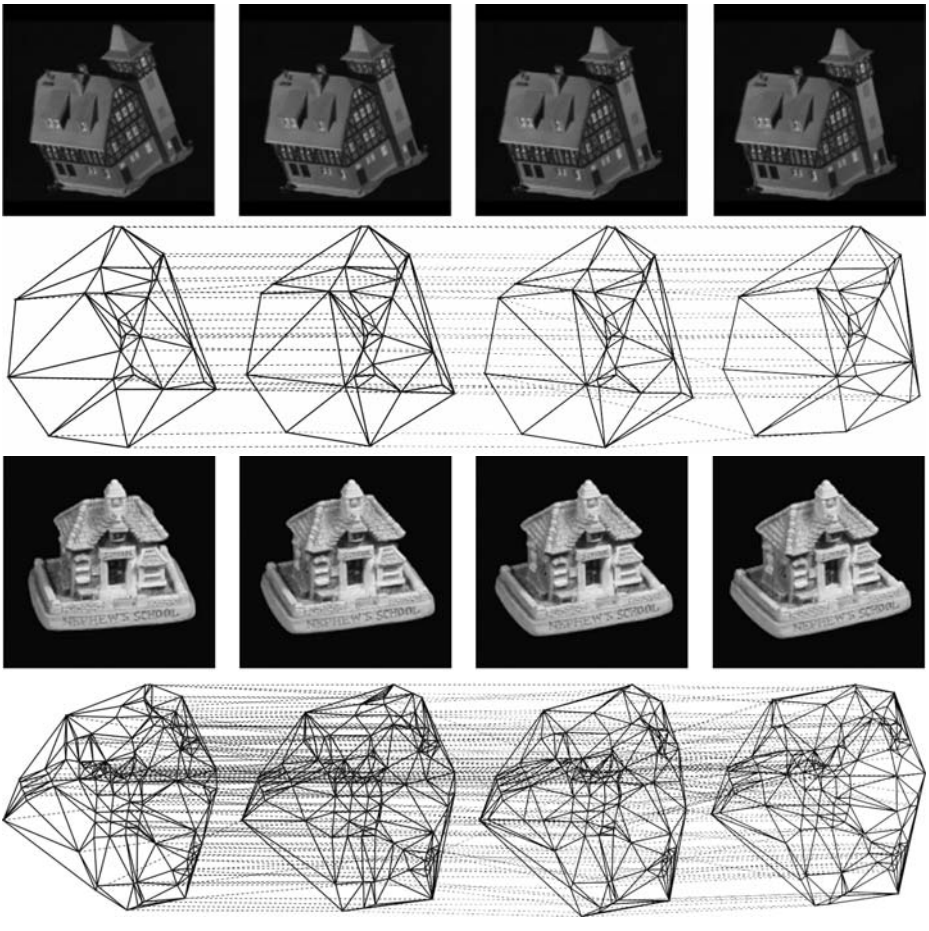
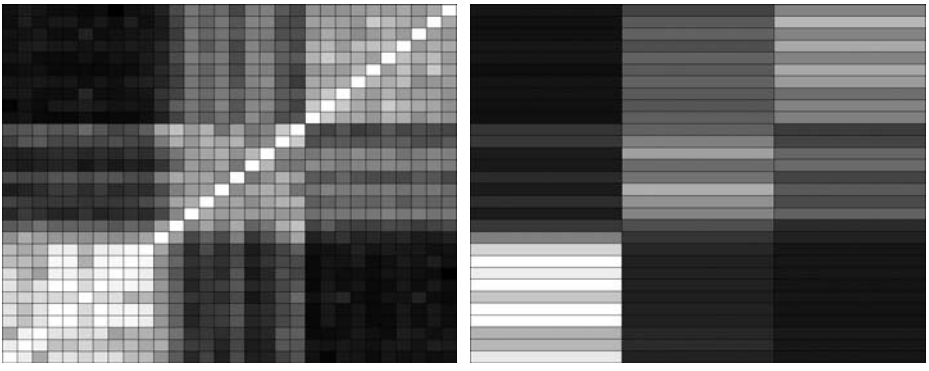


Fig. 3. Recognition results. First and second rows: CMU sequence. Third and fourth row: Chalet sequence

In order to address the problem of graph classification we have obtained the *confusion matrix* given by the normalized distances between all pairs of graphs in the three sequences: CMU, MOV1 and Chalet (see Fig. 4(a)). White indicates 100% of matching success (low distances) and black indicates 0% of success). We have also obtained the prototypes for the three sequences with our EM algorithm described in [14] and we have computed the minimal normalized distance between each graph and the three prototypes: The first 10 rows in Fig. 4(b) corresponds to CMU, the next ones to MOV1 and the rest to Chalet; the first column corresponds to the CMU prototype, the second to the MOV1 one and the third to the Chalet one. Considering the CMU prototype, graphs are well classified; for MOV1 and Chalet, the results are acceptable if one considers the high level of sparseness of these classes and the high number of vertices (in the case of the Chalet sequence). We conclude that due to the improved



(a) (b)

Fig. 4. Classification results. Left: Normalized pairwise distances between graphs. Right: Minimal distances to the prototypes

cost function, these classification results outperform slightly our previous results obtained with the GDK (global diffusion kernel) approach [17]. The reason of a such slight improvement is the fact that these experiments correspond to a medium edge density where all the latter approaches have a similar performance.

Acknowledgements

This work was partially supported by grant $TIC2002-02792$ funded by *Ministerio de Ciencia y Tecnología* of the Spanish Government and by *FEDER*.

References

1. Bai, X, Hancock, E. Heat Kernels, Manifolds and Graph Embedding. In A. Fred et al, (Eds.): SSPR&SPR 2004, LNCS **3138** (2004) 198–206
2. Bearwood, J., Halton, J.H., Hammersley, J.M. The Shortest Path Through Many Points. Proc. Cambridge Philosophical Society **55** (1959) 299–327
3. Belkin, M., Niyogi, P. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. Neural Information Processing Systems, **14** (2002) 634–640
4. Bunke,H.: Recent Developments in Graph Matching. In Proceedings of the International Conference on Pattern Recognition (ICPR'00)- Vol.2 (2000) 2117–2124
5. Bunke,H.: Error Correcting Graph Matching: On the Influence of the Underlying Cost Function. IEEE Transactions on Pattern Analysis and Machine Intelligence **21** (9) (1999) 917–922
6. Costa, J., Hero, A.O. Geodesic Entropic Graphs for Dimension and Entropy Estimation in Manifold Learning. IEEE Transactions on Signal Processing **52** (8) (2004) 2210–2221
7. Chung, F.R.K.: Spectral Graph Theory. Conference Board of the Mathematical Sciences (CBMS) **92**. American Mathematical Society (1997)
8. Finch, A.M., Wilson, R.C., Hancock, E.: An Energy Function and Continuous Edit Process for Graph Matching. Neural Computation, **10** (7) (1998) 1873-1894

9. Gold, S., Rangarajan, A.: A Graduated Assignment Algorithm for Graph Matching. IEEE Transactions on Pattern Analysis and Machine Intelligence **18** (4) (1996) 377–388
10. Hero, A.O., Ma, B., Michel, O., Gorman, J.D. Applications of Entropic Spanning Graphs. IEEE Signal Processing Magazine **19** (5) (2002) 85-85.
11. Jiang, X., Münger, A., Bunke, H.: On Median Graphs: Properties, Algorithms, and Applications. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 23, No. 10 (2001) 1144-1151
12. Kondor, R.I., Lafferty, J.: Diffusion Kernels on Graphs and other Discrete Input Spaces. In: Sammut, C. el al (Eds.): ICML 2002. Morgan Kaufmann (2002) 315–322
13. Lafferty, J., Lebanon, G.: Diffusion Kernels on Statistical Manifolds. CMU Technical Report CMU-04-101 (2004)
14. Lozano, M.A., Escolano, F.: EM Algorithm for Clustering an Ensemble of Graphs with Comb Matching. In Rangarajan, A. et al (Eds.) LNCS **2683** (2003) 52–67
15. Lozano, M.A., Escolano, F.: A Significant Improvement of Softassing with Diffusion Kernels. In A. Fred et al, (Eds.): SSPR&SPR 2004, LNCS **3138** (2004) 76–84.
16. Lozano, M.A., Escolano, F.: Regularization Kernels and Softassign. In Sanfeliu, A. et al, (Eds.): CIAPR 2004, LNCS **3287** (2004) 321–328
17. Lozano, M.A., Escolano, F.: Structural Recognition with Kernelized Softassign. In Lemaître, C. et al (Eds.): IBERAMIA 2004, LNCS **3315** (2004) 626–635.
18. Luo, B., Hancock, E.R.: Structural Graph Matching Using the EM Algorithm and Singular Value Decomposition. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 23, No. 10 (2001) 1120-1136.
19. Luo,B., Wilson, R.C, Hancock,E.R.: A Spectral Approach to Learning Structural Variations in Graphs. In: Crowley,J.L. et al (Eds.): ICVS 2003, LNCS **2626** (2003) 407–417
20. Pelillo, M.: Replicator Equations, Maximal Cliques, and Graph Isomorphism. Neural Computation **11** (1999) 1933–1955
21. Rényi, A. On Measures of Entropy and Information. In Proc. 4th Berkeley Symp. Math. Stat. and Prob. **1** (1961) 547–561
22. Robles-Kelly, A., Hancock, E.R.: Graph Matching Using Spectral Seriation. In Rangarajan, A. et al (Eds.): EMMCVPR 2003, LNCS **2683** (2003) 517–532.
23. Sanfeliu,A., Fu, K.S.: A Distance Measure Between Attributed Relational Graphs for Pattern Recognition. IEEE Transactions on Systems, Man, and Cybernetics **13** (1983) 353–362
24. Serratosa, F., Alquézar, R., Sanfeliu, A.: Function-described graphs for modelling objects represented by sets of attributed graphs, Pattern Recognition, Vol. 23, No. 3 (2003) 781-798
25. Smola, A., Kondor, R.I.: Kernels and Regularization on Graphs. In: Schölkopf, et al, (Eds.): COLT 2003, LNCS **2777** (2003) 144–158
26. Wilson, R.C., Hancock, E.R.:Structual Matching by Discrete Relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (6) (1997) 634–648

Edit Distance Based Kernel Functions for Attributed Graph Matching

Michel Neuhaus and Horst Bunke

Dept. of Computer Science, University of Bern,
Neubrückstrasse 10, CH-3012 Bern, Switzerland
{mneuhaus, bunke}@iam.unibe.ch

Abstract. In this paper we propose the use of a simple kernel function based on the graph edit distance. The kernel function allows us to apply a wide range of statistical algorithms to the problem of attributed graph matching. The function we describe is simple to compute and leads to several convenient interpretations of geometric properties of graphs in their implicit vector space representation. Although the function is not generally positive definite, we show in experiments on real-world data that the kernel approach may result in a significant improvement of the graph matching and classification performance using support vector machines and kernel principal component analysis.

1 Introduction

In recent years results from statistical learning theory [1] have successfully been applied to various pattern recognition problems. The Support Vector Machine (SVM) classifier, among other kernel machines, has performed particularly well on a number of data sets [2, 3]. The development of specific kernel functions such as diffusion kernels, convolutional kernels, and marginalized kernels lead to the application of kernel machines to the structural domain of graphs and strings [4, 5, 6, 7, 8]. In constrast to feature vectors, graphs allow for a more powerful representation of structured data. To measure the similarity of graphs, various formalisms have been proposed, ranging from exact methods such as subgraph isomorphism and maximum common subgraph computation to error-tolerant methods based on continuous optimization theory and the spectral decomposition of graph matrices [9, 10, 11, 12, 13]. Another graph matching approach that has become quite popular is the concept of the graph edit distance [14, 15, 16]. In graph edit distance, the main idea is to model graph difference through a sequence of edit operations, which leads to a dissimilarity measure on graphs.

In the present paper, we propose to use kernel functions based on the edit distance to classify graphs, instead of performing a simple nearest-neighbor classification in the original graph space. In Sect. 2 and 3, graph edit distance and kernel machines are briefly reviewed. The proposed kernel function is described in Sect. 4. Experimental results are presented in Sect. 5, and some concluding remarks follow in Sect. 6.

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 352–361, 2005.

2 Graph Edit Distance

In graph matching, patterns represented by graphs are classified based on their structural similarity. For this purpose, several similarity measures have been defined for different classes of graphs. In the following, we consider the case of numerically labeled graphs. That is, nodes and edges may contain vector attributes to make the graph representation more powerful. One of the most common similarity measures for general attributed graphs is the graph edit distance [14, 15, 16]. The key idea is to describe variations in the structure of graphs by sequences of basic edit operations. A standard set of edit operations consists of insertion, deletion, and substitution operations for nodes and edges. To allow for application-specific similarity measures, it is common to introduce edit costs for edit operations, so that the edit distance of two graphs can be defined by the least expensive sequence of edit operations, or edit path, that transforms one graph into the other.

After computing distances of an input graph to a number of known graphs, the classification of the input pattern is often performed by means of a nearest-neighbor approach, where the neighborhood of every known graph, according to the edit distance, is associated with its class. Theoretical results and practical experiments [17, 18] suggest that nearest-neighbor algorithms are appropriate for pattern recognition problems of various difficulty. Yet, the distances of the input graph to some or all of the known graphs could also be regarded as a vector representation of the graph, thus embedding the input graph in a vector space. As a major advantage, a vector space representation allows us to apply a large number of powerful statistical algorithms to the classification problem. In the remainder of this paper, we will explore the feasibility of using simple kernel functions to apply such algorithms to attributed graphs. In this context, the vector space embedding of the graphs follows implicitly and need not be explicitly constructed.

3 Kernel Machines

The recent widespread use of kernel machines is primarily due to their successful application to various pattern recognition problems [2, 3]. Instead of solving a classification problem in its original domain, the basic idea is to map patterns into a Hilbert space (vector space with an inner product) and find a solution for the vector representation of the patterns [1, 19]. While linear algorithms in the original pattern space may be efficient and easy to compute, they are unable to take non-linear pattern relations into account. By using non-linear transformations when mapping from the original domain into the vector space, we obtain non-linear generalizations of well-known linear algorithms. Cover's theorem on the separability of patterns [20] states that a complex pattern classification problem cast non-linearly into a high-dimensional space is more likely to be linearly separable than in a low-dimensional space. A principal component analysis (PCA) in the original pattern space, for instance, will only detect linear struc-

tures, whereas after mapping the data non-linearly into a vector space, a PCA may lead to non-linear principal components in the original space.

It turns out that it is sufficient for many statistical algorithms to evaluate inner products of vectors instead of carrying out the actual mapping into the vector space. These algorithms include PCA, fisher discriminant analysis, and SVMs, among many others [19]. Due to the work by Vapnik [1], the theoretical learning, convergence, and generalization propertiesof SVMs are mathematically well founded and well-known. Numerous application results suggest that SVMs offer a good compromise between the ability to generalize well on unseen data and to correctly classify the training vectors.

For arbitrary pattern spaces X, the existence of a positive definite [21] kernel function $k : X \times X \rightarrow R$ mapping pairs of patterns to real numbers is sufficient for the existence of a corresponding Hilbert space H such that the inner product of two mapped patterns is equal to the kernel function evaluated on the two patterns before mapping [19, 21]. That is, the kernel function acts as a shortcut and returns the value of the inner product in H without the need of actually mapping the patterns. This substitution is sometimes referred to as the "kernel trick". Given any kernel machine, the algorithm is implicitly carried out in the accompanying Hilbert space, while in fact only the kernel function in the original space must be evaluated. Since the kernelized algorithms only require the evaluation of inner products of patterns, but need not know the patterns itself, it suffices to replace the inner product in the algorithms with the kernel function. Note that the property of positive definiteness has been derived for a number of functions [21].

4 Distance Kernels

Our objective is to define a kernel function based on the graph edit distance. In the following, let us assume that $\mathcal{G}$ denotes the space of attributed graphs and the function $d : \mathcal{G} \times \mathcal{G} \rightarrow R^+ \cup \{0\}$ denotes the graph edit distance. For a specific sample set of graphs, if the negative element-wise squared matrix of edit distances turns out to be conditionally positive definite [21], the function $k : \mathcal{G} \times \mathcal{G} \rightarrow R$, given by

$$k(x, x') = \frac{1}{2}\left(d(x, x_0)^2 + d(x_0, x')^2 - d(x, x')^2\right),$$

where $x_0 \in \mathcal{G}$ can be arbitrarily chosen, is known to be positive definite [21] and thus qualifies as a kernel function in the context of kernel machines. The existence of such a kernel function implies the existence of a mapping from the space of graphs $\mathcal{G}$ into a Hilbert space and a corresponding inner product. Furthermore, it allows us to derive geometric properties from the Hilbert space and apply them to the space of graphs.

The concept of Euclidean distances and angles, for instance, can easily be formulated for graphs in their vectorial representation. First of all, one can show that the kernel function k described above implies that the Euclidean distance of

two graphs in the vector space is equal to their respective edit distance. Hence the vector space preserves distances from the graph space. Kernel machines evaluating Euclidean distances will therefore operate, in fact, on graph edit distances. Next, the length of a graph in its vector representation is found to be equal to its edit distance to x_0. In the Euclidean space the length of a vector is defined by its distance to the origin, hence the graph $x_0 \in \mathcal{G}$ acts to a certain extent similarly to the zero vector in R^n. Also, considering angular properties we arrive at analogous observations. The angle between a graph and x_0 is undefined, and the angle any graph encloses with itself is 0. Moreover, two graphs are found to be orthogonal if the edit distances between the two graphs and x_0 form an orthogonal triangle. These observations show that the implicit vector space embedding of the graphs seems to be reasonable in terms of geometric properties.

In analogy to the case of inner products in Euclidean vector spaces, the kernel function defined on pairs of graphs can be considered a similarity measure: The larger its value, the higher the similarity of the two graphs. If we regard the edit distance function d for a moment as a metric distance measure in a vector space, the kernel function can be interpreted geometrically: The similarity of two graphs x and x' is then defined by the overhead imposed by going from x via x_0 to x' compared to going straight from x to x'. That is, if the indirect path is only slightly longer than the direct path, which is certainly true for graphs far away from each other, the similarity will be low. On the other hand, if the direct path is much shorter than the indirect path, which is true for graphs close to each other, the similarity will be high. An illustration of this obversation is provided in Fig. 1.

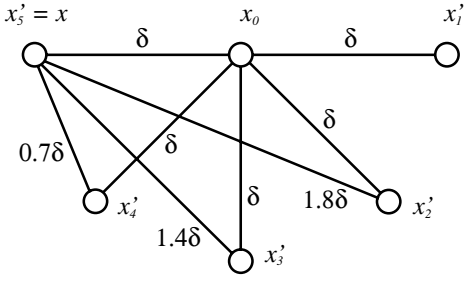


Fig. 1. Illustration of graph distance kernel. $k(x, x_1') = -\delta^2$, $k(x, x_2') = -0.7\delta^2$, $k(x, x_3') = 0$, $k(x, x_4') = 0.7\delta^2$, $k(x, x_5') = \delta^2$

The result of the kernel function is obviously strongly dependent on the choice of a graph x_0. A few possible kernels are given below, where k_{x_0} is used to denote the kernel with respect to graph x_0, depending on which property of the kernel function is to be emphasized:

$$k(x, x') = k_{x_{median}}(x, x') \tag{1}$$

$$k(x, x') = k_{x_{furthermost}}(x, x') \tag{2}$$

$$k(x, x') = k_{x_{empty}}(x, x') \tag{3}$$

$$k(x, x') = \sum\nolimits_{x_i} k_{x_i}(x, x') \tag{4}$$

$$k(x, x') = \prod\nolimits_{x_i} k_{x_i}(x, x'), \tag{5}$$

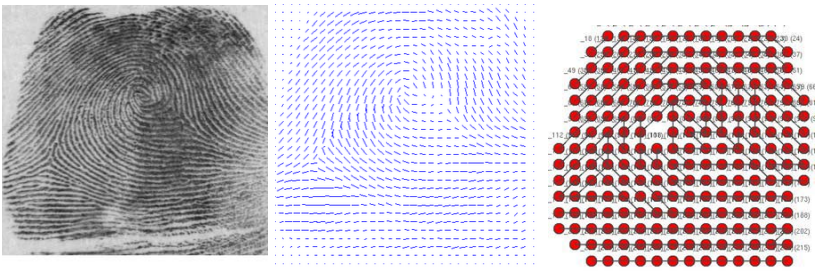
where x_{median}, $x_{furthermost}$, and x_{empty} denote the set median of the training graphs, that is, the training set graph with the smallest average distance to all other training set elements [22], the training set graph with the highest average distance to all other training set graphs, and the empty graph, respectively, and the summation and multiplication is done over all graphs from the training set.

For negative element-wise squared graph edit distance matrices the property of conditional positive definiteness is not generally satisfied. Yet, it is possible to construct a related positive definite matrix for every indefinite matrix such that some conditions are fulfilled [23]; for instance, such that only diagonal elements of the matrix are modified. The Modified Cholesky algorithm, naive Cholesky factorization, and methods based on the spectral decomposition of the kernel matrix are some of the most common of these algorithms.

5 Experimental Results

To evaluate the graph kernel approach, we apply the Kernel PCA and the SVM algorithm to the problem of fingerprint classification. In fingerprint classification, the objective is to divide fingerprints into classes based on characteristics of their global ridge flow. For our experiments we take 250 grayscale fingerprint images from the NIST-4 database [24] — 50 from each class *arch*, *tented arch*, *left loop*, *right loop*, and *whorl*. An example of a *whorl* image is shown in Fig. 2a. The images are converted into graphs using an approach similar to the segmentation of the orientation field proposed by Lumini et al. [25]. To separate the image foreground from the background, we compute the variance around each pixel and mark low-variance pixels as background. We proceed by applying a discrete Sobel operator to every pixel in order to obtain an estimate of the gradient of the ridge flow. After a smoothing procedure we obtain a ridge orientation field as illustrated in Fig. 2b. We eventually obtain a graph structure by representing pixel windows by nodes and the average direction of a window by edges. An illustration is provided in Fig. 2c. Nodes contain no attributes, but edges carry an angle attribute giving the average direction discretized to one out of the eight main directions.

The edit cost functions we use are defined as simple as possible: We assign constant costs c_n to node insertions and deletions, constant costs c_e to edge insertions and deletions, and zero costs to node substitutions (since all nodes are unlabeled). Edge substitution costs are set proportional to the absolute difference of the two involved edge angle attributes (taking the angular circularity into account). Note that no extensive optimization of the parameters is needed for this simple cost function.



a) b) c)

Fig. 2. a) Fingerprint image `f0011` from NIST-4, b) its average ridge orientation field, and c) the corresponding orientation graph

The 250 resulting graphs contain an average of 185 nodes and 206 edges per graph at a resolution of 32×32 pixels per window. Due to the large size of the graphs, the standard computation of the edit distance — which is only feasible for graphs with up to about 10 nodes — is intractable. Therefore, we employ an efficient approximate algorithm for the edit distance computation [26]. The approximate algorithm performs a locally optimal matching of the structure using an efficient node alignment technique with respect to edges, so that the resulting approximate distance provides an upper bound of the exact edit distance. Experiments show that the running time and memory requirements can thus be reduced by several orders of magnitude.

In our experiments we first perform a leave-one-out nearest-neighbor classification in the original graph space using the edit distance. That is, we classify every graph according to its closest neighbor in the training set and compute the overall recognition rate. This recognition rate of the leave-one-out classifier is used as a benchmark against which the proposed kernel methods are compared.

The result of the Kernel PCA computation is a set of principal components of the original graphs. In other words, the attributed graphs are transformed into principal components of the implicit vector space representation. Moreover, a weight is associated with every principal component that corresponds to the amount of variance that can be explained by the component. For an example, the first four principal components we obtain in our first experiment account for 15% of the variance. A dimensionality reduction of the resulting data can be achieved by dropping the least significant components. In the (reduced) vector space we can again compute the leave-one-out nearest-neighbor recognition rate based on the Euclidean distance.

Using the Median kernel and the Kernel PCA algorithm on the dataset described above, a leave-one-out recognition rate of up to 65% is obtained in the principal component space, while the corresponding leave-one-out recognition rate in the graph space amounts to 49%. In fact we observe that for most dimensions the classifier in the principal component space outperforms the original classifier by a margin of 10% and more. A graphical illustration of the Kernel PCA results is shown in Fig. 3. From this experiment, we conclude that the kernel

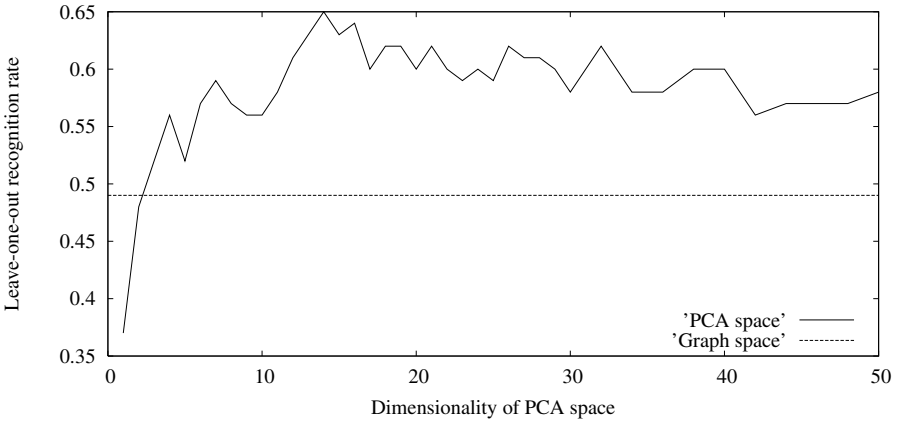


Fig. 3. Nearest-neighbor classifier performance in principal component space of graphs and in original graph space

function is able to detect relevant non-linear relations. Even a three-dimensional vector representation of the original graphs and the standard Euclidean distance instead of the edit distance leads to an improved recognition rate.

In our second experiment, we evaluate the performance of an SVM on the same dataset. The 250 fingerprint graphs are divided into a test set and a training set. In the case of the nearest-neighbor classifier in the graph space, the training set is the set of known prototypes, and the graphs from the test set are classified according to the nearest-neighbors from the training set. For large training sets, we expect the nearest-neighbor classifier to perform well, as it should be able to model arbitrarily complex decision functions. For smaller training sets, however, the nearest-neighbor prototypes might not cover the region of interest sufficiently well, particularly if classes are non-compact and overlapping. Similarly, a small training set may affect the ability of a SVM to generalize well on unseen patterns, for the resulting support vectors heavily depend on the position of single patterns. In order to evaluate the behavior of the nearest-neighbor classifier and the SVM, we vary the number of training and test samples: The test set is chosen to contain the first k graphs from every class and the training set to contain the remaining $250 - 5k$ graphs. Thus for $k = 1$, for instance, the test set contains a single graph from each of the 5 classes and the training set the remaining 245 graphs. Hence with an increasing number of test samples, the training set decreases at equal rate.

The performance of the nearest-neighbor classifier in the graph space and the SVM are evaluated for different numbers $k \in \{1, \ldots, 49\}$ of test elements per class. The resulting recognition rates are illustrated in Fig. 4. First we note that the SVM recognition rate on the training set increases as expected with a decreasing number of training samples, as it becomes easier to find class boundaries for smaller samples. Conversely, the nearest-neighbor recognition on the test set degrades when decreasing the number of known prototypes. The same tendency

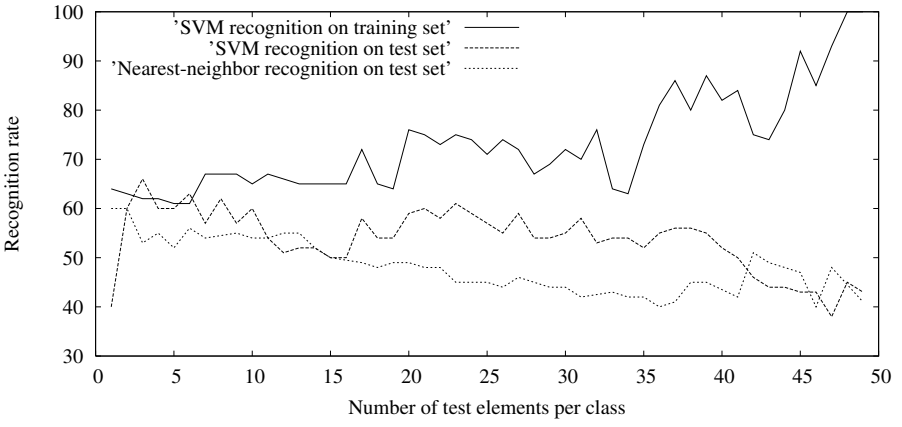


Fig. 4. Performance of nearest-neighbor classifier in graph space and SVM classifier

can also be observed in the case of the SVM recognition on the test set, but is less evident. The extreme cases of very few training elements and very few test elements are not representative. For the meaningful range of $k \in \{10, \ldots, 40\}$, the recognition rate obtained with the Median graph kernel SVM is inferior to that of the nearest-neighbor classifier in only 2 out of 31 cases.

In our third experiment, we apply the SVM classifier to the task of diatom classification. Diatoms are unicellular algae occurring in water and humid places on earth [27]. Their classification is an important, but difficult task in various disciplines such as environmental monitoring and climate research. We use a database consisting of diatom images from 22 classes containing 5 elements each. By means of a segmentation process, diatom images are transformed into attributed region adjacency graphs. The edit cost function reported in [28] defines node insertion and deletion costs proportional to the size of the corresponding region and substitution costs proportional to the Euclidean distance of attributes. In addition, splitting and merging operations of nodes can be carried out for free to account for over- and under-segmentation errors. Details about the graph representation and the extraction process are reported in [28]. Using a nearest-neighbor classifier in the graph space, we obtain a leave-one-out recognition rate of 71.8%, whereas the leave-one-out recognition rate of the SVM (sum kernel, Modified Cholesky algorithm) amounts to 85.0%. Hence we find that the SVM is able to outperform the best recognition rate reported for this dataset [28], even without optimizing parameters and using a simpler cost function (no node splitting and merging operations and constant insertion and deletion costs) than the one applied in [28].

6 Conclusions

In the present paper we describe a simple kernel function for attributed graphs that is based on the computation of the graph edit distance. A common approach

to using the edit distance for graph classification includes a nearest-neighbor approach of some sort, as the space of attributed graphs offers only a distance measure, but no additional information about the structure of the patterns in this space. Using a kernel function, this limitation can be overcome, since various statistical algorithms can be applied in an implicitly existing vector space where every graph is represented by a vector. The kernel function we describe is determined by the edit distance of graphs. It turns out that a number of geometric properties can be derived that suggest that the kernel function vector space embedding of the graphs is reasonable. To evaluate the kernel approach in comparison to a nearest-neighbor classifier, we perform a Kernel PCA and train an SVM on attributed graphs extracted from fingerprint images and graphs extracted from segmented diatom images. The resulting graph classification in the principal component space and the SVM classification outperform the nearest-neighbor classification in most cases. In the future, we intend to study the applicability of the proposed kernel function to other real-world graph and string data. Also, we would like to further investigate what properties a well-performing origin graph x_0 needs to satisfy.

Acknowledgment

This research was supported by the Swiss National Science Foundation NCCR program "Interactive Multimodal Information Management (IM)2" in the Individual Project "Multimedia Information Access and Content Protection".

References

1. Vapnik, V.: Statistical Learning Theory. John Wiley (1998)
2. Byun, H., Lee, S.: A survey on pattern recognition applications of support vector machines. Int. Journal of Pattern Recognition and Art. Intelligence **17** (2003) 459–486
3. Müller, K., Mika, S., Rätsch, G., Tsuda, K., Schölkopf, B.: An introduction to kernel-based learning algorithms. IEEE Transactions on Neural Networks **12** (2001) 181–202
4. Watkins, C.: Dynamic alignment kernels. In Smola, A., Bartlett, P., Schölkopf, B., Schuurmans, D., eds.: Advances in Large Margin Classifiers. MIT Press (2000) 39–50
5. Gärtner, T., Lloyd, J., Flach, P.: Kernels and distances for structured data. Machine Learning **57** (2004) 205–232
6. Haussler, D.: Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz (1999)
7. Kondor, R., Lafferty, J.: Diffusion kernels on graphs and other discrete input spaces. In: Proc. 19th Int. Conf. on Machine Learning. (2002) 315–322
8. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proc. 20th Int. Conf. on Machine Learning. (2003) 321–328
9. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters **19** (1998) 255–259

10. Wallis, W., Shoubridge, P., Kraetzl, M., Ray, D.: Graph distances using graph union. Pattern Recognition Letters **22** (2001) 701–704
11. Christmas, W., Kittler, J., Petrou, M.: Structural matching in computer vision using probabilistic relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence **17** (1995) 749–764
12. Wilson, R., Hancock, E.: Structural matching by discrete relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997) 634–648
13. Luo, B., Hancock, R.: Structural graph matching using the EM algorithm and singular value decomposition. IEEE Transactions on Pattern Analysis and Machine Intelligence **23** (2001) 1120–1136
14. Sanfeliu, A., Fu, K.: A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics **13** (1983) 353–363
15. Myers, R., Wilson, R., Hancock, E.: Bayesian graph edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (2000) 628–635
16. Messmer, B., Bunke, H.: A new algorithm for error-tolerant subgraph isomorphism detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998) 493–504
17. Duda, R., Hart, P., Stork, D.: Pattern Classification. John Wiley (2000)
18. Mansilla, E., Ho, T.: On classifier domains of competence. In: Proc. 17th Int. Conf. on Pattern Recognition. Volume 1. (2004) 136–139
19. Schölkopf, B., Smolar, A.: Learning With Kernels. MIT Press (2002)
20. Cover, T.: Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Trans. on Electronic Computers **14** (1965) 326–334
21. Berg, C., Christensen, J., Ressel, P.: Harmonic Analysis on Semigroups. Springer (1984)
22. Jiang, X., Münger, A., Bunke, H.: On median graphs: Properties, algorithms, and applications. IEEE Transactions on Pattern Analysis and Machine Intelligence **23** (2001) 1144–1151
23. Gill, P., Murray, W., Wright, M.: Practical Optimization. Academic Press (1981)
24. Watson, C., Wilson, C.: NIST Special Database 4. Fingerprint Database. National Institute of Standards and Technology (1992)
25. Lumini, A., Maio, D., Maltoni, D.: Inexact graph matching for fingerprint classification. Machine Graphics and Vision, Special Issue on Graph Transformations in Pattern Generation and CAD **8** (1999) 231–248
26. Neuhaus, M., Bunke, H.: An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In: Proc. 10th Int. Workshop on Structural and Syntactic Pattern Recognition. LNCS 3138, Springer (2004) 180–189
27. du Buf, H., Bayer, M., eds.: Automatic Diatom Identification. World Scientific (2002)
28. Ambauen, R., Fischer, S., Bunke, H.: Graph edit distance with node splitting and merging and its application to diatom identification. In: Proc. 4th Int. Workshop on Graph Based Representations in Pattern Recognition. LNCS 2726 (2003) 95–106

A Robust Graph Partition Method from the Path-Weighted Adjacency Matrix

Huaijun Qiu and Edwin R. Hancock

Department of Computer Science, University of York,
Heslington, York, YO10 5DD, UK

Abstract. In this paper we develop a new graph representation based on the path-weighted adjacency matrix for characterising global graph structure. The representation is derived from the heat-kernel of the graph. We investigate whether the path-weighted adjacency matrix can be used for the problem of graph partitioning. Here we demonstrate that the method out-performs the use of the adjacency matrix. The main advantage of the new method is that it both preserves partition consistency and shows improved stability to structural error.

1 Introduction

Many combinatorial problems in computer vision can be posed as locating a partition of the vertices of a given graph into subsets that satisfy mutual consistency constraints. The problem arrises in electronic circuit design, parallel processing [15], graph coloring [16], clustering [12] and segmentation [14, 10]. As a concrete example from the computer vision and pattern recognition literature, Shi and Malik [14] have shown how the image segmentation problem can be posed as one of graph partition and have shown how the normalized cut method can be used to locate approximate solutions. Pavan and Pelillo [10] improve the partition measure by introducing the concept of a dominant set. The resulting utility measure is optimised using a relaxation scheme, and is applied to the problems of graph clustering and image segmentation. In prior work [11, 12], aimed at realizing inexact graph matching using a subgraph decomposition, we have shown how graphs can be partitioned into non-overlapping subunits using the Fiedler vector, i.e. the eigenvector associated with the second smallest eigenvalue of the Laplacian matrix. In this paper we aim to investigate whether this method can be improved using a more robust partition measure motivated by the study of the heat-kernel of the graph [6].

Kernel-based methods have been widely used for pattern recognition and have lead to the development of a number of methods including support vector machines [5] and kernel PCA [13]. For instance, Watkins et al. [7] have used a modified string kernel for categorizing text documents. Kondor and Lafferty [6] have surveyed the more general class of exponential kernels, which are applicable to a wide variety of symbolic or discrete entities. Of the alternatives, one of the most important is the heat kernel, which is found by solving the diffusion

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 362–372, 2005.

equation for the discrete structure in-hand. The heat kernel is an important analytical tool, which has been used in many areas. In spectral graph theory, Chung [4, 3] has summarised some of the important properties of heat kernel. Belkin and Niyogi [2] have used the connections between the Laplacian and the heat kernel to embed patterns in a high dimensional space for the purposes of visualisation and clustering.

The aim in this paper is to investigate whether the information conveyed by the heat kernel can be used for characterising the structure of graphs. We seek a more globalized graph representative for the purpose of graph simplification and graph clustering than can be achieved using the adjacency matrix alone. We use the heat-kernel to construct a path-weighted adjacency matrix to represent the graph structure. The weighting process aims to smooth away the effects of structural error due to node or edge deletions. We explore whether this representation can be used to characterise the graph globally and whether it is stable under structural error. In particular, we explore its use in conjunction with our recently reported graph partition method [11, 12].

2 Weighted Laplacian Matrix and Normalized Laplacian Matrix

Let the weighted graph G be the quadruple (V, E, Ω, ω), where V is the set of nodes, E is the set of arcs, $\Omega = \{W_u, \forall u \in V\}$ is a set of weights associated with the nodes and $\omega = \{w_{u,v}, \forall (u,v) \in E\}$ is a set of weights associated with the edges. Further let $D = diag(d_v; v \in V(G))$ be the diagonal weighted degree matrix with $D_u = \sum_{v=1}^{n} w_{u,v}$. The un-normalised weighted Laplacian matrix is given by $L = D - A$, and has elements given by

$$L_{uv}(G) = \begin{cases} \sum_{\langle u,k\rangle \in E} w_{u,k} & \text{if } u = v \\ -w_{u,v} & \text{if } u \neq v \text{ and } (u,v) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The normalized weighted Laplacian matrix on the other hand is defined to be $\hat{L} = D^{-1/2} L D^{-1/2}$, and has elements

$$\hat{L}_{uv}(G) = \begin{cases} 1 & \text{if } u = v \\ -\frac{w_{u,v}}{\sqrt{d_u d_v}} & \text{if } u \neq v \text{ and } (u,v) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The normalised Laplacian $\hat{L}$ can also be viewed as a harmonic operator that acts on the function $f : V(G) \mapsto \Re$ with the result that $\hat{L}f(x) = \sum_{x'} \hat{L}_{x,x'} f(x')$. The spectral decomposition of the normalised Laplacian matrix is $\hat{L} = \Phi \Lambda \Phi^T = \sum_{v=1}^{|V|} \lambda_v \phi_v \phi_v^T$. where $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_{|V|})$ is the diagonal matrix with the ordered eigenvalues as elements and $\Phi = (\phi_1 | \phi_2 | .... | \phi_{|V|})$ is the matrix with the ordered eigenvectors as columns.

3 Heat Kernel and the Path-Weighted Matrix

We are interested in the heat equation associated with the Laplacian, i.e. $\frac{\partial H_t}{\partial t} = -\hat{L}H_t$ where H_t is the heat kernel and t is time. The solution is found by exponentiating the Laplacian eigenspectrum i.e. $H_t = \Phi \exp[-t\Lambda]\Phi^T$. The heat kernel is a $|V| \times |V|$ matrix, and for the nodes u and v of the graph G the resulting component is

$$H_t(u,v) = \sum_{i=1}^{|V|} \exp[-\lambda_i t]\phi_i(u)\phi_i(v) \tag{3}$$

3.1 Path Length Distribution

Let us consider the matrix $P = I - \hat{L}$, where I is the identity matrix, then the heat kernel can be rewritten as $H_t = e^{-t(I-P)}$. We can perform a McLaurin expansion on the heat-kernel to re-express it as a polynomial in t. The result of this expansion is

$$\begin{aligned} H_t &= e^{-t(I-P)} \\ &= e^{-t}\left(I + tP + \frac{(tP)^2}{2!} + \frac{(tP)^3}{3!} + \cdots\right) \\ &= e^{-t}\sum_r P^r \frac{t^r}{r!} \end{aligned}$$

We have used the eigendecomposition of the normalised Laplacian $P^r = (I - \hat{L})^r = \Phi(I-\Lambda)^r\Phi^T$ and as a result

$$P^r(u,v) = \sum_i (1-\lambda_i)^r \phi_i(u)\phi_i(v) \tag{4}$$

If, on the other hand, we consider the element-wise definition of P

$$P(u,v) = \begin{cases} 1 & \text{if } u = v \\ \frac{w_{u,v}}{\sqrt{d_u d_v}} & \text{if } u \neq v \text{ and } (u,v) \in E \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

then we have that

$$P^r(u,v) = \sum_{s_r} \prod_i \frac{w(u_i, u_{i+1})}{\sqrt{d_{u_i} d_{u_{i+1}}}} \tag{6}$$

Hence, P^r can be interpreted as the sum of weights of all walks of length r joining nodes u and v. A walk S_r is a sequence of vertices $u_0, \cdots, u_r$ such that $u_i = u_{i+1}$ or $(u_i, u_{i+1}) \in E$. By defining $P(u,u) = 1$, we create a self-loop for each node on the walk. So the walk can pause on any node for a number of steps before the next move. This gives us better behaved distribution of P^r over the path length r. Here the definition of $P(u,u) = 1$ is important because it allows self-loops in the adjacency matrix. In this paper, we aim to exploit the fact that the matrix P^r contains information concerning the inter-node distance distribution to construct a measure that can be used to partition graphs.

3.2 Proximity Weights

Our idea is to use the distribution of distances to compute the average path-length between pairs of nodes in the graph. For the nodes u and v the average path-length is given by

$$\hat{d}(u,v) = \frac{\sum_r rP^r(u,v)}{\sum_r P^r(u,v)} \tag{7}$$

This average distance measure can be used to compute a Gaussian weighted node proximity matrix. For the nodes u and v the proximity weight is given by *path-weighted matrix*.

$$W_{uv}(G) = exp\left[-\frac{\hat{d}^2(u,v)}{2\sigma^2(u,v)}\right] \tag{8}$$

where

$$\sigma^2(u,v) = \frac{\sum_r (r-\hat{d}(u,v))^2 P^r(u,v)}{\sum_r P^r(u,v)} \tag{9}$$

is the variance of the path-length distribution for nodes u and v.

3.3 Properties of the Proximity Matrix

The proximity matrix defined in the previous section has some interesting properties that distinguish it from the raw adjacency matrix. Here we focus on some of these in detail.

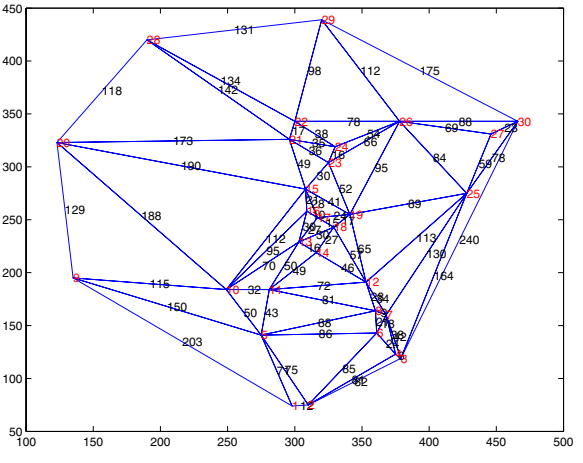
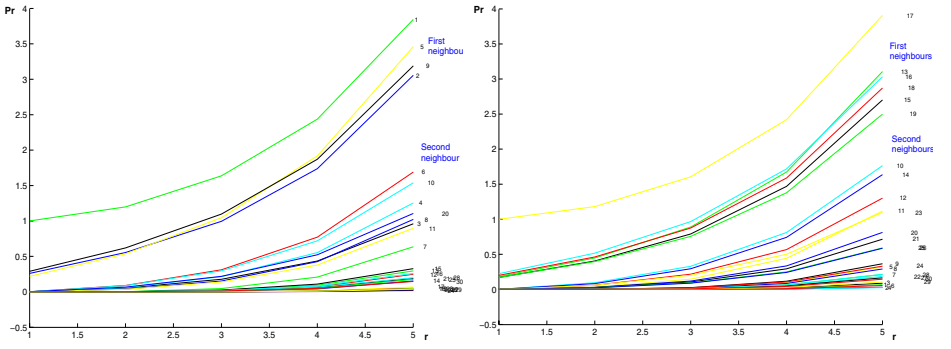


Fig. 1. Delaunay graph from detected corner triangulations

Firstly, although the adjacency matrix may contain a significant number of zero off-diagonal entries, provided that the graph under study is connected, then the path-length proximity matrix will not have zero off-diagonal entries since a path of finite length can always be located between a pair of nodes. The consequence of this is that path-length proximity matrix will be less likely to be singular or to have a zero determinant.

Second, nodes which have a similar locations with respect to the boundary of the graph will have similar path-weight values. Since the matrix is constructed using node distance, the nodes on the boundary will have different values to



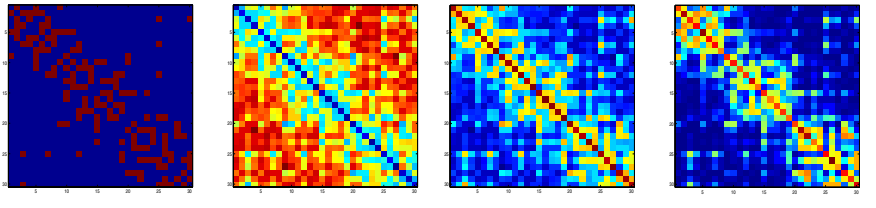
(a) Path length distribution of node 01 (b) Path length distribution of node 17

Fig. 2. Distribution of P^r based on the path step r

those near the centre of the graph. This means that the measure could be useful for the purposes of assigning node affinity in the problem of graph-matching.

Figure 1 shows a Delaunay triangulation as an illustration of the points mentioned above. Figure 2(a) and Figure 2(b) show $P^r(u, v)$ as a function of r for the nodes labelled 1 and 17 in the Delaunay graph. The different curves are obtained when v runs over the remaining nodes of the graph, and are labelled with node number. From the figure we can see that $P^r(u, u)$ always takes on the largest value, irrespective of r, since it counts the number of loops of length r to node u. The remaining curves are ordered in descending order according to whether nodes are first, second or third etc. neighbours. The most distant nodes are associated with the smallest values of $P^r(u, v)$. Another important property is that the nodes in the interior of the graphs always have larger values of $P^r(u, v)$ than those on or near the boundary.

Finally, we note that when compared to the binary adjacency matrix, the path-weighted proximity matrix is more robust to changes in graph structure. To illustrate this point consider the deletion of an edge. In the case of the adjacency



(a) Adjancency matrix (b) Distance matrix $\hat{d}$ (c) Delta matrix δ (d) Path-weighted matrix W_G

Fig. 3. Similarity matrices

matrix, two symmetrically placed elements flip from one to zero. Hence, all memory of the edge is lost. However, in the case of the path-weighted proximity matrix the mean distance between the nodes is increased.

For the graph shown above, in Figure 3 the four panels show the adjacency matrix, the matrix of path weighted distances $d(u, v)$, the path length variance $\sigma(u, v)$ and the path weighted proximity matrix $W_u v(G)$. The entries in the adjacency matrix correspond to maxima in the weight matrix.

4 Graph Partition Method

Generally speaking, graph partition aims to decompose a graph into non-overlapping sets of nodes so as to minimize a cost function.

In a recent paper [11], we addressed the problem of partitioning graphs into non-overlapping *supercliques*. The superclique of the node i consists of its center node, together with its immediate neighbours N_i connected by edges in the graph, i.e., $C_i = \{i\} \cup \{u; (i, u) \in E\}$. Our next step is to partition the graph into a set of non-overlapping supercliques using the node order defined by the *Fiedler vector*. The Fiedler vector is the eigenvector corresponding to the second smallest eigenvalue of the Laplacian matrix. Instead of using the original adjacency matrix, we use the path-weighted matrix since it contains more global information concerning the structure of the graph than the adjacency matrix which contains information concerning only its immediate neighbours.

We commence by assigning weights to the nodes on the basis of their rank-order in the path defined by the Fiedler vector. We first sort the nodes in descending order according to the magnitude of the corresponding component of the Fiedler vector. The order is given by $\Pi^* = \{\pi_1, \pi_2, \pi_3, \ldots, \pi_n\}^T$ with $\pi_i > \pi_{i+1}$. Having the vector sorted, we assign weights on the nodes based on their rank order $\Psi_i = Rank(\pi_i)$. With this weighted graph in hand, we can gauge the significance of each node using the score function: $\Theta_i = \alpha\,(Deg(i) + |N_i \cap \Phi|) + \frac{\beta}{\Psi_i}$. where Φ is the set of nodes on the perimeter of the graph. The first term depends on the degree of the node and its proximity to the perimeter. Hence, it will sort nodes according to their distance

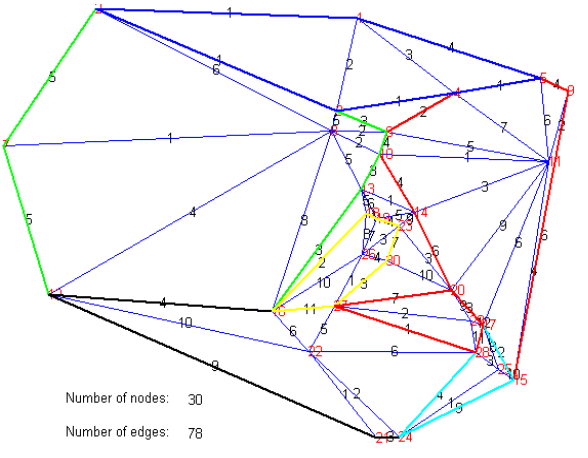


Fig. 4. Graph Partition

from the perimeter. This will allow us to partition nodes from the outer layer first and then work inwards. The second term ensures that the first ranked nodes in the Fielder vector are visited first.

Let $\Gamma =< j_1, j_2, j_3, ... >$ be the rank-order of the nodes as defined by the Fiedler vector, i.e. $\pi(j_1) < \pi(j_2) < \pi(j_3).....\pi(j_{|V|})$. We traverse this list until we find a node k_1 which is neither in the perimeter, i.e. $k_1 \notin \Phi$ and whose score exceeds those of its neighbours, i.e. $\Theta_{k_1} = \arg\max_{i \in k_1 \cup N_{k_1}} \Theta_i$. When this condition is satisfied, then the node k_1 together with its neighbours N_{k_1} represent the first superclique. The set of nodes $C_{k_1} = k_1 \cup N_{k_1}$ are appended to a list A that tracks the set of nodes assigned to supercliques. This process is repeated for all the nodes which have not yet been assigned to supercliques i.e. $R = \Gamma - A$. The procedure terminates when all the nodes of the graph have been assigned to non-overlapping supercliques. An example performed on figure 1 is shown at Figure 4.

5 Experiments and Comparisons

Our aim in this section is to explore how the path-weighted proximity matrix can be used for the purposes of graph partition, and to determine whether it can render the process more robust to structural error.

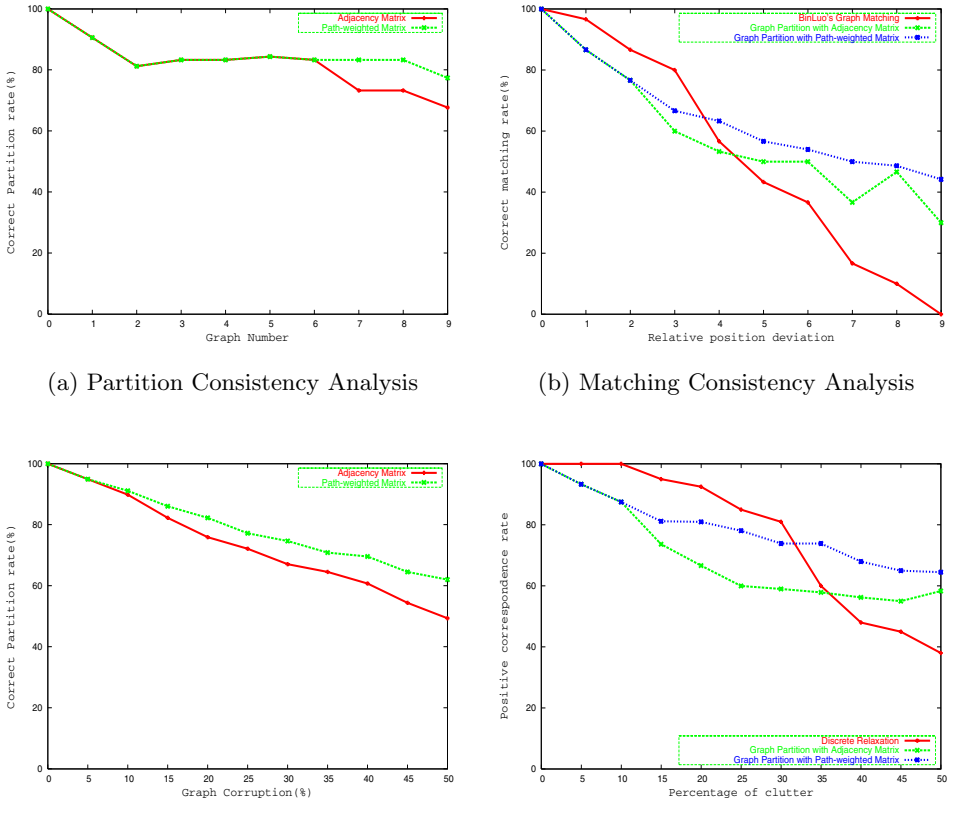
There are two aspects to our study. We commence by investigating the difference in the partitions obtained with the adjacency and path-weighted proximity matrix. Second, we perform a sensitivity study to compare the robustance of the partitions under node or edge deletions from both the graphs.

The graphs used here are extracted from the CMU model-house sequence. This sequence is made up of a series of images which have been captured from different viewpoints. Our graphs are the Delaunay triangulations of the corner-features detected using the Luo, Cross and Hancock [8] method.

5.1 Partition and Matching Consistency Analysis

Since our graphs represent a series with similar structure, they should share a similar partition arrangements. This is important since if we are to use the partitions for graph-matching, then they must be stable. The more similar two partitions, the better the matching result will be. Our aim here is to check which matrix-representation preserves the partition consistency better. We use the partition of the first graph as the model pattern and compare with the partitions of the remaining graphs in the sequence.

Figure 5(a) we show the fraction of edges that remain in the same partition as a function of the difference in view number. The green curve shows the result obtained using the path-weighted proximity matrix, while the red curve shows the result obtained with the adjacency matrix. For large differences in view number, i.e. when the structural differences are greatest, then the path weighted proximity seems to be more stable than the adjacency matrix.



(a) Partition Consistency Analysis

(b) Matching Consistency Analysis

(c) Partition Stability Analysis

(d) Matching Stability Analysis

Fig. 5. Partition and Matching Analysis

In Figure 5(b) we show the fraction of correct matches as a function of difference in view number. The blue curve, which represents the path-weighted proximity matrix, outperforms the green one from the adjacency matrix and the red one which is the result of the EM graph matching method described [9].

5.2 Partition and Matching Stability Analysis

In this subsection we aim to measure the sensitivity of our graph partition method to structural error, and compare the results obtained with the path-weighted proximity matrix and the adjacency matrix.

The effects of structural error are simulated by randomly deleting nodes or edges from the graphs under study. Figure 5(c) shows fraction of nodes that remain in the same partition as the graph shown in Figure 4 is subjected to increasing corruption. The graph corruption rate is defined to be the number of deleted edges divided by the total number of original edges. As the level of corruption is increased, then the path-weighted proximity matrix outperforms the adjacency

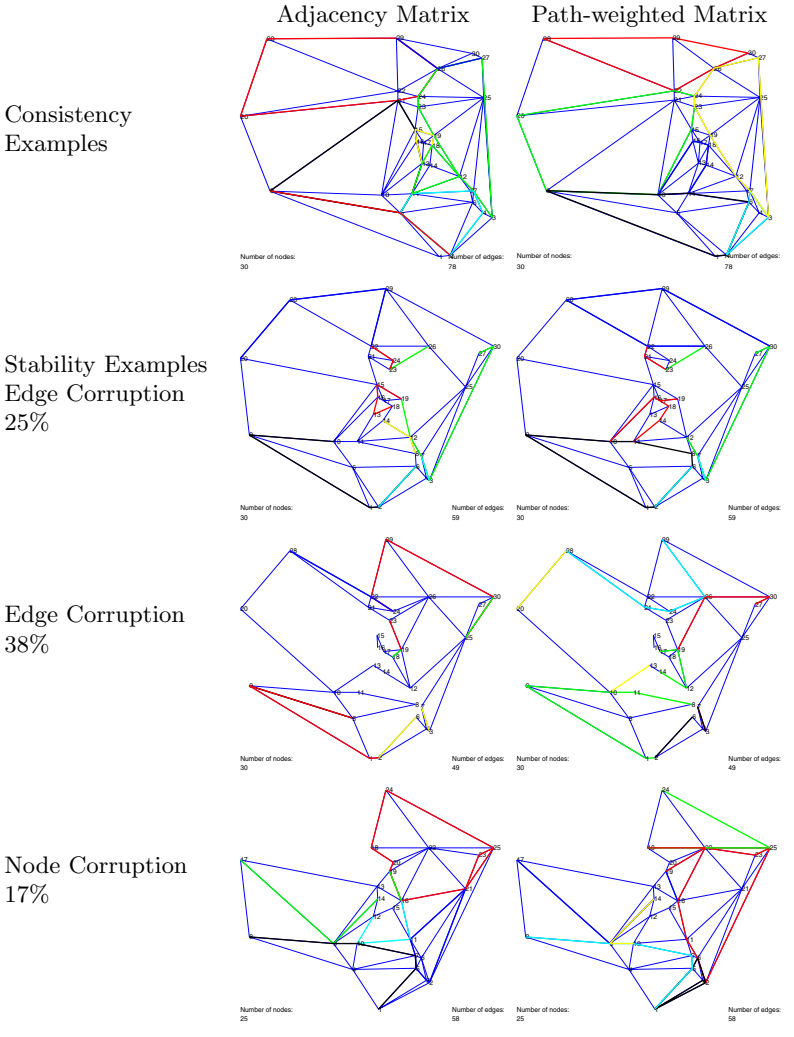


Fig. 6. Examples of the partitions

matrix in terms of partition stability. This means that the path-weighted proximity matrix better preserves the partition structure and is more stable under structural error. This stability property has knock-on effects for the performance of the graph-matching method. In Figure 5(d) we show the performance of the matching process as the fraction of corruption is increased. Here the red curve is the result of the original discrete relaxation scheme [9], the blue curve that obtained when we apply spectral partitioning to the adjacency matrix [12], and the blue curve the result when we apply spectral partitioning to the path-weighted adjacency matrix. For large levels of corruption, the results obtained using the path-weighted adjacency matrix outperform those obtained using the alternative methods.

Finally, we provide some examples to illustrate the stability of the partitions obtained. In the left hand column, we show the partitions obtained using the adjacency matrix while the right-hand column shows the partitions obtained using the path-weighted adjacency matrix. The differently coloured edges of the graph indicate the different partitions obtained by the two methods. In the top row of Figure 6 we show the partitions of the graph shown in Figure 4, and here the result obtained using the path-weighted adjacency matrix is closer to the original than that delivered by the adjacency matrix. The remaining rows in Figure 6 show the effect of graph-corruption on the partitions. Rows 2 and 3 show the effect of different levels of edge corruption, and Row 4 the effect of node corruption. In all case the path-weighted adjacency matrix is more stable than the adjacency matrix.

6 Conclusions

In this paper, we have shown how ideas from the spectral theory of the heat kernel can be used to construct a path-weighted proximity matrix. We have studied its properties and this shows that it gives us more stable representation of graph-structure under structural error. Our future plans are to extend the work in two directions. First, we aim to use the existing method for other graph partition problems including clustering and image segmentation. Second, we aim to use alternative methods for re-weighting the adjacency matrix. For instance, we could use alternative statistics of the path length distribution or modify the distribution using our recently reported heat-kernel method for graph embedding [1].

References

1. X. Bai and E. R. Hancock. Heat kernels, manifolds and graph embedding. SSPR, 2004.
2. M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*, pages 585–591, 2002.
3. F. R. K. Chung and S.-T. Yau. Coverings, heat kernels and spanning trees. *The Electronic Journal of Combinatorics*, 6, 1999.
4. F.R.K. Chung. *Spectral Graph Theory.* CBMS series 92. American Mathmatical Society Ed., 1997.
5. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.
6. R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. *19th Intl. Conf. on Machine Learning (ICML) [ICM02].*
7. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *The Journal of Machine Learning Research*, 2:419–444, 2002.
8. B. Luo, A. D. Cross, and E. R. Hancock. Corner detection via topographic analysis of vector potential. *Proceedings of the 9 th British Machine Vision Conference*, 1998.

9. B. Luo and E. R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *IEEE PAMI*, 23(10):1120–1136, 2001.
10. M. Pavan and M. Pelillo. Dominant sets and hierarchical clustering. pages 362–369. ICCV, 2003.
11. H. Qiu and E.R. Hancock. Graph partition for mathing. Graph Based Representations in Pattern Recognition, 2003.
12. H. Qiu and E.R. Hancock. Spectral simplification of graphs. ECCV, 2004.
13. B. Sch, A. Smola, and K. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.
14. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE PAMI*, 22(8):888–905, 2000.
15. H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2:135–148, 1991.
16. R.J. Wilson and John J. Watkins. *Graphs: an introductory approach: a first course in discrete mathematics*. Wiley international edition. New York, etc., Wiley, 1990.

Recent Results on Heat Kernel Embedding of Graphs

Xiao Bai and Edwin R. Hancock

Department of Computer Science, University of York, UK

Abstract. This paper describes how heat-kernel asymptotics can be used to compute approximate Euclidean distances between nodes in a graph. The distances are used to embed the graph-nodes in a low-dimensional space by performing Multidimensional Scaling(MDS). We perform an analysis of the distances, and demonstrate that they are related to the sectional curvature of the connecting geodesic on the manifold. Experiments with moment invariants computed from the embedded points show that they can be used for graph clustering.

1 Introduction

The spectrum of the Laplacian matrix has been widely studied in spectral graph theory [4] and has proved to be a versatile mathematical tool that can be put to many practical applications including routing [1], clustering [10] and graph-matching [13, 9]. One of the most important properties of the Laplacian spectrum is its close relationship with the heat equation. The heat equation can be used to specify the flow of information with time across a network or a manifold [12]. Its solution is obtained by exponentiating the Laplacian eigensystem. For a Riemannian manifold, the heat kernel is determined by pattern of geodesic distances, and can provide a means of analysing both the local and global differential geometry of the manifold. In particular, differential invariants can be computed from the heat kernel, and these in turn are related to the Laplacian eigensystem. This field of study is sometimes referred to as spectral geometry and has close links with K-theory and Morse theory. It has topical interest in particle physics, since Witten [3] has shown how such theories can be used to understand the geometries of space-time that underpin superstrings.

There are a number of different invariants that can be computed from the heat-kernel. Asymptotically for small time, the trace of the heat kernel [4] (or the sum of the Laplacian eigenvalues exponentiated with time) can be expanded as a rational polynomial in time, and the co-efficients of the leading terms in the series are directly related to the geometry of the manifold. For instance, the leading co-efficient is the volume of the manifold, the second co-efficient is related to the Euler characteristic, and the third co-efficient to the Ricci curvature. The zeta-function (i.e. the sum of the eigenvalues raised to a non-integer power) for the Laplacian also contains geometric information. For instance its derivative at

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 373–382, 2005.

the origin is related to the torsion tensor. Finally, Colin de Verdiere has shown how to compute geodesic invariants from the Laplacian spectrum [2].

In recent papers [17, 15], we have shown how heat-kernel asympotics can be used to compute Euclidean distances between pairs of nodes in a graph. These geodesic distances can be used to embed the nodes of the graph in a low dimensional space using multidimensional scaling. Once embedded in such a space then graph matching may be effected as point-set alignment[16], and graph clustering can be realised using features extracted from the point-set distribution[15]. The aim in this paper is to take the study several steps further. In particular, we focus on the theoretical question of how the geometry of the manifold relates to the spectrum of the heat kernel, and the practical question of how moment invariants for the embedded points can be used to characterise and cluster graphs.

The outline of this paper is as follows. We commence in Section 2 by describing our method for computing Euclidean distances. Section 3 uses results from spectral geometry to analyse the geodesics distances, and describes their relationship to the spectral invariants of the manifold. In Section 4 we describe how the distances may be used to embed points in a low-dimensional space and how the distribution of points characterised using moment invariants. Experiments on graphs extracted from the COIL database are presented in Section 5. Finally, Section 6 offers some conclusions and discusses directions for future work.

2 Heat Kernels and Riemannian Manifolds

In this section, we develop a method for approximating the geodesic distance between nodes by exploiting the properties of the heat kernel. To commence, suppose that the graph under study is denoted by $G = (V, E)$ where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. Since we wish to adopt a graph-spectral approach we introduce the adjacency matrix A for the graph where

$$A(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

We also construct the diagonal degree matrix D, whose elements are given by $D(u, u) = \sum_{v \in V} A(u, v)$. From the degree matrix and the adjacency matrix we construct the Laplacian matrix $L = D - A$. The normalised Laplacian is given by $\hat{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$.

Spectral Analysis: The spectral decomposition of the normalised Laplacian matrix is

$$\hat{L} = \Phi \Lambda \Phi^T = \sum_{i=1}^{|V|} \lambda_i \phi_i \phi_i^T \tag{2}$$

where $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_{|V|})$ is the diagonal matrix with the ordered eigenvalues as elements and $\Phi = (\phi_1 | \phi_2 | .... | \phi_{|V|})$ is the matrix with the ordered eigenvectors as columns. The eigenvector associated with the smallest non-zero eigenvector is referred to as the Fiedler-vector. We are interested in the heat equation

associated with the Laplacian, i.e. $\frac{\partial h_t}{\partial t} = -\hat{L}h_t$ where h_t is the heat kernel and t is time. The solution is found by exponentiating the normalized Laplacian eigenspectrum i.e. $h_t = \Phi \exp[-t\Lambda]\Phi^T$. The heat kernel is a $|V| \times |V|$ matrix, and for the nodes u and v of the graph G the resulting component is

$$h_t(u,v) = \sum_{i=1}^{|V|} \exp[-\lambda_i t]\phi_i(u)\phi_i(v) \tag{3}$$

When t tends to zero, then $h_t \simeq I - \hat{L}t$, i.e. the kernel depends on the local connectivity structure or topology of the graph. If, on the other hand, t is large, then $h_t \simeq \exp[-t\lambda_m]\phi_m\phi_m^T$, where λ_m is the smallest non-zero eigenvalue and ϕ_m is the associated eigenvector, i.e. the Fiedler vector. Hence, the large time behavior is governed by the global structure of the graph.

It is interesting to note that the heat kernel is also related to the path length distribution on the graph. If $P_k(u,v)$ is the number of paths of length k between nodes u and v then Chung [4] has concluded that

$$h_t(u,v) = \exp[-t] \sum_{k=1}^{|V|^2} P_k(u,v)\frac{t^k}{k!} \tag{4}$$

The path-length distribution is itself related to the eigenspectrum of the Laplacian. By equating the derivatives of the spectral and path-length forms of the heat kernel it is straightforward to show that

$$P_k(u,v) = \sum_{i=1}^{|V|} (1-\lambda_i)^k \phi_i(u)\phi_i(v) \tag{5}$$

The geodesic distance between nodes can be found by searching for the smallest value of k for which $P_k(u,v)$ is non zero, i.e. $d_G(u,v) = floor_k P_k(u,v)$.

Asymptotic Heat Kernel: On a Riemannian manifold, the heat kernel can be approximated by the so-called parametric [11]

$$h_t(u,v) = [4\pi t]^{-\frac{n}{2}} \exp[-\frac{1}{4t}d_G(u,v)^2] \sum_{m=0}^{\infty} q_m(u,v)t^m \tag{6}$$

where $d_G(u,v)$ is the geodesic distance between the nodes u and v on the Euclidean manifold, n is the dimensionality of the space and $q_m(u,v)$ real-valued co-efficients. When the manifold is locally Euclidean, then only the first term in the polynomial series is significant and the heat kernel is approximated by a Gaussian. Hence, to approximate the Euclidean distance between nodes in the embedding we can equate the spectral and Gaussian forms for the kernel. The result is

$$d_E^2(u,v) = -4t \ln\left\{ (4\pi t)^{\frac{n}{2}} \sum_{i=1}^{|V|} \exp[-\lambda_i t]\phi_i(u)\phi_i(v) \right\} \tag{7}$$

3 Geometric Properties of the Heat Kernel

In this section we review some of the known geometric and topological properties of the heat-kernel. Gilkey [5] has used spectral geometry to analyse the properties of manifolds using the Laplacian. To commence, we note that the trace of the heat kernel is determined by the Laplacian eigenvalues and asymptotically, for a Riemannian manifold M with metric-tensor g we can write

$$Z(t) = \sum_{i=1}^{N} \exp[-\lambda_i t] = \frac{1}{(4\pi t)^{\frac{n}{2}}} \sum_{m=0}^{\infty} a_m t^m \tag{8}$$

The co-efficients of the polynomial appearing in the above expression is determined by the Riemannian curvature tensor for the manifold. In particular, the co-efficient of the first term in the expansion is just the volume of the manifold, i.e. $a_o = Vol[M] = \frac{1}{\sqrt{\det g}}$, and the second co-efficient is related to the integral of the scalar curvature κ_s over the manifold, i.e. $a_1 = \frac{1}{6}\int_M \kappa_s dV$. The first result can be linked to the eigenvalues of the Laplacian. Suppose the eigenvalues are sorted into increasing magnitude order, i.e. $\lambda_1 \leq \lambda_2 \leq ...$, then as $k \rightarrow \infty$

$$a_o = Vol[M] = \frac{(2\pi)^n k}{\lambda_k^{\frac{n}{2}} \omega_n} \tag{9}$$

where ω_n is the area of the unit disk in $\mathcal{R}^n$. The second result can be linked to the topology of the manifold via the Gauss-Bonnet theorem. First recall that the scalar curvature κ_s is related to the Gaussian curvature K via the simple formula $\kappa_s = -\frac{1}{2}K$. The Euler index of the manifold is related to the area integral of the Gauss curvature by to the Gauss-Bonnet theorem

$$\chi(M) = -\frac{1}{2\pi}\int_M K dA \tag{10}$$

It is interesting to note that the Euler characteristic can be computed using the sectional curvatures for geodesic triangles on the manifold. If k_s is the sectional curvature of the geodesics and α_i the jump angles of the geodesic triangles, then

$$\int_M K dA = 2\pi\chi(M) - \sum_i \alpha_i - \int_{\partial M} k_s ds \tag{11}$$

Hence, the value of a_1 may be used to estimate the Euler index, or mean Gaussian curvature, of the manifold.

The sectional curvature of a geodesic on the manifold can be estimated easily if the Euclidean and geodesic distances are known. Suppose that the geodesic can be locally approximated by a circle. Let the geodesic distance between the pair of points u and v be $d_G(u, v)$ and the corresponding Euclidean distance

be $d_E(u,v)$. Further let the maximum deviation between the geodesic and the Euclidean chord by ξ. If the radius of curvature of the approximating circle be $r_s(u,v)$, then we have that

$$\xi(r_s(u,v)-\xi)=\frac{d_E(u,v)^2}{4} \tag{12}$$

If ξ is small then the sectional curvature is given by

$$k_s(u,v)=\frac{1}{r_s(u,v)}=\frac{4\xi}{d_E(u,v)^2} \tag{13}$$

If the geodesic path on the surface can be approximated by back-to-back right-angled triangles of height ξ, base $\frac{1}{2}d_E(u,v)$ and hypotenuse $\frac{1}{2}d_G(u,v)$, then

$$\xi=\frac{1}{2}\sqrt{d_G(u,v)^2-d_E(u,v)^2} \tag{14}$$

Hence, the sectional curvature is given by

$$k_s(u,v)=\frac{2\sqrt{d_G(u,v)^2-d_E(u,v)^2}}{d_E(u,v)^2} \tag{15}$$

Hence, with the expressions presented in the previous section of this paper, we could in principal chart the manifold by estimating the sectional curvatures of geodesics between nodes. The resulting representation could then be embedded by using a procedure similar to that suggested by Lindman and Caelli [7] using the Kruskal co-ordinates

$$Y(s)=\begin{cases}\frac{\sin(\sqrt{K}s)}{\sqrt{K}}\eta & \text{if } K>0\\ s\eta & \text{if } K=0\\ -\frac{\sinh(\sqrt{-K}s)}{\sqrt{-K}}\eta & \text{if } K<0\end{cases} \tag{16}$$

where the vector η is in the tangent space of M.

4 Multidimensional Scaling

The programme suggested in the previous section is ambitious and is the topic on ongoing research. However, to deliver proof of concept, here we adopt a simpler approach where we embed the pattern of Euclidean distances in a low dimensional in a manner which minimises the distortion using multidimensional scaling(MDS). In this way we use the distribution of embedded nodes rather than the curvature of the normalised Laplacian to charaterise the graphs understudy. The pairwise Euclidean distances between nodes $d_E(u,v)$ are used as the elements of an $|V|\times|V|$ dissimilarity matrix S, whose elements are defined as follows

$$S(u,v)=\begin{cases}d_E(u,v) & \text{if } u\neq v\\ 0 & \text{if } u=v\end{cases} \tag{17}$$

The first step of MDS is to calculate a matrix T whose element with row r and column c is given by $T(r,c) = -\frac{1}{2}[d_E(r,c)^2 - \hat{d}_E(r,.)^2 - \hat{d}_E(.,c)^2 + \hat{d}_E(.,.)^2]$, where $\hat{d}_E(r,.) = \frac{1}{|V|}\sum_{c=1}^{|V|} d_E(r,c)$ is the average dissimilarity value over the rth row, $\hat{d}_E.c$ is the dissimilarly defined average value over the cth column and $\hat{d}_E(.,.) = \frac{1}{|V|^2}\sum_{r=1}^{|V|}\sum_{c=1}^{|V|} d_E(r,c)$ is the average dissimilarity value over all rows and columns of the dissimilarity matrix T.

We subject the matrix T to an eigenvector analysis to obtain a matrix of embedding co-ordinates X. If the rank of T is $k, k \leq |V|$, then we will have k non-zero eigenvalues. We arrange these k non-zero eigenvalues in descending order, i.e. $l_1 \geq l_2 \geq \ldots \geq l_k > 0$. The corresponding ordered eigenvectors are denoted by $\boldsymbol{u}_i$ where l_i is the ith eigenvalue. The embedding co-ordinate system for the graphs obtained from different views is $X = [\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_s]$, where $\boldsymbol{f}_i = \sqrt{l_i}\boldsymbol{u}_i$ are the scaled eigenvectors. For the graph node indexed i, the embedded vector of co-ordinates is $\boldsymbol{x}_i = (X_{i,1}, X_{i,2}, ..., X_{i,s})^T$.

We use spatial moments to characterise the embedded point sets. The general moment is defined to be

$$\mu_{pq} = \sum_{i=1}^{|V|}\sum_{j=1}^{|V|}(X_{i,1} - \hat{X}_1)^p (X_{j,2} - \hat{X}_2)^q \tag{18}$$

where $\hat{X}_k = \frac{1}{|V|}\sum_{i=1}^{|V|} X_{i,k}$. From the raw moment data, we compute the four affine invariants suggested by Flusser and Suk [8]:

$$I_1 = \frac{\mu_{20}\mu_{02} - \mu_{11}^2}{\mu_{00}^4} \tag{19}$$

$$I_2 = \frac{\mu_{30}^2\mu_{03}^2 - 6\mu_{30}\mu_{21}\mu_{12}\mu_{03} + 4\mu_{30}\mu_{12}^3 + 4\mu_{21}^3\mu_{03} - 3\mu_{21}^2\mu_{12}^2}{\mu_{00}^{10}} \tag{20}$$

$$I_3 = \frac{\mu_{20(\mu_{21}\mu_{03} - \mu^2 12)} - \mu 11(\mu_{30}\mu_{03} - \mu_{21}\mu_{12}) + \mu_{02}(\mu_{30}\mu_{12} - \mu_{21}^2)}{\mu_{00}^7} \tag{21}$$

$$\begin{aligned} I_4 = (&\mu_{20}^3\mu_{03}^2 - 6\mu_{20}^2\mu_{11}\mu_{12}\mu_{03} - 6\mu_{20}^2\mu_{02}\mu_{21}\mu_{03} + 9\mu_{20}^2\mu_{02}\mu_{12}^2 \\ &+12\mu_{20}\mu_{11}^2\mu_{21}\mu_{03} + 6\mu_{20}\mu_{11}\mu_{02}\mu_{30}\mu_{03} - 18\mu_{20}\mu_{11}\mu_{02}\mu_{21}\mu_{12} \\ &-8\mu_{11}^3\mu_{30}\mu_{03} - 6\mu_{20}\mu_{02}^2\mu_{30}\mu_{12} + 9\mu_{20}\mu_{02}^2\mu_{21}^2 \\ &+12\mu_{11}^2\mu_{02}\mu_{30}\mu_{12} - 6\mu_{11}\mu_{02}^2\mu_{30}\mu_{21} + \mu_{02}^3\mu_{30}^2)/\mu_{00}^{11} \end{aligned} \tag{22}$$

The four moment invariants are used to compute the graph feature vector $\boldsymbol{F} = (I_1, I_2, I_3, I_4)^T$. We apply PCA to these feature vectors to project the pattern-spaces for sets of graphs.

5 Experiments

We have applied our embedding method to images from the COIL data-base. The data-base contains views of 3D objects under controlled viewer and lighting

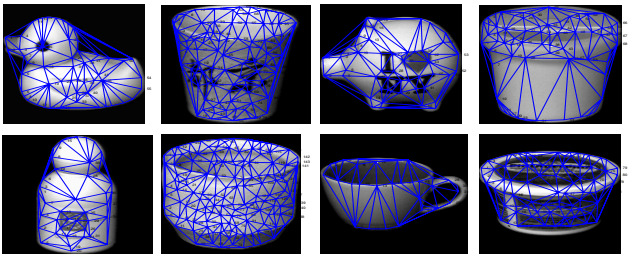


Fig. 1. Eight objects with their Delaunay graphs overlayed

conditions. For each object in the data-base there are 72 equally spaced views, which are obtained as the camera circumscribes the object. We study the images from eight example objects. A sample view of each object is shown in Figure 1. For each image image of each object we extract feature points using the method of [6]. We have extracted graphs from the images by computing the Voronoi tessellations of the feature-points, and constructing the region adjacency graph, i.e. the Delaunay triangulation, of the Voronoi regions. Our embedding procedure has been applied to the resulting graph structures.

We commence by investigating the behavior of the moments extracted from the embedded points. In Figure 2 we plot the four moments as a function of the time parameter t, we choose the duck graph on the top of the left in Figure 1 for

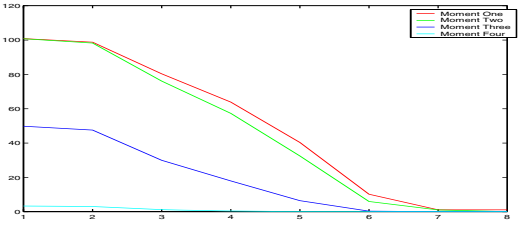


Fig. 2. Moments as a function of t for a graph from the Coil database

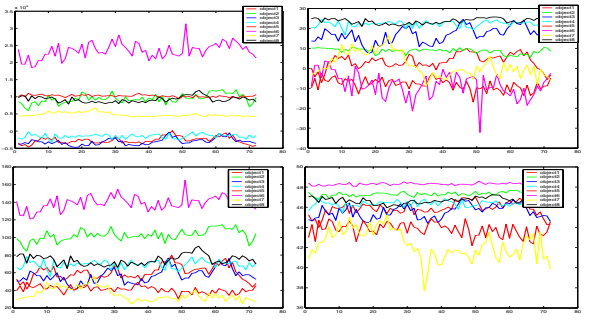


Fig. 3. Individual moment from the COIL database as a function of view number

our experiment graph. The main effect to note here is that as the time parameter increases then so the four moments become small and indistinguishable. In Figures 3 we plot the four moments separately as a function of the view number for the images of the eight objects studied in the COIL data-base. From the top-left and clockwise the sequence shows the first, second, third and fourth moments respectively. The individual moments appear relatively stable with view number. It is clear that although the individual moments could not be used to distinguish

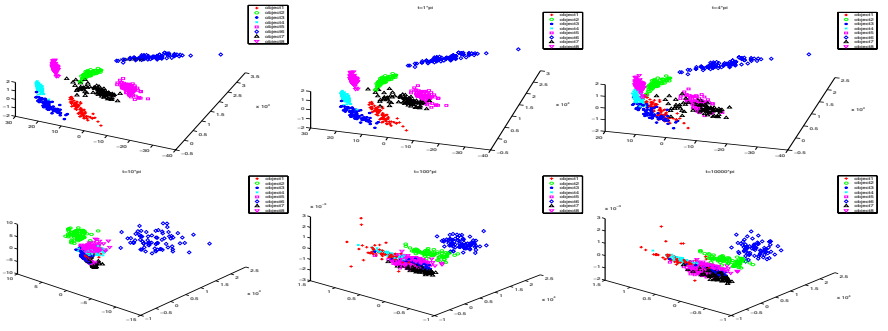


Fig. 4. Embedding with varying t

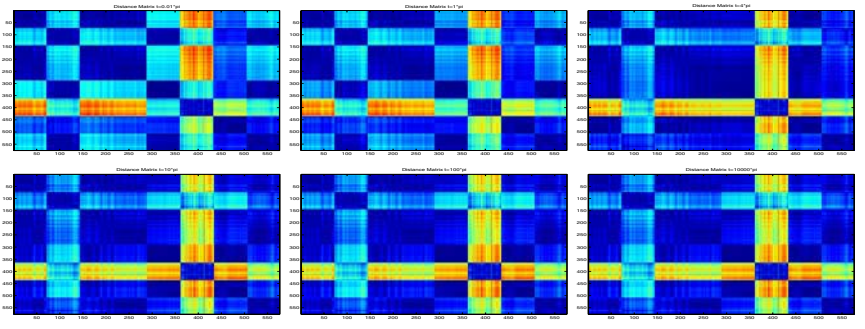


Fig. 5. Distance matrices with varying t

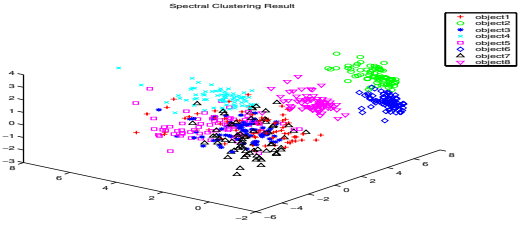


Fig. 6. Spectral Clustering Result

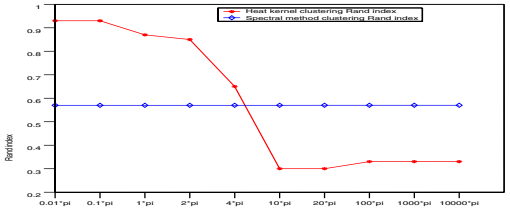


Fig. 7. Rand index for the heat-kernel and spectral clustering methods

the objects since their traces with view number overlap, when combined they are sufficient to distinguish the objects.

Based on this study of the moments, it appears that they may provide the basis for a stable clustering of the graphs. We have therefore performed PCA on vectors whose components are the four moments. The data has been projected onto the space spanned by the leading three eigenvectors for the moment-vectors. We then investigate the effect of varying the time parameter t. In Figure 4 we show the effect on the graph embeddings when we vary t from 0.01π to 10000π. In the top row of the figure from left to right, we show the clustering results obtained when t equals 0.01π,1π and 4π. In the bottom row, we show the results when t equals 10π, 100π and 10000π from left to right. In the figures the different views of the same object are displayed as differently colored symbols. In Figures 5 we show corresponding plots for the pairwise distances for the embedded graph nodes. The main feature to note from these plots is that as the value of t increases, then so the clusters corresponding to the different objects become overlapped.

For comparison, Figure 6 shows the corresponding result when spectral clustering is used. Here we use the leading eigenvalues of the adjacency matrix as the components of a feature vector. The main qualitative feature is that the different views of the ten objects are more overlapped than when the heat-kernel method is used with a low value of t.

To investigate the behavior of the two methods in a more quantitative way, we have plotted the Rand index [14] for the different objects. The Rand index is defined as: $R_I = \frac{A}{A+D}$ where A is the number of "agreements" and D is the number of "disagreements" in cluster assignment. The index is hence the fraction of views of a particular class that are closer to an object of the same class than to one of another class. The solid curve in the plot shows the Rand index as a function of t. The performance of the method drops off rapidly once t exceeds 2π. The Rand index for the spectral clustering method is shown as a dotted line.

6 Conclusions

In this paper we have shown how the Laplacian spectrum can be used to compute both Euclidean and geodesic distances between nodes in a graph. We provide a discussion which shows how the geometry of the manifold on which the nodes

reside can be linked to the properties of the heat kernel of the graph. This suggests that the manifold could be charted using information provided by the two distances. The analysis leads us to a simpler pragmatic approach to the problem where we perform low distortion embedding the Euclidean distances using MDS. The geometry of the embedded points is captured using affine moment invariants, and these are demonstrated to deliver good graph-clusters.

References

1. J. E. Atkins, E. G. Bowman, and B. Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM J. Comput.*, 28:297310, 1998.
2. Colin de Verdiere. Spectra of graphs. *Math of France*, 4, 1998.
3. E.Witten, M.B.Green, and J.H.Schwarz. Superstring theory. *Cambridge University Press*, 1988.
4. F.R.K.Chung. Spectral graph theory. *American Mathematical Society*, 1997.
5. P. B. Gilkey. Invariance theory, the heat equation, and the atiyah-singer index theorem. *Perish Inc.*, 1984.
6. C.G. Harris and M.J. Stephens. A combined corner and edge detector. *Fourth Alvey Vision Conference*, pages 147–151, 1994.
7. H.Lindman and T.Caelli. Constant curvature Riemannian scaling. *Journal of Mathematical Psychology*, 17:89–109, 1978.
8. J.Flusser and T.Suk. Pattern recognition by affine moment invariants. *Pattern Recognition*, 26:167–174, 1993.
9. B. Luo and E.R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *IEEE PAMI*, 23:1120–1136, 2001.
10. Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE PAMI*, 22:888–905, 2000.
11. S.Rosenberg. The laplacian on a Riemannian manifold. *Cambridge University Press*, 2002.
12. S.T.Yau and R.M.Schoen. Differential geometry. *Science Publication*, 1988.
13. S.Umeyama. An eigen decomposition approach to weighted graph matching problems. *IEEE PAMI*, 10:695–703, 1988.
14. W.M.Rand. Objective criteria for the evaluation of clustering. *Journal of the American Statistical Association*, 66:846–850, 1971.
15. X.Bai and E.R.Hancock. Heat kernels, manifolds and graph embedding. *Structural,Syntactic, and Statistical Pattern Recognition*, pages 198–206, 2004.
16. X.Bai, H.Yu, and E.R.Hancock. Graph matching using manifold embedding. *International Conference on Image Analysis and Recognition, LNCS, 2004*, pages 198–206, 2004.
17. X.Bai, H.Yu, and E.R.Hancock. Graph matching using spectral embedding and semidefinite programming. *British Machine Vision Conference*, pages 297–307, 2004.

Author Index