

Paolo Atzeni
Albertas Caplinskas
Hannu Jaakkola (Eds.)

LNCS 5207

Advances in Databases and Information Systems

12th East European Conference, ADBIS 2008
Pori, Finland, September 2008
Proceedings

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Paolo Atzeni
Albertas Caplinskas
Hannu Jaakkola (Eds.)

Advances in Databases and Information Systems

12th East European Conference, ADBIS 2008
Pori, Finland, September 5-9, 2008
Proceedings

Volume Editors

Paolo Atzeni
Università Roma Tre, Informatica e Automazione
Via Vasca Navale 79, 00146 Roma, Italy
E-mail: atzeni@dia.uniroma3.it

Albertas Caplinskas
Institute of Mathematics and Informatics
Akademijos str. 4, 08663, Vilnius, Lithuania
E-mail: alcapl@ktl.mii.lt

Hannu Jaakkola
Tampere University of Technology, Pori
P.O.Box 300, 28101 Pori, Finland
E-mail: hannu.jaakkola@tut.fi

Library of Congress Control Number: 2008933208

CR Subject Classification (1998): H.2, H.3, H.5.3, J.1

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-85712-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-85712-9 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2008
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12456345 06/3180 5 4 3 2 1 0

Preface

This volume contains the best papers presented at the 12th East-European Conference on Advances in Databases and Information Systems (ADBIS 2008) held during September 5–9, 2008, in Pori, Finland. The series of ADBIS conferences is the successor of the annual international workshops with the same title that during 1993-1996 were organized in Russia by the Moscow ACM SIGMOD Chapter. ADBIS 2008 continues the series of ADBIS conferences held in St. Petersburg, Russia (1997), Poznan, Poland (1998), Maribor, Slovenia (1999), Prague, Czech Republic (2000), Vilnius, Lithuania (2001), Bratislava, Slovakia (2002), Dresden, Germany (2003), Budapest, Hungary (2004), Tallinn, Estonia (2005), Thessaloniki, Greece (2006), and Varna, Bulgaria (2007). The conferences are initiated and supervised by an international Steering Committee chaired by professor Leonid Kalinichenko.

The ADBIS conferences established an outstanding reputation as a scientific event of high quality serving as an internationally highly visible showcase for research achievements in the field of databases and information systems. ADBIS 2008 aimed to create conditions for experienced researchers to impart their knowledge and experience to the young researchers at pre- or post-doctoral level, and to promote interaction and collaboration between European research communities (especially from Central and East Europe) and the rest of the world. The conference encourages contacts between the participants who are nationals of, but active outside, the Member States and Associated States and their colleagues in Member States and Associated States. Special attention is paid to collaboration of researchers in Central and East Europe.

The call for papers attracted 66 submissions from 21 countries. In a rigorous reviewing process the international Program Committee selected 22 papers for publishing in this volume. These papers were reviewed by three reviewers who evaluated their originality, significance, relevance as well as presentation and found their quality suitable for international publication. Topically, the accepted research papers span a wide spectrum of the database and information systems field: from query optimization and transaction processing via design methods to application oriented topics like XML and data on the Web.

We would like to express our thanks to all the people who contributed to making ADBIS 2008 a success. We thank the organizers of the previous ADBIS conferences, who made ADBIS a valuable trademark and we are proud to continue their work. We also thank the authors, who submitted papers to the conference, the international Program Committee and the external reviewers who made this conference possible by voluntarily spending their time to ensure the quality of submitted papers, the good spirit of the local organizing team in Pori (Finland) and the conference system administration team in Vilnius (Lithuania) for voluntarily giving their time and expertise to ensure the success of the conference. Special thanks to all sponsors whose significant

financial support made the conference possible. And last but not least we thank the Steering Committee and, in particular, its chairman, Leonid Kalinichenko, for their advice and guidance.

September 2008

Paolo Atzeni
Albertas Caplinskas
Hannu Jaakkola

Conference Organization

General Chair

Hannu Jaakkola Tampere University of Technology, Pori, Finland

Program Committee Co-chairs

Albertas Caplinskas Institute of Mathematics and Informatics, Lithuania
Paolo Atzeni Università Roma Tre, Italy

Program Committee

Marko Bajec University of Ljubljana, Slovenia
Guntis Barzdins University of Latvia, Latvia
Mária Bielíková Slovak University of Technology in Bratislava, Slovakia
Miklós Biró Corvinus University of Budapest, Hungary
Loreto Bravo University of Edinburgh, UK
Bostjan Brumen University of Maribor, Slovenia
Andrea Cali Free University of Bolzano, Italy
Alina Campan University of Cluj-Napoca, Romania
Daniela Durakova Technical University of Ostrava, Czech Republic
Marie Duži Technical University of Ostrava, Czech Republic
Johann Eder University of Vienna, Austria
Johann Gamper Free University of Bozen-Bolzano, Italy
Torsten Grust Technische Universität München, Germany
Hele-Mai Haav Tallinn University of Technology, Estonia
Piotr Habela Polish-Japanese Institute of Information Technology, Poland
Roland Hausser University of Erlangen-Nuremberg, Germany
Anneli Heimbürger University of Jyväskylä, Finland
Jaak Henno Tallinn Technical University, Estonia
Manfred A. Jeusfeld Tilburg University, Netherlands
Leonid Kalinichenko Institute for Problems of Informatics of the Russian Academy of Science, Russia
Ahto Kalja Tallinn University of Technology, Estonia
Kalinka Kaloyanova Sofia University, Bulgaria
Hannu Kangassalo University of Tampere, Finland
Natalya Keberle Zaporozhye National University, Ukraine
Maurice van Keulen University of Twente, The Netherlands

Margita Kon-Popovska	Cyril and Methodius University, FYRO Macedonia
Sergei Kuznetsov	Institute for System Programming of Russian Academy of Science, Russia
Mong Li Lee	National University of Singapore, Singapore
Mauri Leppänen	University of Jyväskylä, Finland
Bertram Ludaescher	University of California, USA
Esperanza Marcos	Rey Juan Carlos University, Spain
Tommi Mikkonen	Tampere University of Technology, Finland
Bernhard Mitschang	Stuttgart University, Germany
Jari Multisilta	Tampere University of Technology, Finland
Alexandros Nanopoulos	Aristotle University, Greece
Igor Nekrestyanov	St. Petersburg State University, Russia
Lina Nemuraite	Kaunas University of Technology, Lithuania
Mykola Nikitchenko	Kyiv National Taras Shevchenko University, Ukraine
Boris Novikov	University of St-Petersburg, Russia
Harri Oinas-Kukkonen	University of Oulu, Finland
Gultekin Ozsoyoglu	Case Western Reserve University, USA
Jari Palomäki	Tampere University of Technology, Finland
Apostolos Papadopoulos	Aristotle University, Greece
Oscar Pastor Lopez	Valencia University of Technology, Spain
Norman Paton	University of Manchester, United Kingdom
Evaggelia Pitoura	University of Ioannina, Greece
Jaroslav Pokorny	Charles University, Czech Republic
Boris Rachev	Technical University of Varna, Bulgaria
Peter Revesz	University of Nebraska-Lincoln, USA
Ita Richardsson	University of Limerick, Ireland
Karel Richta	Czech Technical University, Czech Republic
Tore Risch	Uppsala University, Sweden
Gunter Saake	Otto-von-Guericke-University Magdeburg, Germany
Peter Scheuermann	Northwestern University, USA
Vaclav Snasel	Technical University of Ostrava, Czech Republic
Nicolas Spyrtatos	University of Paris-South, France
Sergey Stupnikov	Institute for Problems of Informatics of the Russian Academy of Science, Russian
Kuldar Taveter	University of Melbourne, Australia
Ernest Teniente	Technical University of Catalonia, Spain
Bernhard Thalheim	Christian-Albrechts-Universität zu Kiel, Germany
Martin Theobald	Stanford University, USA
Pasi Tyrväinen	University of Jyväskylä, Finland
Ozgur Ulusoy	Bilkent University, Turkey
Yannis Velegrakis	University of Trento, Italy
Peter Vojtas	Charles University, Czech Republic
Tatjana Welzer	University of Maribor, Slovenia
Robert Wrembel	Poznań University of Technology, Poland
Shuigeng Zhou	Fudan University, China

ADBIS Steering Committee

Chairman: Leonid Kalinichenko (Russian Academy of Science, Russia)

Andras Benczur (Hungary)
Albertas Caplinskas (Lithuania)
Johann Eder (Austria)
Janis Eiduks (Latvia)
Hele-Mai Haav (Estonia)
Mirjana Ivanovic (Serbia)
Mikhail Kogalovsky (Russia)
Yannis Manolopoulos (Greece)
Rainer Manthey (Germany)
Manuk Manukyan (Armenia)
Joris Mihaeli (Israel)
Tadeusz Morzy (Poland)
Pavol Navrat (Slovakia)
Boris Novikov (Russia)
Mykola Nikitchenko (Ukraine)
Jaroslav Pokorný (Czech Republic)
Boris Rachev (Bulgaria)
Bernhard Thalheim (Germany)
Tatjana Welzer (Slovenia)
Viacheslav Wolfengagen (Russia)
Ester Zumpano (Italy)

Organizing Committee

Hannu Jaakkola, Ulla Nevanranta (Tampere University of Technology, Pori, Finland)

Additional Referees

L. Ablonskis
I.S. Altingovde
W.S. Cheah
H. Ding
J. Handl
T. Härder
M. Kratky
D. Lavbic
Y.X. Luo
P.C. Garçça de Marina
R. Ozcan
A. Sasa
N. Siegmund
G. Trajcevski

Table of Contents

Invited Paper

Randomization Techniques for Data Mining Methods	1
<i>Heikki Mannila</i>	

Submitted Papers

Dynamic Aggregation of Relational Attributes Based on Feature Construction	2
<i>Rayner Alfred</i>	
HypereiDoc – An XML Based Framework Supporting Cooperative Text Editions	14
<i>Péter Bauer, Zsolt Hernáth, Zoltán Horváth, Gyula Mayer, Zsolt Parragi, Zoltán Porkoláb, and Zsolt Sztupák</i>	
Reducing Temporary Trees in XQuery	30
<i>David Bednárek</i>	
Predictive Join Processing between Regions and Moving Objects	46
<i>Antonio Corral, Manuel Torres, Michael Vassilakopoulos, and Yannis Manolopoulos</i>	
Multiple-Objective Compression of Data Cubes in Cooperative OLAP Environments	62
<i>Alfredo Cuzzocrea</i>	
Reference Management in a Loosely Coupled, Distributed Information System	81
<i>Matthias Grossmann, Nicola Hönle, Daniela Nicklas, and Bernhard Mitschang</i>	
On Top- k Search with No Random Access Using Small Memory	97
<i>Peter Gurský and Peter Vojtáš</i>	
IDFQ: An Interface for Database Flexible Querying	112
<i>Mohamed Ali Ben Hassine and Habib Ounelli</i>	
Autonomic Databases: Detection of Workload Shifts with n -Gram-Models	127
<i>Marc Holze and Norbert Ritter</i>	
Large Datasets Visualization with Neural Network Using Clustered Training Data	143
<i>Sergėjus Ivanikovas, Gintautas Dzemyda, and Viktor Medvedev</i>	

Efficient Execution of Small (Single-Tuple) Transactions in Main-Memory Databases	153
<i>Heine Kolltveit and Svein-Olaf Hvasshovd</i>	
Analysis and Interpretation of Visual Hierarchical Heavy Hitters of Binary Relations	168
<i>Arturas Mazeika, Michael H. Böhlen, and Daniel Trivellato</i>	
QUESTO: A Query Language for Uncertain and Exact Spatio-temporal Objects	184
<i>Hoda Mokhtar and Jianwen Su</i>	
Extending Fagin’s Algorithm for More Users Based on Multidimensional B-Tree	199
<i>Matúš Ondreička and Jaroslav Pokorný</i>	
Increasing Expressiveness of Composite Events Using Parameter Contexts	215
<i>Indrakshi Ray and Wei Huang</i>	
Reclassification of Linearly Classified Data Using Constraint Databases	231
<i>Peter Revesz and Thomas Triplet</i>	
Evaluating Performance and Quality of XML-Based Similarity Joins	246
<i>Leonardo Ribeiro and Theo Härder</i>	
Preserving Functional Dependency in XML Data Transformation	262
<i>Md. Sumon Shahriar and Jixue Liu</i>	
Enabling XPath Optional Axes Cardinality Estimation Using Path Synopses	279
<i>Yury Soldak and Maxim Lukichev</i>	
Improvement of Data Warehouse Optimization Process by Workflow Gridification	295
<i>Goran Velinov, Boro Jakimovski, Darko Cerepnalkoski, and Margita Kon-Popovska</i>	
Towards the Design of a Scalable Email Archiving and Discovery Solution	305
<i>Frank Wagner, Kathleen Krebs, Cataldo Mega, Bernhard Mitschang, and Norbert Ritter</i>	
Author Index	321

Randomization Techniques for Data Mining Methods

Heikki Mannila

HIIT

Helsinki University of Technology and
University of Helsinki, Finland

`heikki.mannila@tkk.fi`

Abstract. Data mining research has concentrated on inventing novel methods for finding interesting information from large masses of data. This has indeed led to many new computational tasks and some interesting algorithmic developments. However, there has been less emphasis on issues of significance testing of the discovered patterns or models. We discuss the issues in testing the results of data mining methods, and review some of the recent work in the development of scalable algorithmic techniques for randomization tests for data mining methods. We consider suitable null models and generation algorithms for randomization of 0-1 -matrices, arbitrary real valued matrices, and segmentations. We also discuss randomization for database queries.

References

1. Gionis, A., Mannila, H., Mielikainen, T., Tsaparas, P.: Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1, 3, Article No. 14 (2007)
2. Ojala, M., Vuokko, N., Kallio, A., Haiminen, N., Mannila, H.: Randomization of real-valued matrices for assessing the significance of data mining results. In: *SIAM Data Mining Conference*, pp. 494–505 (2008)
3. Haiminen, N., Mannila, H., Terzi, E.: Comparing segmentations by applying randomization techniques. *BMC Bioinformatics* 8, 171 (2007)

Dynamic Aggregation of Relational Attributes Based on Feature Construction

Rayner Alfred

School of Engineering and Information Technology,
Universiti Malaysia Sabah,
Locked Bag 2073, 88999, Kota Kinabalu, Sabah, Malaysia
ralfred@ums.edu.my

Abstract. The importance of input representation has been recognised already in machine learning. This paper discusses the application of genetic-based feature construction methods to generate input data for the data summarisation method called Dynamic Aggregation of Relational Attributes (*DARA*). Here, feature construction methods are applied in order to improve the descriptive accuracy of the *DARA* algorithm. The *DARA* algorithm is designed to summarise data stored in the non-target tables by clustering them into groups, where multiple records stored in non-target tables correspond to a single record stored in a target table. This paper addresses the question whether or not the descriptive accuracy of the *DARA* algorithm benefits from the feature construction process. This involves solving the problem of constructing a relevant set of features for the *DARA* algorithm by using a genetic-based algorithm. This work also evaluates several scoring measures used as fitness functions to find the best set of constructed features.

Keywords: Feature Construction, Data Summarisation, Genetic Algorithm, Clustering.

1 Introduction

Learning is an important aspect of research in Artificial Intelligence (*AI*). Many of the existing learning approaches consider the learning algorithm as a passive process that makes use of the information presented to it. This paper studies the application of feature construction to improve the descriptive accuracy of a data summarisation algorithm, which is called Dynamic Aggregation of Relational Attributes (*DARA*) [1]. The *DARA* algorithm summarises data stored in non-target tables that have many-to-one relationships with data stored in the target table. Feature construction methods are mostly related to classification problems where the data are stored in target table. In this case, the predictive accuracy can often be significantly improved by constructing new features which are more relevant for predicting the class of an object. On the other hand, feature construction also has been used in descriptive induction algorithms, particularly those algorithms that are based on inductive logic programming (e.g., Warmr

[2](#) and *Relational Subgroup Discovery (RSD)* [3](#)), in order to discover patterns described in the form of individual rules.

The *DARA* algorithm is designed to summarise data stored in the non-target tables by clustering them into groups, where multiple records exist in non-target tables that correspond to a single record stored in the target table. In this case, the performance of the *DARA* algorithm is evaluated based on the descriptive accuracy of the algorithm. Here, feature construction can also be applied in order to improve the descriptive accuracy of the *DARA* algorithm. This paper addresses the question whether or not the descriptive accuracy of the *DARA* algorithm benefits from the feature construction process. This involves solving the problem of constructing a relevant set of features for the *DARA* algorithm. These features are then used to generate patterns that represent objects, stored in the non-target table, in the *TF-IDF* weighted frequency matrix in order to cluster these objects.

Section 2 will introduce the framework of our data summarisation approach, *DARA* [1](#). The data summarisation method employs the *TF-IDF* weighted frequency matrix (vector space model [4](#)) to represent the relational data model, where the representation of data stored in multiple tables will be analysed and it will be transformed into data representation in a vector space model. Then, section 3 describes the process of feature construction and introduces genetic-based (i.e., evolutionary) feature construction algorithm that uses a non-algebraic form to represent an individual solution to construct features. This genetic-based feature construction algorithm constructs features to produce patterns that characterise each unique object stored in the non-target table. Section 4 describes the experimental design and evaluates several scoring measures used as fitness functions to find the best set of constructed features. The performance accuracy of the *J48* classifier for the classification tasks using these summarised data will be presented and finally, this paper is concluded in section 5.

2 Dynamic Aggregation of Relational Attributes (*DARA*)

In order to classify records stored in the target table that have one-to-many relations with records stored in non-target tables, the *DARA* algorithm transforms the representation of data stored in the non-target tables into an $(n \times p)$ matrix in order to cluster these records (see Figure [1](#)), where n is the number of records to be clustered and p is the number of patterns considered for clustering. As a result, the records stored in the non-target tables are summarised by clustering them into groups that share similar characteristics. Clustering is considered as one of the descriptive tasks that seeks to identify natural groupings in the data based on the patterns given. Developing techniques to automatically discover such groupings is an important part of knowledge discovery and data mining research.

In Figure [1](#), the target relation has a one-to-many relationship with the non-target relation. The non-target table is then converted into bags of patterns

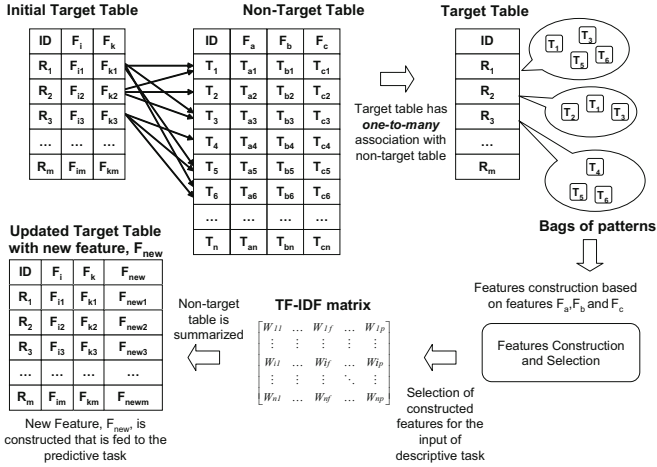


Fig. 1. Feature transformation process for data stored in multiple tables with one-to-many relations into a vector space data representation

associated with records stored in the target table. In order to generate these patterns to represent objects in the *TF-IDF* weighted frequency matrix, one can enrich the objects representation by constructing new features from the original features given in the non-target relation. The new features are constructed by combining attributes obtained from the given attributes in the non-target table randomly. For instance, given a non-target table with attributes (F_a, F_b, F_c) , all possible constructed features are $F_a, F_b, F_c, F_aF_b, F_bF_c, F_aF_c$ and $F_aF_bF_c$. These newly constructed features will be used to produce patterns or instances to represent records stored in the non-target table, in the $(n \times p)$ *TF-IDF* weighted frequency matrix. After the records stored in the non-target relation are clustered, a new column, F_{new} , is added to the set of original features in the target table. This new column contains the cluster identification number for all records stored in the non-target table. In this way, we aim to map data stored in the non-target table to the records stored in the target table.

3 Feature Construction in Machine Learning

3.1 Feature Construction

The problem of *feature construction* can be defined as the task of constructing new features, based on some functional expressions that use the values of original features, that describe the hypothesis at least as well as the original set. The application of feature construction for the purpose of summarising data stored in the non-target tables has several benefits. First, by generating relevant patterns describing each object stored in the non-target table, the descriptive accuracy of the data summarisation can be improved. Next, when the summarised data are

appended to the target table (e.g., the newly constructed feature, F_{new} , is added to the set of original features given in the target table as shown in Figure 1), it can facilitate the predictive modelling task for the data stored in the target table. And finally, feature construction can be used to optimise the feature space that describes objects stored in the non-target table.

With respect to the construction strategy, feature construction methods can be roughly divided into two groups: *Hypothesis-driven* methods and *data-driven* methods [5]. *Hypothesis-driven* methods construct new features based on the previously-generated hypothesis (discovered rules). They start by constructing a new hypothesis and this new hypothesis is examined to construct new features. These new features are then added to the set of original features to construct another new hypothesis again. This process is repeated until the stopping condition is satisfied. This type of feature construction is highly dependent on the quality of the previously generated hypotheses. On the other hand, *data-driven* methods, such as GALA [6] and GPCI [7], construct new features by directly detecting relationships in the data. GALA constructs new features based on the combination of booleanised original features using the two logical operators, *AND* and *OR*. GPCI is inspired by GALA, in which GPCI used an evolutionary algorithm to construct features. One of the disadvantages of GALA and GPCI is that the booleanisation of features can lead to a significant loss of relevant information [8].

There are essentially two approaches to constructing features in relation to data mining. The first method is as a separate, independent pre-processing stage, in which the new attributes are constructed before the classification algorithm is applied to build the model [9]. In other words, the quality of a candidate new feature is evaluated by directly accessing the data, without running any inductive learning algorithm. In this approach, the features constructed can be fed to different kinds of inductive learning methods. This method is also known as the Filter approach.

The second method is an integration of construction and induction, in which new features are constructed within the induction process. This method is also referred to as *interleaving* [10,11] or the wrapper approach. The quality of a candidate new feature is evaluated by executing the inductive learning algorithm used to extract knowledge from the data, so that in principle the constructed features' usefulness tends to be limited to that inductive learning algorithm. In this work, the filtering approach that uses the *data-driven* strategy is applied to construct features for the descriptive task, since the wrapper approaches are computationally more expensive than the filtering approaches.

3.2 Feature Scoring

The scoring of the newly constructed feature can be performed using some of the measures used in machine learning, such as information gain (Equation 1), to assign a score to the constructed feature. For instance, the *ID3* decision-tree [12] induction algorithm applies information gain to evaluate features. The information gain of a new feature F , denoted $InfoGain(F)$, represents the difference

of the class entropy in data set before the usage of feature F , denoted $Ent(C)$, and after the usage of feature F for splitting the data set into subsets, denoted $Ent(C|F)$, as shown in Equation [1](#)

$$InfoGain(F) = Ent(C) - Ent(C|F) \quad (1)$$

where

$$Ent(C) = - \sum_{j=1}^n Pr(C_j) \cdot \log_2 Pr(C_j) \quad (2)$$

$$Ent(C|F) = - \sum_{i=1}^m Pr(F_i) \cdot \left(- \sum_{j=1}^n Pr(C_j|F_i) \cdot \log_2 Pr(C_j|F_i) \right) \quad (3)$$

where $Pr(C_j)$ is the estimated probability of observing the j th class, n is the number of classes, $Pr(F_i)$ is the estimated probability of observing the i th value of feature F , m is the number of values of the feature F , and $Pr(C_j|F_i)$ is the probability of observing the j th class conditional on having observed the i th value of the feature F . Information Gain Ratio (IGR) is sometimes used when considering attributes with a large number of distinct values. The Information Gain Ratio of a feature, denoted by $IGR(F)$, is computed by dividing the Information Gain, $InfoGain(F)$ shown in Equation [1](#), by the amount of information of the feature F , denoted $Ent(F)$,

$$IGR(F) = \frac{InfoGain(F)}{Ent(F)} \quad (4)$$

$$Ent(F) = - \sum_{i=1}^m Pr(F_i) \cdot \log_2 Pr(F_i) \quad (5)$$

and $Pr(F_i)$ is the estimated probability of observing the i th value of the feature F and m is the number of values of the feature F .

3.3 Feature Construction for Data Summarisation

In the *DARA* algorithm, the patterns produced to represent objects in the *TF-IDF* weighted frequency matrix are based on simple algorithms. These patterns are produced based on the number of attributes combined that can be categorised into three categories. These categories include 1) a set of patterns produced from an individual attribute using an algorithm called P_{Single} 2) a set of patterns produced from the combination of all attributes by using an algorithm called P_{All} 3) a set of patterns produced from variable length attributes that are selected and combined randomly from the given attributes.

For example, given a set of attributes $\{F_1, F_2, F_3, F_4, F_5\}$, one could have $(F_1, F_2, F_3, F_4, F_5)$ as the constructed features by using the P_{Single} algorithm. In contrast, with the same example, one will only have a single feature $(F_1F_2F_3F_4F_5)$ produced by using the P_{All} algorithm. As a result, data stored

across multiple tables with high cardinality attributes can be represented as bags of patterns produced using these constructed features. An object can also be represented by patterns produced on the basis of randomly constructed features (e.g., (F_1F_5, F_2F_4, F_3)), where features are combined based on some pre-computed feature scoring measures.

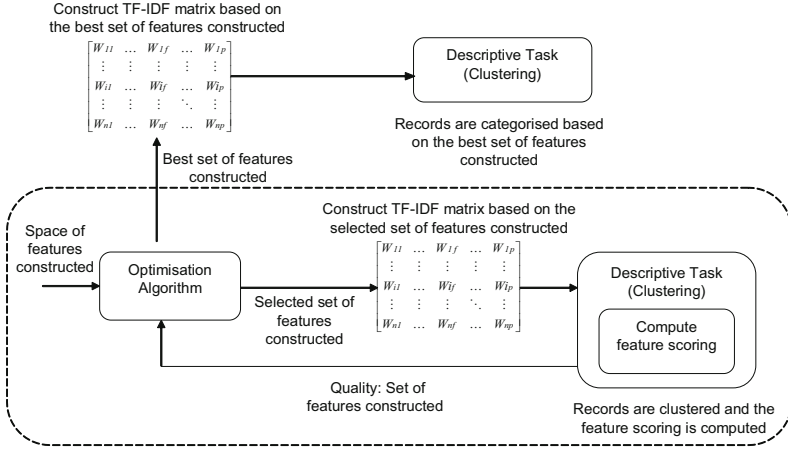


Fig. 2. Illustration of the Filtering approach to feature construction

This work studies a filtering approach to feature construction for the purpose of data summarisation using the *DARA* algorithm (see Figure 2). A set of constructed features is used to produce patterns for each unique record stored in the non-target table. As a result, these patterns can be used to represent objects stored in the non-target table in the form of a vector space. The vectors of patterns are then used to construct the *TF-IDF* weighted frequency matrix. Then, the clustering technique can be applied to categorise these objects. Next, the quality of each set of the constructed features is measured. This process is repeated for the other sets of constructed features. The set of constructed features that produces the highest measure of quality is maintained to produce the final clustering result.

3.4 Genetic-Based Approach to Feature Construction for Data Summarisation

Feature construction methods that are based on greedy search usually suffer from the local optima problem. When the constructed feature is complex due to the interaction among attributes, the search space for constructing new features has more variation. An exhaustive search may be feasible, if the number of attributes is not too large. In general, the problem is known to be NP-hard [14] and the search becomes quickly computationally intractable. As a result, the

feature construction method requires a heuristic search strategy such as Genetic Algorithms to be able to avoid the local optima and find the global optima solutions [15,16].

Genetic Algorithms (*GA*) are a kind of multidirectional parallel search, and viable alternative to the intractable exhaustive search and complicated search space [13]. For this reason, we also use a GA-based algorithm to construct features for the data summarisation task, here. This section describes a GA-based feature construction algorithm that generates patterns for the purpose of summarising data stored in the non-target tables. With the summarised data obtained from the related data stored in the the non-target tables, the *DARA* algorithm may facilitate the classification task performed on the data stored in the target table.

Individual Representation. There are two alternative representations of features: algebraic and non-algebraic [16]. In algebraic form, the features are shown by means of some algebraic operators such as arithmetic or Boolean operators. Most genetic-based feature construction methods like GCI [9], GPCI [7] and Gabret [17] apply the algebraic form of representation using a parse tree [18]. GPCI uses a fix set of operators, AND and NOT, applicable to all Boolean domains. The use of operators makes the method applicable to a wider range of problems. In contrast, GCI [9] and Gabret [17] apply domain-specific operators to reduce complexity. In addition to the issue of defining operators, an algebraic form of representation can produce an unlimited search space since any feature can appear in infinitely many forms [16]. Therefore, a feature construction method based on an algebraic form needs a restriction to limit the growth of constructed functions.

Features can also be represented in a non-algebraic form, in which the representation uses no operators. For example, in this work, given a set of attributes $\{X_1, X_2, X_3, X_4, X_5\}$, an algebraic feature like $((X_1 \wedge X_2) \vee (X_3 \wedge X_4 \wedge X_5))$ can be represented in a non-algebraic form as $\langle X_1X_2X_3X_4X_5, 2 \rangle$, where the digit, “2”, refers to the number of attributes combined to generate the first constructed feature.

The non-algebraic representation of features has several advantages over the algebraic representation [16]. These include the simplicity of the non-algebraic form to represent each individual in the process of constructing features, since there are no operator required. Next, when using a genetic-based algorithm to find the best set of features constructed, traversal of the search space of a non-algebraic is much easier.

A genetic-based feature construction method can be designed to construct a list of newly constructed features. For instance, during the population initialisation, each chromosome is initialised with the following format, $\langle X, A, B \rangle$, where X represents a list of the attribute’s indices, A represents the number of attributes combined, and B represents the point of crossover. Thus, given a chromosome $\langle 1234567, 3, 4 \rangle$, where the list 1234567 represents the sequence of seven attributes, the digit “3” represents the number of attributes combined and the digit “4” represents the point of crossover, the possible constructed features are $(F_1F_3F_4)$, $(F_6F_7F_5)$ and (F_2) , with the assumption that the attributes are

selected randomly from attribute F_1 through attribute F_7 to form the new features. The crossover process simply copies the sequence (string of attributes), (1234567), and rearranges it so that its tail, (567), is moved to the front to form the new sequence (5671234). The mutation process simply changes the number of attributes combined, A , and the point of crossover in the string, B . The rest of the feature representations can be obtained by mutating A , and B , and these values should be less than or equal to the number of attributes considered in the problem. As a result, this form of representation results in more variation after performing genetic operators and can provide more useful features.

Fitness Function. Information Gain (Equation 1) is often used as a fitness function to evaluate the quality of the constructed features in order to improve the predictive accuracy of a supervised learner [19,8]. In contrast, if the objective of the feature construction is to improve the descriptive accuracy of an unsupervised clustering technique, one may use the Davies-Bouldin Index (*DBI*) [20], as the fitness function. However, if the objective of the feature construction is to improve the descriptive accuracy of a semi-supervised clustering technique, the total cluster entropy (Equation 6) can be used as the fitness function to evaluate how well the newly constructed feature clusters the objects.

In our approach to summarising data in a multi-relational database, in order to improve the predictive accuracy of a classification task, the fitness function for the GA-based feature construction algorithm can be defined in several ways. In these experiments, we examine the case of semi-supervised learning to improve the predictive accuracy of a classification task. As a result, we will perform experiments that evaluate four types of feature-scoring measures (fitness functions) including the Information Gain (Equation 1), Total Cluster Entropy (Equation 6), Information Gain coupled with Cluster Entropy (Equation 8), Davies-Bouldin Index [20].

The information gain (Equation 1) of a feature F represents the difference of the class entropy in data set before the usage of feature F and after the usage of feature F for splitting the data set into subsets. This information gain measure is generally used for classification tasks. On the other hand, if the objective of the data modelling task is to separate objects from different classes (like different protein families, types of wood, or species of dogs), the cluster's diversity, for the k th cluster, refers to the number of classes within the k th cluster. If this value is large for any cluster, there are many classes within this cluster and there is a large diversity. In this genetic approach to feature construction for the proposed data summarisation technique, the fitness function can also be defined as the diversity of the clusters produced. In other words, the fitness of each individual non-algebraic form of constructed features depends on the diversity of each cluster produced.

In these experiments, in order to cluster a given set of categorised records into K clusters, the fitness function for a given set of constructed features is defined as the total clusters entropy, $H(K)$, of all clusters produced (Equation 6). This is also known as the Shannon-Weiner diversity [21,22],

$$H(K) = \frac{\sum_{k=1}^K n_k \cdot H_k}{N} \quad (6)$$

where n_k is the number of objects in k th cluster, N is the total number of objects, H_k is the entropy of the k th cluster, which is defined in Equation 7 where S is the number of classes, P_{sk} , is the probability that an object randomly chosen from the k th cluster belongs to the s th class. The smaller the value of the fitness function using the total cluster entropy (CE), the better is the quality of clusters produced.

$$H_k = - \sum_{s=1}^S P_{sk} \cdot \log_2(P_{sk}) \quad (7)$$

Next, we will also study the effect of combining the *Information Gain* (Equation 1) and *Total Cluster Entropy* (Equation 6) measures, denoted as $CE - IG(F, K)$, as the fitness function in our genetic algorithm, as shown in Equation 8, where K is the number of clusters and F is the constructed feature.

$$CE - IG(F, K) = InfoGain(F) + \frac{\sum_{k=1}^K n_k \cdot H_k}{N} \quad (8)$$

Finally, we are also interested in evaluating the effectiveness of feature construction based on the quality of the cluster's structure, which is measured using the Davies-Bouldin Index (DBI) [20], to improve the predictive accuracy of a classification task.

4 Experiments and Results

In these experiments we observe the influence of the constructed features for the *DARA* algorithm on the final result of the classification task. Referring to Figures 1, the constructed features are used to generate patterns representing the characteristics of records stored in the non-target tables. These characteristics are then summarised and the results appended as a new attribute into the target table. The classification task is then carried out as before. The Mutagenesis databases ($B1, B2, B3$) [23] and Hepatitis databases ($H1, H2, H3$) from PKDD 2005 are chosen for these experiments.

The genetic-based feature construction algorithm used in these experiments applies different types of fitness functions to construct the set of new features. These fitness functions include the *Information Gain* (IG) (Equation 1), *Total Cluster Entropy* (CE) (Equation 6), the combined measures of *Information Gain* and *Total Cluster Entropy* ($CE - IG$) (Equation 8) and, finally, the *Davies-Bouldin Index* (DBI) [20]. For each experiment, the evaluation is repeated ten times independently with ten different numbers of clusters, k , ranging from 3 to 21. The *J48* classifier (as implemented in *WEKA* [24]) is used to evaluate the quality of the constructed features based on the predictive accuracy of the

classification task. Hence, in these experiments we compare the predictive accuracy of the decision trees produced by the *J48* for the data when using P_{Single} and P_{All} methods. The performance accuracy is computed using the 10-fold cross-validation procedure.

Table 1. Predictive accuracy results based on leave-one-out cross validation using *J48* (C4.5) Classifier

<i>Datasets</i>	P_{Single}	P_{All}	CE	$CE-IG$	IG	DBI
B1	80.9 ± 1.4	80.0 ± 2.0	81.8 ± 1.3	81.3 ± 0.7	81.3 ± 0.7	78.6 ± 2.9
B2	81.1 ± 1.4	79.2 ± 3.0	82.4 ± 1.5	80.3 ± 2.1	80.2 ± 2.3	78.8 ± 1.3
B3	78.8 ± 3.3	79.2 ± 5.7	85.3 ± 3.9	84.4 ± 3.9	75.5 ± 4.7	78.9 ± 4.6
H1	70.3 ± 1.6	72.3 ± 1.7	75.1 ± 2.5	75.2 ± 2.4	74.9 ± 2.5	74.0 ± 2.0
H2	71.8 ± 2.9	74.7 ± 1.3	77.1 ± 3.3	76.9 ± 3.0	76.3 ± 3.8	76.1 ± 2.1
H3	72.3 ± 3.0	74.8 ± 1.3	77.1 ± 3.3	76.4 ± 3.8	76.5 ± 3.9	76.3 ± 2.6

The results for the mutagenesis (*B1*, *B2*, *B3*) and hepatitis (*H1*, *H2*, *H3*) datasets are reported in Table 1. Table 1 shows the average performance accuracy of the *J48* classifier (for all values of k), using a 10-fold cross-validation procedure. The predictive accuracy results of the *J48* classifier are higher when the genetic-based feature construction algorithms are used compared to the predictive accuracy results for the data with features constructed by using the P_{Single} and P_{All} methods.

Among the different types of genetic-based feature construction algorithms studied in this work, the CE genetic-based feature construction algorithm produces the highest average predictive accuracy. The improvement of using the CE genetic-based feature construction algorithm is due to the fact that the CE genetic-based feature construction algorithm constructs features that develop a better organisation of the objects in the clusters, which contributes to the improvement of the predictive accuracy of the classification tasks. That is, objects which are truly related remain closer in the same cluster.

In our results, it is shown that the final predictive accuracy for the data with constructed features using the IG genetic-based feature construction algorithm is not as good as the final predictive accuracy obtained for the data with constructed features using the CE genetic-based feature construction algorithm. The IG genetic-based feature construction algorithm constructs features based on the class information and this method assumes that each row in the non-target table represents a single instance. However, data stored in the non-target tables in relational databases have a set of rows representing a single instance. As a result, this has effects on the descriptive accuracy of the proposed data summarisation technique, *DARA*, when using the IG genetic-based feature construction algorithm to construct features. When we have unbalanced distribution of individual records stored in the non-target table, the IG measurement will be affected. In Figure 2, the data summarisation process is performed to summarise data stored in the non-target table before the actual classification task is

performed. As a result, the final predictive accuracy obtained is directly affected by the quality of the summarised data.

5 Conclusions

In this paper, we have proposed a genetic-based feature construction algorithm that constructs a set of features to generate patterns that can be used to represent records stored in the non-target tables. The genetic-based feature construction method makes use of four predefined fitness functions studied in these experiments. We evaluated the quality of the newly constructed features by comparing the predictive accuracy of the $J48$ classifier obtained from the data with patterns generated using these newly constructed features with the predictive accuracy of the $J48$ classifier obtained from the data with patterns generated using the original attributes. This paper has described how feature construction can be used in the data summarisation process to get better descriptive accuracy, and indirectly improve the predictive accuracy of a classification task. In particular, we have investigated the use of Information Gain (IG), Cluster Entropy (CE), Davies-Bouldin Index (DBI) and a combination of Information Gain and Cluster Entropy ($CE - IG$) as the fitness functions used in the genetic-based feature construction algorithm to construct new features.

It is shown in the experimental results that the quality of summarised data is directly influenced by the methods used to create patterns that represent records in the $(n \times p)$ $TF-IDF$ weighted frequency matrix. The results of the evaluation of the genetic-based feature construction algorithm show that the data summarisation results can be improved by constructing features by using the cluster entropy (CE) genetic-based feature construction algorithm. Finally, by improving the descriptive accuracy of the data summarisation approach, the predictive accuracy of a classification problem can also be improved, provided that the summarised data is fed to the classification task.

References

1. Alfred, R., Kazakov, D.: Data Summarisation Approach to Relational Domain Learning Based on Frequent Pattern to Support the Development of Decision Making. In: 2nd ADMA International Conference, pp. 889–898 (2006)
2. Blockeel, H., Dehaspe, L.: Tilde and Warmr User Manual (1999), <http://www.cs.kuleuven.ac.be/~ml/PS/TWuser.ps.gz>
3. Lavrač, N., Flach, P.A.: An extended transformation approach to Inductive Logic Programming. *ACM Trans. Comput. Log.* 2(4), 458–494 (2001)
4. Salton, G., Wong, A., Yang, C.S.: A Vector Space Model for Automatic Indexing. *Commun. ACM* 18(11), 613–620 (1975)
5. Pagallo, G., Haussler, D.: Boolean Feature Discovery in Empirical Learning. *Machine Learning* 5, 71–99 (1990)
6. Hu, Y.J., Kibler, D.F.: Generation of Attributes for Learning Algorithms. *AAAI/IAAI* 1, 806–811 (1996)

7. Hu, Y.J.: A genetic programming approach to constructive induction. In: Proc. of the Third Annual Genetic Programming Conference, pp. 146–157. Morgan Kaufman, Madison (1998)
8. Otero, F.E.B., Silva, M.S., Freitas, A.A., Nievola, J.C.: Genetic Programming for Attribute Construction in Data Mining. In: EuroGP, pp. 384–393 (2003)
9. Bensusan, H., Kuscü, I.: Constructive Induction using Genetic Programming. In: ICML 1996 Evolutionary computing and Machine Learning Workshop (1996)
10. Zheng, Z.: Constructing X-of-N Attributes for Decision Tree Learning. *Machine Learning* 40(1), 35–75 (2000)
11. Zheng, Z.: Effects of Different Types of New Attribute on Constructive Induction. In: ICTAI, pp. 254–257 (1996)
12. Quinlan, R.J.: Decision-Tree. In: C4.5: Programs for Machine Learning. Morgan Kaufmann Series in Machine Learning (1993)
13. Holland, J.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press (1975)
14. Amaldi, E., Kann, V.: On the Approximability of Minimising Nonzero Variables or Unsatisfied Relations in Linear Systems. *Theory Computer Science* 209(1-2), 237–260 (1998)
15. Freitas, A.A.: Understanding the Crucial Role of Attribute Interaction in Data Mining. *Artif. Intell. Rev.* 16(3), 177–199 (2001)
16. Shafti, L.S., Pérez, E.: Genetic Approach to Constructive Induction Based on Non-algebraic Feature Representation. In: R. Berthold, M., Lenz, H.-J., Bradley, E., Kruse, R., Borgelt, C. (eds.) IDA 2003. LNCS, vol. 2810, pp. 599–610. Springer, Heidelberg (2003)
17. Vafaie, H., DeJong, K.: Feature Space Transformation Using Genetic Algorithms. *IEEE Intelligent Systems* 13(2), 57–65 (1998)
18. Koza, J.R.: Genetic Programming: On the programming of computers by means of natural selection. *Statistics and Computing* 4(2) (1994)
19. Krawiec, K.: Genetic Programming-based Construction of Features for Machine Learning and Knowledge Discovery Tasks. *Genetic Programming and Evolvable Machines* 3, 329–343 (2002)
20. Davies, D.L., Bouldin, D.W.: A Cluster Separation Measure. *IEEE Trans. Pattern Analysis and Machine Intelligence.* 1, 224–227 (1979)
21. Shannon, C.E.: A mathematical theory of communication. *Bell system technical journal* 27 (1948)
22. Wiener, N.: *Cybernetics: Or Control and Communication in Animal and the Machine*. MIT Press, Cambridge (2000)
23. Srinivasan, A., Muggleton, S., Sternberg, M.J.E., King, R.D.: Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction. *Artif. Intell.* 85(1-2), 277–299 (1996)
24. Witten, I.H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco (1999)

HypereiDoc – An XML Based Framework Supporting Cooperative Text Editions^{*,**}

Péter Bauer¹, Zsolt Hernáth², Zoltán Horváth¹, Gyula Mayer³, Zsolt Parragi¹,
Zoltán Porkoláb¹, and Zsolt Sztupák¹

¹ Dept. of Programming Languages and Compilers

² Dept. of Information Systems

Faculty of Informatics, Eötvös Loránd University


Pázmány Péter sétány 1/C H-1117 Budapest, Hungary

bauer_p@inf.elte.hu, hernath@ullman.inf.elte.hu, hz@inf.elte.hu
gsd@inf.elte.hu, zsolt.parragi@eotvos.elte.hu, sztupy@eotvos.elte.hu

³ Hungarian Academy of Sciences, Research Center for Ancient Studies

Múzeum krt. 4/F. H-1088 Budapest, Hungary

gam@cs.elte.hu

Abstract. HypereiDoc is an XML based framework supporting distributed, multi-layered, version-controlled processing of epigraphical, papyrological or similar texts in a modern critical edition. Such studies are typically based on independent work of philologists using annotation systems like the Leiden Conventions. Current initiatives like TEI and Epidoc have definitive limitations both in expressional power and the way how individual results can form a cooperative product. The HypereiDoc framework provides XML schema definition for a set of annotation-based layers connected by an extensive reference system, validating and building tools, and an editor on-line visualizing the base text and the annotations. The framework makes scholars able to work on the same text in a cooperative and distributed way. Our framework has been successfully tested by philologists working on the Hypereides palimpsest.

1 Introduction

The XML document format is a well-respected solution for the document processing domain. A wide range of applications are based on XML from *DocBook*

* Supported by ELTE Informatikai Kooperációs Kutatási és Oktatási Központ.

** The initiative and the frames of the interdisciplinary co-operations have been established and are maintained by the Classical Philology Workshop – Eötvös József Collegium. (László Horváth; OTKA inv. no. T 47136 and IN 71311; horvathl@eotvos.elte.hu)

¹ The text edition of Hypereides' speech against Diondas is based on the above described editor. The publication is forthcoming in the Zeitschrift für Papyrologie und Epigraphik vol. 2008 (October). Similarly, this editor will be applied in the revised edition of Hypereides' Against Timandros (cf. [\[8,9\]](#)) forthcoming in AAHung vol. 2008. After the above mentioned publications the entire Greek texts together with the editor will be made accessible on the URL: <http://hypereidoc.elte.hu/>

[11] to *Office Open XML*, the new document format of Microsoft Word [22]. The flexibility with the ease of machine processing makes XML an ideal format for document handling. Very special but increasingly important areas of document handling are *epigraphy* and *papyrology*. Epigraphy is the study of inscriptions or epigraphs engraved into durable materials (e.g. stone). Papyrology focuses on the study of ancient literature, correspondence, legal archives, etc. as preserved in papyri. Epigraphy and papyrology include both the interpretation and translation of ancient documents. Such historical relicts are often damaged and their study produces controversial results by nature. Scholars have solutions of long standing for the situation: the system of critical annotations to the text.

Annotations may mark missing, unreadable, ambiguous, or superfluous parts of text. They should also quote information about the *reason* of the scholar's decision e.g. other document sources, well-accepted historical facts or advances in technology. Annotations also provide meta-information about the author of the individual critical notes and expose the supposed meaning according to the given scholar. It is of a primary importance that no information should be lost during the transcription process, even those remarks which will never appear in any critical edition should be kept either. The *Leiden Conventions* are the most accepted set of rules and symbols to indicate annotations in literary, epigraphical or papyrological texts [4].

The aim of the project is to equip the scholarly teams with general and flexible tools, which enable them to create both consistently tagged source files and pretty printed output.

For the realization of the project goals the XML document format has been chosen for storing the documents and the associated pieces of information. This format primarily supports the Unicode character encoding. The Text Encoding Initiative (TEI) Guidelines [25,27] provide detailed recommendations for storing documents in XML format. Its epigraphical customization is called EpiDoc [18]. During the course of the project we take these standards and recommendations as starting points. The TEI Guidelines' version at the time the project started was P4, now it is P5. The current EpiDoc (version 5) is based on TEI P4 and is not yet updated to TEI P5 but we follow the Guidelines of it. However, our goals exceed the possibilities of these recommendations, thus their customization and completion is required.

The rest of the paper is organized as follows: We formalize the problem in section 2. The design and implementation of the HypereiDoc framework are discussed in details in section 3. The successful application of the HypereiDoc framework in the Hypereides palimpsest project is described in section 4. In section 5 we give an overview of XML based projects from the epigraphical and papyrological domain. We summarize our results in section 6.

2 Formalizing the Problem

In order to formalize the problem and the needs of computer support discussed informally above, consider the following computer produced linguistic puzzle:

- get a human-readable text-file and split it into portions, called pages;
- damage strings by inserting, deleting, or replacing few characters;
- concatenate the pages in a random order.

To solve such text puzzles, i.e. to reconstruct the hypothetically original text

- the first task is to find the joints where the text can be splitted into pages
- the second is to restore damaged piece of text or characters,
- last concatenate pages in an adequate order found.

Following the above instructions the hypothetically original text can only be more or less exactly reconstructed after several tries. Each try may modify some words, may insert, delete, or replace characters, some of them use text versions produced by a sequence of earlier tries, etc. To make, catalogue, reference them, computer support is needed, which itself needs a problem-oriented data model that provides occasionally nested text operations cited informally above. To establish the adequate data model we need some base notions and terminology introduced next.

2.1 Basics

Definition 1 (Base text-document, Raw text-document, Raw text).

Given R – a text, i.e a particular sequence of UNICODEs (UTF8), let X_R denote the TEI P5 conformed XML document being valid against the document grammar [21]. Texts R , and X_R are referred to as *raw*, and (R -based) *base text-documents*, respectively. Any portion of text R and of #PCDATA typed element content in X_R is called *raw text*.

Remark 1. Notice that X_R defines a *frame of reference* to locate and reference any piece of raw text of R .

Going on with introducing our base terminology, we introduce three primitive binary operators used to locate, and issue semantics to, or modify raw texts inside base text-documents. Locating a raw text takes place by specifying the positions of its first and last character. In case of semantics issuer and corrective operations the first operand is a reference to a located raw text inside a base text-document, the second, in turn, always a raw text literal.

Definition 2 (Primitive Operations). Let O be the set of primitive operations $O = \{\mathbf{LO}, \mathbf{IN}, \mathbf{RE}\}$. They are semantic operations in the sense that each of them issues some semantics to raw texts inside a base text-document as seen below:

- **LO**cate: locate raw texts inside a base text-document.
Given X_R base text-document, and Xpointers x_s, x_e being valid according to the tagging of document X_R , $\mathbf{LO}(x_s, x_e)$ is the raw text between characters pointed to by x_s , and x_e , inclusive. An LO is called performable, if their operands are valid.

- **IN**terpret: interpret raw texts inside a base text-document.

Given raw text locator l_t for X_R base text-document as $\mathbf{LO}(x_{st}, x_{et})$, and t raw text literal, $\mathbf{IN}(l_t, t)$ issue semantics given by t to raw text located by l_t .

- **RE**vise: revise raw texts inside a base text-document.

Keeping notations l_t and t used above, $\mathbf{RE}(l_t, t)$ replaces raw text located by l_t with raw text t that may supply additional semantics as well.

2.2 Virtual Operation Performance

It is very important to see that base text-documents have to be kept untouched, even if semantics issuer or corrective operations are applied on it. One way to perform such operations would be to follow what database journal mechanisms do: executing operations results in new versioned complete (base) text-document, occasionally inheriting issued semantics, and corrections from other versions. This way would, however, lead to an unnecessary growth of base text-document versions, and instead, we develop a kind of *virtual* execution method whenever operations are to be performed. Informally, we say that applying a primitive operation o_1 to a base text-document X_R can be considered as an expression of form

$$(X_R, o_1),$$

and called a *virtual text-document*. If one wants to perform an operation o_2 on virtual text-document (X_R, o_1) , simple create an expression of form

$$((X_R, o_1), o_2),$$

and so on. Following this philosophy, and considering the expression

$$(((\dots (X_R, o_0) \dots), o_{r-1}), o_r),$$

$\forall 0 < i \leq r$, operation o_i refers to a raw text inside the virtual text-document

$$((\dots (X_R, o_0) \dots), o_{i-1}).$$

The latter, however, means that operation **LO** could be able to locate raw texts that are completely outside or partially inside X_R . That, in turn, in harmony with the definition of **LO** is possible, if operations' raw text literal can also be marked off by Xpointers, and all operations above implicitly refer to the same base text-document. We now formalize our conclusion as follows.

Definition 3 (Homogeneous Operation Sequence). A (possible empty) sequence $\{o_0, \dots, o_n\}$ of primitive operations is called homogeneous, if operands of each **LO** occurrence inside the sequence – being present either as the first operand of some **IN** or **RE**, or as an operation of its own – refers implicitly either to the same base text-document, or to a raw text literal operand of some preceding operation.

Definition 4 (Annotation). A possibly empty sequence of homogeneous operations that refers implicitly to a base text-document X_R , and established as a TEI P5 conformed XML document being valid against document grammar [21] is called an annotation. The annotation that implements an empty sequence is called the empty annotation, and denoted by A_\emptyset .

Definition 5 (Virtual text-document). Given R -based base text-document X_R , and a non-empty annotation sequence $\{A_{t_0}, \dots, A_{t_r}\}$, where indexes of annotations are some kind of time stamps indicating their creation time. (X_R, A_\emptyset) is an X_R -rooted virtual text-document, identical with X_R . Given X_R -rooted virtual text-document V_R , $(V_R, \{A_{t_0}, \dots, A_{t_r}\})$ is an X_R -rooted virtual text-document, defined by the expression $((\dots(V_R, A_{t_0})\dots), A_{t_r})$. The raw text content of virtual text-document $(V_R, \{A_{t_0}, \dots, A_{t_r}\})$ results in from V_R , by processing annotations A_{t_0}, \dots, A_{t_r} in the given order. Processing an annotation means performing its primitive operations in the order of the operation sequence that it implements.

Definition 6 (Merging Virtual text-documents having common roots). Given $V_R = (X_R, \{A_{t_{i_0}}, \dots, A_{t_{i_r}}\})$, $V'_R = (X_R, \{A_{t_{k_0}}, \dots, A_{t_{k_s}}\})$ X_R -rooted virtual text-documents, and $\{B_{t_0}, \dots, B_{t_n}\}$ annotation sequence. Suppose, for each natural number m , for that $0 \leq m \leq n$ holds, there exists $0 \leq j \leq r$, or $0 \leq l \leq s$ such that either $B_{t_m} = A_{t_{i_j}}$ or $B_{t_m} = A_{t_{k_l}}$ hold. A virtual text-document of form $(X_R, \{B_{t_0}, \dots, B_{t_n}\})$ is called a merge of V_R and V'_R .

2.3 A Data Model for Text Annotations

The data model developed here is referred to as VITAM². Informally, it contains *virtual text-documents* as data items and *annotation sequences* and virtual text-documents' *merging* as operations. Since data are virtual i.e. their raw text content can only be achieved by processing annotation sequences over virtual text-documents, an important issue is to define the *well-formedness*, and the *validity* of virtual text-documents.

Definition 7 (Well-formedness). Given X_R base text-documents, (X_R, A_\emptyset) is a well-formed virtual text-document. Given V_R well-formed virtual text-document, and $\{A_0, \dots, A_r\}$ annotation sequence, $(V_R, \{A_0, \dots, A_r\})$ is a well-formed virtual text-document.

Definition 8 (Annotation Validity). Annotation A_\emptyset is valid with respect to any virtual text-document. Given V_R virtual text-document, an annotation A is valid with respect to V_R , iff all occurrences of operations **LO** inside A is performable. An annotation sequence $\{A_0, \dots, A_s\}$ is valid w.r.t. V_R , iff A_0 is valid w.r.t. V_R , and $\forall 1 \leq i \leq s$, annotation A_i is valid w.r.t. $(V_R, \{A_0, \dots, A_{i-1}\})$.

Definition 9 (Virtual text-document Validity). Given, X_R base text - document, (X_R, A_\emptyset) is a valid virtual text-document. Given V_R valid virtual

² Virtual Text-document Annotation Model.

text-document, and $\{A_0, \dots, A_r\}$ annotation sequence being valid w.r.t. V_R , $(V_R, \{A_0, \dots, A_r\})$ is a valid virtual text-document.

Remark 2. Notice, well-formedness does only declare the validity of XML documents virtual text-documents consist of against the document grammar [21]. It is also important to see that while valid annotation sequences w.r.t. valid virtual text-documents produce valid virtual text-documents from those, merging commonly rooted virtual text-documents does not, however, warrant valid merge, unless the merging procedure involves forced validity check.

3 Implementation

The XML model is based upon the TEI Guidelines (version P4 and P5) and its epigraphical customization, Epidoc Guidelines (version 5). We extended these standards to meet the HypereiDoc project requirements, thus we can use embedded and overlapped annotations and we also support a more free way to use the Critical Apparatus.

3.1 Layered Structure

Our schema is based on a multi-layer approach. We defined a *Base Text Layer* (see definition [1]) where only the original text and its physical structure is stored and which may not be modified later, an *Ordering and Indexing Layer* defining the pages' order and place in the codices and one or more *Annotation Layers* (see definition [4]) with the attached philological metadata. This model provides the means for stepwise adding of basic semantic information, summarizing the scholar team's knowledge base, team work, cross-checking, and proof-reading. Later editions may be based upon one or more previously published layers, thus creating critical editions is also supported.

Philologists can define their own Annotation Layers which may refer to only the Base Text Layer or one or more Annotation Layers. They can add notes and annotations to the original text and to previous annotations, they can make reflections on earlier work or create a new interpretation. We have designed a schema to handle these references and to support the distributed and collaborative work with using more Annotation Layers in one edition.

To make exact references to any point of the text, we need to discuss the structure of the text. The primary structure of the text is its logical structure according to the TEI Guidelines. The TEI suggests the header part for storing the associated information, while the text is structured by `div` tags. Also, for the physical structuring of the text representation empty tags are suggested according to XML's milestone technique. We store the transcription text in the Base Text Layer this way.

The palimpsest provides an existing physical structure of the text. This presents a well-identifiable base for processing the document as it can be clearly sectioned into codices, quires, leaves, sides, columns, and lines. Therefore we intend to regard these as the primary structure for our Reference System, making it possible to

define exact references to the document's specific parts. The references are needed for philological processing, for annotating the text and for mapping between the images and the transcription.

For philologists processing the document the most important aspect is the annotation facilities, as they can use it with the Leiden Conventions, and the application of the critical apparatus. The TEI sets up the structured recording of this information in the text, while the EpiDoc describes guidelines for the application of these techniques. However, a weakness of TEI P4 and EpiDoc is that these pieces of information are stored in the form of XML tags inserted into the document [2,3]. Therefore due to the requirements for well formed XML documents, annotations defined by the philologists can be embedded only if the tags are balanced.³

Let us consider the following example. The string *omen* is readable, however, *aut* beside it is missing due to a flaw in the material of the codex. At the same time, the transcriber has succeeded at reconstructing the missing part. According to the Leiden conventions the respective annotation is *[aut]omen*. Nevertheless, the transcribing philologist observes that the *t* and *o* characters are superfluous, and probably got into the text as an error on the part of the original copyist of the document. This can be annotated as *[au{t}o]men*, but this annotation cannot be encoded with the XML tags suggested by TEI P4 and Epidoc. This is a quite possible situation in the palimpsest. Besides the philological annotations the text parts marked by the apparatus of similar passages may also overlap.

Consequently, we have developed a Reference System built on the physical structure of the document. This enables the handling of any overlapping annotation. With this reference system missing word and sentence boundaries can easily be described, even if interpreted differently by various philologists. Punctuations missing from the document can also easily be coded. As a result, the XML transcription may consist of several layers.

We face a special situation in the case of our sample project: the Archimedes Palimpsest is a secondary product, it has been created from reused sheets of former manuscript books. Before the secondary usage the leaves must have been cleaned as much as possible to make them fit for bearing the new texts. Scholars are interested in both the old texts (hardly visible remains of a lower layer on the surface of the pages, as called *undertext*) and the new texts (an upper layer, as called *overtext*).

Since the undertext has not yet been exactly identified on all leaves and it is also possible that by finding new leaves we need to reorder the whole codex, we intend to regard the page numbering of the overtext as the base for the Reference System. This can be extended or changed while it does not affect the interpretation of the undertext. Since the undertext can only be interpreted or even displayed in its original page order if the exact structure of the undertext

³ TEI P4 has draft recommendations on solving this problem [26], but these are not elaborated and less powerful than our Reference System. TEI P5 supports multiple ways to handle overlapping tags, and we use one of these techniques to implement the Reference System.

is known, we defined the Ordering and Indexing Layer independently from the Base Text Layer. We store this data in an external XML file because philologists may not agree on the page order and they may want to use their own Ordering and Indexing XML file. Ordering and Indexing mean that we assign the overtext leaves and sides to undertext leaves and sides and this assignment can be changed without modifying the Base Text Layer. Therefore we can change the page order in the restructured codices if needed without the need of changing the references of the annotations.

The Base Text Layer’s physical structure is based on the overtext, the pages are identified with the overtext leaf and side while columns, lines are marked regarding the undertext, thus the undertext lines are exactly identifiable. The Ordering and Indexing Layer assigns the overtext leaves and sides to undertext quires, leaves and sides.

3.2 Reference System

Due to the embedded and overlapped annotations and the multi-layer approach we define three types of references. The *Absolute References* point at a character position in the Base Text (cf. remark [1](#)). Their structure is overleaf, overside, column, line, (optional) character, and (optional) position. The *Internal Relative References* point at a character position in text inserted by a previous annotation in the same Annotation Layer. Their structure is annotation identifier, (optional) character, and (optional) position. The *External Relative References* point at a character position in text inserted by an annotation in a previous Annotation Layer. They identify the previous layer, the annotation, (optionally) the character, and (optionally) the position. Notice, the Relative References above are particular cases of virtual operation performance (cf. section [2.2](#)).

Only alphanumeric characters are numbered, whitespaces and the philologists’ various brackets are disregarded because, in harmony with definition [1](#), they are annotations in the text. Character means only alphanumeric characters in the paper. Please note that the “character” field does not refer to a character but the position between two characters. The zeroth referred position is before the first character in the line while the first referred position is after the first character of the line and before the second. In a line containing n characters the n^{th} position is after the last character of the line.

Most annotations may have two different meaning. It is possible that the character string we refer to is present in the base text or in a previous annotation. We call this type of annotation *Marking Annotation* (see operation **IN** in definition [2](#)). The marked text may be later referred absolutely or relatively to this annotation. It is also possible that the annotation inserts new characters in the text. This type of annotation is called *Inserting Annotation* (see operation **RE** in definition [2](#)). The inserted text may only be referred relatively to this annotation.

Relative References are used if we want to refer to characters inserted by a previous annotation. To make this possible all annotations regardless their type have an identifier which is unique in the given layer. Annotations must be

processed by the course of their identifier’s lexicographic order which is identical to the order of the tags in the XML document.

Please note that if we want to refer to a character position in a text portion which is inserted by multiple embedded annotations, the exact annotation which has actually inserted the character is known, therefore we do not have to deal in the references with the annotation hierarchy.

In cases of embedded and overlapping annotations it is possible that the reference is ambiguous. For instance the base text is *abc* and the annotation claims that *def* is missing after *a* which is marked as *ab[def]c* according to the Leiden Conventions. After that the absolute character position 1 is ambiguous: it can either point to *ab|[def]c* or *ab[def]|c*. (The point where the examples refer to is marked with a | character.) In this case we use the Position attribute which has four values: "l" for left side, "r" for right, "b" for before, and "a" for after.

The "l" value is applicable when the Character attribute is present and it points the position before the annotation that was inserted to the Character position given. The "r" value is applicable when the Character attribute is present and it points the position after the annotation that was inserted to the Character position given. The "b" value is applicable in Relative References when the Character attribute is not present and it points the position before the referred annotation, while the "a" is applicable in Relative References when the Character attribute is not present and it points the position after the referred annotation.

Please note that the Position attribute is omissible but Character attribute can’t be omitted if Position is not present or has the value of "l" or "r". If the Position attribute takes value "b" or "a" then Character attribute must be omitted.

We can also use Relative References to Marking Annotations. This makes unambiguous the character positions at the end of embedded and overlapped annotations. In the previous example the Relative Reference for character position 0 in the annotation refers to *ab|[def]c* and the relative reference for character position 3 refers to *ab[def]|c*. This is useful when marked text is inserted before or after an already marked text part.

3.3 XML Pointer-Based Implementation of the Reference System

The XML Pointer Language (XPointer) Framework is a W3C standard that allows one to point to an arbitrary position in an XML document. An extension of Xpointers introduced by TEI P5 Guidelines offers some supplements [28], such as the `left` and `right` pointer schemes, to the main standard.

To implement our reference system with XPointer we use the following mechanism: An *Absolute Reference* to leaf 27, side recto, column a, line 1 and character 3 looks like the following figure:

```
archimedes/P2/#xpointer(//pb[n='27:r']/following::cb[n='a']/
  following::lb[n='1']/following::text()[1]/point()[3])
```

In the example above `archimedes/P2` is the name of the base text file. In this encoding not only the file name but also the version number of the base text

is included, therefore possible later changes of the base text will not interfere with the reference system. After the file name the location part of the reference is converted into the exact position within the XML document. Because our XML structure has empty tags to mark the physical structure we had to use the following axis, which unfortunately makes the references more complex. After linking to the correct text node we use the point function of the XPointer scheme to point to the exact position.

In internal relative references we use

```
#xpointer(//[xml:id='b13']/text()/point()[2])
```

which means the second character position in the raw text in the thirteenth annotation. Our reference system allows us to include positional information like *left*, *right*, *before* or *after*. To use positions like *left*, or *right* we use TEI P5's supplement functions: **left** and **right**. We think of *before* and *after* as the position before or after an annotation, thus they are used like this:

```
#left(xpointer(//[xml:id='b13']))
```

This refers to the leftmost point of an annotation tag, which in turn is “before” the annotation. The after tag is similar but uses the **right** function instead of **left**.

External relative references are composed of the file information from *absolute references* and the relative positional information from *internal relative references*. We need to include a file name which includes the name of the annotator, a version number, and the location of the annotation like this:

```
annotations/mgy/a001#xpointer(//[xml:id='b2']/text()/point()[1])
```

Unfortunately, when the TEI Consortium designed the TEI P5 Guidelines they did not think about XPointer as a pointer to an arbitrary position, but as a pointer to an arbitrary tag. Because of this the guidelines lack support of the type of overlapping we need.

The guidelines allow us to implement new features by creating a new XML namespace but we wanted to stick with the P5 guidelines to maintain maximum compatibility. Therefore we had to use two of the tags that allow us to link to an interval in the text. These two tags are the **app** and the **note** tags. The **app** tags contains a critical notes to spans of texts. From a philological point of view this can also be used to describe the annotations we need. We use the **from** and **to** attributes to denote the start and end position of the annotation. The **note** tags have the **target** and **targetEnd** attributes to express the start and end position of annotations.

To ease the publication there is also a “flat file format” that is more close to the basic TEI P5 Guidelines. In this file format we do not use the XPointer scheme, because there are only a few tools that can handle it. Instead we use that feature of the **app** and the **note** tags, that allows them to be inlined into the text. In this mode at the beginning of the location the annotation refers to, we add

an **anchor** tag, and we add the **app** or the **note** tag at the end of the referenced interval with a link to the anchor. Of course this breaks the collaborative nature of our system, so this file format should only be used for a frozen digital dataset of a publication.

3.4 Version Control

Though this type of collaborative work is agile by nature, we do not establish a real version control system like Subversion or CVS does, where you can acquire the whole file, regardless whether it is the head revision or an earlier version. In our system what you have is a directory tree containing the base text-document and its annotation sequences. Since the base text-document is read-only, and therefore, not under version control, the annotation sequence may be, however, extended. This extension means that new annotations may be added to such a sequence (e.g. annotations that occasionally have impacts on some previous annotations, but can never change them). Practically this is the same as other version controlling systems store their files internally, because they usually does only store the differences between the different versions.

To accomplish this, we use a web-server called WEBRick [10], that handles the requests in a RESTful [7] way. In this system a virtual-text document can be considered as a resource on which following version control operations are defined. The **list** operation shows the annotations of this resource (the base virtual-text document), the **create** operation adds a new annotation to the sequence (automatically time-stamped), and the **show** operation gets the appropriate version of the file from the server. Because we deny the modification and deletion of resources, operations **modify** and **delete** are not supported by our system.

3.5 Tools

Our XML format contains a flexible XPointer scheme which is not easily editable by simple text editors. To support user friendly editing of the texts, we developed a What You See Is What You Get editor. It is not only an editor but also helps with the publishing of the finished document.

It supports working with layered and flat XML files: it has a Base Text mode which is used when one start working with a new codex. In this mode the philologist may edit the base text and its annotations at the same time. Its primary output is the flat XML format which is after the base text is finished can be converted to the layered structure.

For the layered structure the editor has an Annotation mode. In this mode editing the base text is disabled, but adding, modifying and deleting annotations are still possible. One can select already existing and published annotation XML files which will be the base of the work. The editor helps with the conflict resolution between the dependencies – when two or more XML documents conflicts each other – and it can also flatten the finished work to help the publication, because other TEI P5 compatible tools might use the flat file format more easily (see definition [6]).

When one loads a layered XML document, the editor first checks whether or not the XPointers point to existing locations. If the input is valid it also validates the dependencies and tries to resolve the conflicts and unifies them controlled by the philologist whenever necessary (cf. remark 2). If the previous process succeeds the user is greeted with a window like the screenshot in Figure 1. For

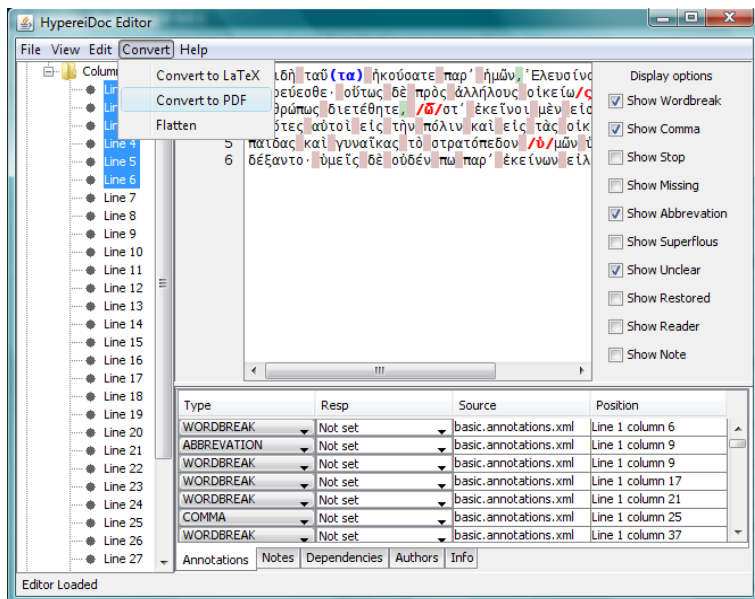


Fig. 1. Editor in Annotation mode

publications we provide some additional tools, such as the \LaTeX converter, that converts the base text, and the selected annotations into a \LaTeX file, that can be converted to PDF. Our integrated toolset consists of:

- Java based, platform-independent editor with graphical interface producing valid XML output.
- Java based tool for displaying the XML encoded transcription data in a form which is traditionally used by scholars, based on a compiler producing PDF.
- Java based validator tool which checks basic semantic relations.
- Java based converter tool exporting and importing flattened (one-layer) TEI P5 transcriptions.
- GTK based, platform-independent GIMP plug-in for linking image positions with lines of transcription producing XML output.

4 The Hypereides Palimpsest – A Sample Project

The HypereiDoc system has been created with the application to the decipherment of the now famous Archimedes Palimpsest [13] in view 4. It consists of

⁴ The story of the Palimpsest is described in detail in [6].

remains of at least five former codices. One of those discarded and reused books contained speeches of Hyperides [8], [9]. The transcription process involves many scholars working in different groups, making new suggestions and referring to each other's work.

It is the intention of the owner of the manuscript, that the Archimedes Palimpsest should be presented to the public in the most adequate way. Prime targets are the lower texts of the codex, together with the complex history of their decipherment. Presentation includes tagging and formatting. To format the sort of complex scholarly texts to be created, there exists an excellent platform, i.e. Edmac [5] resp. its variant for L^AT_EX Ledmac [30].

However, as excellent as is L^AT_EX at formatting, and as good as it is at tagging, the level of sophistication it provides at the latter is way behind what we need. Our goal is to document not only the final result, but also important steps in the scholarly process of creating the transcription. Of course, consistent tagging must not be so permissible as it is the case with a T_EX based system. Thus we could retain L^AT_EX as our frontend for paper publication and paper-like visualization, but had to find an adequate system for tagging. Therefore a system meeting the complex scholarly requirements has been devised – the HypereiDoc framework.

Figure 2 shows the first five lines of a page. In the formatted output margins are reserved to refer to the present physical structure of the codex: in the left margin leaf 138, side recto is noted, in the right margin the line numbers. In line 3 parentheses indicate complementation of an abbreviation, and in line 5 curly brackets enclose a letter visible and readable in the codex, but superfluous according to grammatical rules. A dot beneath a letter shows that the letter is incomplete, but the traces are compatible with the interpretation presented. The

138r	τοῦ μὲν εὐρίσκοντος ἐν τῷ δικαστηρίῳ μὴ ἔλαττον ἢ τοῖς παισίν· ἐὰν δὲ πλείω περιποιήσῃσι τοῖς παι- σίν, τούτων εἴη φιλοτιμί(α). αὐτοῖς δὲ τοὺς ἐπιτρό- πους ἀπαγορεύουσιν οἱ νόμοι μὴ ἐξεῖναι τὸν οἶκον μισθώσασθαι· ἔξεστι δ' {ἐ} ἐν τῷ δικαστηρίῳ ἀμφισ-	5
------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---

Fig. 2. Formatted main text

main text is accompanied by two series of annotations (a so called apparatus of similar passages and a critical apparatus). Figure 3 is a snippet of the second apparatus. Traditionally, the language of the apparatuses is latin with commonly used abbreviations. Since the existing pages do not contain the beginning and the title of the speech, the title is reconstructed from other sources, and does not have a corresponding line number. A significant difference between the XML source files and the output formatted for print is, that the latter contains only a selection of the information available. E.g. in line 5 the emendation of the text is so obvious, that no reason needs to be given. In line 1 reading has been substantially improved against the first publication in [8], and account should be given of it. In line 3 the cited scholar suggests to insert a two letter word, but this emendation is not regarded necessary by the editor of the apparatus. In line 10

Tit. vel Ὑπερ Ἀκαδήμους (or. [I] Jn.) coni. Todd || Fol. 138r
 1 μὴ ἔλαττον ἦ Wilson || 3 τούτων (ἀν) εἴη Handley || 10 scriba
 spatium quinque litt. in fin. v vacuum reliquit propter defectum membr.,
 ubi antea cum stylo materiam perforavit lacunamque creavit || 14

Fig. 3. Formatted annotations

we face a strange and rare situation, which can not be formalized and therefore is described in a ‘human readable’ sentence (“free text” within the annotations). The formatted version illustrated above is tailored to the conditions of a traditional paper publication. The XML source files contain much more information in an easily parseable form and will be made public simultaneously with the printed version to appear autumn 2008 in the same journal as [8].

Scholars throughout the world will be able to contribute, enhance or even fork new versions of the fileset if they deem so.

5 Related Work

During the HypereiDoc project, a number of existing related projects have been carefully revised.

Gothic Bible and minor fragments [20] are using TEI P4 without any reference to Epidoc. No overlapping annotation occurs in this project. Perseus Digital Library [24] is a huge library of TEI documents without any annotations. Aphrodisias Project at UNC and Kings College [12] is using Epidoc XML, but no overlapping annotation occurs. Digital Library Production Services at University of Virginia Library (DLPS) [16] uses TEI P4 with local modifications. The project adapts the standard to their needs. Center for Hellenic Studies - The Homer Multitext Project [15] is a TEI Core structure with embedded pictures instead of textual transcription. Most of Oxford Text Archive’s [23] projects use the SGML (not the XML) version of TEI for the header only with simple ASCII text representation. The Newton Project uses TEI XML with nesting but without overlapping, and is not related to epigraphy. Cambridge University Press [14] is publishing CD-ROM versions of English literary classics, including the works of Chaucer, Shakespeare, Samuel Johnson, and John Ruskin. These projects are not related to epigraphy. UVA Library, University of Virginia Text Center [29] is using TEI without overlaps, and is not related to epigraphy. Duke University Digitized Collections [17] has mostly 20th century texts, and is not related to epigraphy.

6 Conclusions

The HypereiDoc framework has been created to provide informatics background for transcribing literary, papyrological or epigraphical documents. An XML-based multi-layered structure is introduced to allow distributed, version-controlled work of scholars in a cooperative way. Handling of philological notations, associated interpretations, and commentaries are supported by an extensive reference system

based on XML pointers. The expressive power of the defined document model exceeds the capability of the other proposals.

A software package of supporting tools was created to provide a convenient interface for recording information in an error-free way, validating and building, as well as for the creation of critical editions. The solutions provide the greatest portability possible between operating systems with respect to both the tools and the finished documents.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the WEB – From Relations to Semistructured Data and XML, W3C Proposed Edited Recommendation, October 30, 2003, San Francisco (2000) ISBN 1-55860-622-X
2. Bauman, S.: TEI HORSEing Around. In: Proceedings of Extreme Markup Languages (2005)
3. De Rose, S.: Markup Overlap: A Review and a Horse. In Proceedings of Extreme Markup Languages (2004)
4. van Groningen, B.A.: De signis criticis in edendo adhibendis. *Menemosyne* 59, 362–365 (1932)
5. Lavagnino, J., Wujastyk, D.: Critical Edition Typesetting: The EDMAC format for plain \TeX . San Francisco and Birmingham, \TeX Users Group and UK \TeX Users Group (1996)
6. Netz, R., Noel, W.: The Archimedes Codex. Revealing The Secrets Of The World's Greatest Palimpsest, London (2007) ISBN-13: 9780297645474
7. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly, Sebastopol (2007)
8. Tchernetska, N.: New Fragments of Hyperides from the Archimedes Palimpsest. *Zeitschrift für Papyrologie und Epigraphik* 154, 1–6 (2005)
9. Tchernetska, N., Handley, E.W., Austin, C.F.L., Horváth, L.: New Readings in the Fragment of Hyperides' Against Timadros. *Zeitschrift für Papyrologie und Epigraphik* 162, 1–4 (2007)
10. Thomas, D., Fowler, C., Hunt, A.: Programming Ruby: The Pragmatic Programmers' Guide, p. 733. O'Reilly, Sebastopol (2004)
11. Walsh, N., Muellner, L.: DocBook: The Definitive Guide. O'Reilly, Sebastopol (1999)
12. The Aphrodisias Project, UNC and Kings College, <http://insaph.kcl.ac.uk/ala2004/>
13. Archimedes Palimpsest, <http://www.archimedespalimpsest.org/>
14. Cambridge University Press, <http://www.cup.cam.ac.uk/>
15. Center for Hellenic Studies – The Homer Multitext Project, http://www.chs.harvard.edu/publications.sec/homer_multitext.ssp
16. The Digital Library Production Services, http://www.lib.virginia.edu/digital/reports/teiPractices/dlpsPractices_postkb.html
17. Duke University Digitized Collections, <http://library.duke.edu/specialcollections/collections/digitized/>
18. Epidoc Guidelines, <http://www.stoa.org/epidoc/gl/5/toc.html>
19. Extensible Markup Language (XML) 1.0 (Third Edition), <http://www.w3.org/TR/2003/PER-xml-20031030>
20. The Gothic Bible, <http://www.wulfila.be/gothic/>

21. HypereiDoc Project Homepage, <http://hypereidoc.elte.hu/>
22. Microsoft Office Word 97-2007 Binary File Format, http://www.ecma-international.org/news/PressReleases/PR_TC45_Dec2006.htm
23. The Oxford Text Archive, <http://ota.ahds.ac.uk/>
24. The Perseus Digital Library, <http://www.perseus.tufts.edu/hopper/>
25. TEI P4 Guidelines, <http://www.tei-c.org/Guidelines/P4/index.xml>
26. TEI P4 Multiple Hierarchies, <http://www.tei-c.org/release/doc/tei-p4-doc/html/NH.html>
27. TEI P5 Guidelines, <http://www.tei-c.org/Guidelines/P5/index.xml>
28. TEI XPointer Supplements, <http://www.tei-c.org/release/doc/tei-p5-doc/en/html/SA.html>
29. University of Virginia Text Center - UVA Library, <http://etext.lib.virginia.edu/standards/tei/uvatei.html>
30. Wilson, P.: Ledmac, <ftp://dante.ctan.org/tex-archive/macros/latex/contrib/ledmac/>

Reducing Temporary Trees in XQuery

David Bednárek

Department of Software Engineering*
Faculty of Mathematics and Physics, Charles University Prague
david.bednarek@mff.cuni.cz

Abstract. The creation, maintenance and disposal of tree fragments during XQuery execution form a significant issue in the design of XQuery processors. The problem is further complicated by the definition of node identity which violates the functional nature of the XQuery language. This paper presents a novel mathematical model of XQuery execution that reflects temporary tree construction and manipulation, including navigation. Using this model as reference, an efficient algorithm of static analysis is presented that determines the level of information required at a particular place of the XQuery program. As a side effect, the algorithm also decides on the ordered/unordered context as defined by the XQuery language. Based on this algorithm, the amount of information stored during the execution as well as the complexity of operations may be significantly reduced.

1 Introduction

The recent development in the area of query languages for XML shows that the XQuery language will likely be used as one of the main application development languages in the XML world [2]. This shift from a query language towards a universal (although still based on the functional paradigm) programming language will be accompanied by increased complexity of XQuery programs. In particular, intensive use of user-defined functions and temporal data structures may be expected.

Contemporary XQuery processing and optimization techniques are usually focused on querying and, in most cases, ignore the existence of user-defined functions. One of the rare exceptions may be found in the stream-processing engine described in [4]. In the era of XSLT 1.0, the implementation techniques had to recognize user-defined functions (templates) well (see for instance [6]); however, this branch of research appears discontinued as the community shifted to XQuery.

Temporally existing trees were introduced in the XQuery 1.0 and XSLT 2.0 standards and many XQuery processing models can handle them. Examples may be found in the [13], based on the TLC algebra, or in the theoretical study on the tree-transformation power of XSLT [10]. However, the ignorance of user-defined

* Project of the program “Information society” of the Thematic program II of the National research program of the Czech Republic, No. 1ET100300419.

functions rendered the problem simpler than it really was. In the presence of user-defined functions, more sophisticated models and algorithms are required.

In this paper, we define a mathematical model of the evaluation of a XQuery program. The model is based on the following principles:

- Nodes within a tree are identified by *node identifiers* using *Dewey ID* labeling scheme.
- A tree is encoded using a mapping of Dewey labels to *node properties*.
- A tree created during XQuery evaluation is identified by a *tree identifier* derived from the context in which the tree was constructed.
- A node is globally identified by the pair of a tree identifier and a node identifier.
- A sequence is modeled using a mapping of *sequence identifiers* to *sequence items*.
- Each *sequence* containing nodes is accompanied by a *tree environment* which contains the encoding of the trees to which the nodes of the sequence belong.
- Evaluating a `for`-expression corresponds to iteration through all sequence identifiers in the value of the `in`-clause.
- A particular context reached during XQuery evaluation is identified by the pair of a *call stack*, containing positions in the program code, and a *control variable stack*, containing sequence identifiers selected by the `for`-expressions along the call stack.
- Node identifiers, tree identifiers, sequence identifiers, and control variable stacks share the same domain of *hierarchical strings*, allowing to construct each kind of identifier from the others.

There is a handful of related model variants: The *canonical* model exactly reflects the evaluation defined by the XQuery standard, other models are used to reduce the cost of evaluation by reducing the space of sequences and/or their tree environments. Optimized models are applied in the locations of the XQuery program determined by the backward analysis algorithm presented in this paper.

Each model consists of a group of *relations*, i.e. sets of tuples. Members of the tuples, i.e. *attributes* are either hierarchical strings or atomic values from the XML universe of simple types. Basic operations on these relations are described using notation derived from *relational algebra*. All the basic XQuery operators (FLWOR expressions, concatenation, node-set operators, node construction, and navigation) are expressed in terms of union, set difference, natural join, selection, projection, and function application.

Several XML processing models based on the relational algebra were already published [3,11,12]. The main contribution of this paper is the incorporation of user-defined functions into the model and the backward analysis algorithm.

The rest of the paper is organized as follows: In the Section 2, an abstraction of a XQuery program is described and three kinds of mathematical models of program execution are presented. In the third section, *requirement vectors and matrices* are defined as the base of the static analysis of a program with respect to the previously defined models. A backward propagation algorithm used to compute these matrices is then presented.

2 Mathematical Models

There are several variants of *core* subsets of XQuery, including the *core grammar* defined in the W3C standard [14], the LixQuery framework [8], and others [5,9]. Since the XSLT and XQuery are related languages and the translation from XSLT to XQuery is known (see [5]), the model may be applied also to XSLT. In our model, a selection of operators close to the standard core is used:

- declare function and function calls
- for-clause, let-clause, where-clause, stable-order-by-clause
- quantified expressions
- concatenation (,) operator
- **union**, **intersection**, **except** operators
- statically named document references (**fn:doc**)
- forward/reverse axis navigation, name tests, **fn:root**
- element constructors
- operators on atomic singletons (including Boolean operators)
- implicit conversions (atomization, effective Boolean value)

Due to space limitations in this paper, not all the models are presented.

2.1 Preliminaries

Hierarchical Strings. *Hierarchical alphabet* is an (infinite) totally ordered set Σ with a function $\alpha : \Sigma^* \rightarrow \Sigma$ that is a homomorphism with respect to the (lexicographical) ordering, $\alpha(u) < \alpha(w) \Leftrightarrow u < w$. Additionally, the alphabet shall contain all natural numbers, $\mathcal{N} \subseteq \Sigma$, and zero (0) shall be the minimal element of Σ . *Hierarchical string* is a word over hierarchical alphabet, a member of Σ^* . Besides the empty string (λ) and concatenation operator (\cdot), we will need the following functions: $rtrim(x) = y$ such that $y.a = x$ for some $a \in \Sigma$, $after(x, y) = z$ such that $x = y.z$, and the predicate $prefix(y, x) = (\exists z)(x = y.z)$. We will use the abbreviation $\alpha(x_1, x_2, \dots, x_n)$ for $\alpha(\alpha(x_1).\alpha(x_2).\dots.\alpha(x_n))$.

Relational Algebra Notation. We will use the following operators borrowed from classical relational algebra:

- *Union* and *set difference*, $R \cup S$ and $R \setminus S$, on two relations with the same set of attributes.
- *Natural join*, $R \bowtie S$, as a Cartesian product tied by equivalence on common attributes.
- *Selection*, $\sigma_{P(a_1, \dots, a_n)}(R)$, based on a predicate P .
- *Projection*, $\pi_{a_1, \dots, a_n}(R)$, reducing the relation R to the attributes a_1, \dots, a_n . For removing attributes, we will also use the abbreviation $\pi_{\setminus a_1, \dots, a_n}(R)$ for $\pi_{A_R \setminus \{a_1, \dots, a_n\}}(R)$ where A_R is the set of attributes of R .
- *Rename*, $\rho_{b/a}(R)$, renaming the attribute a to b .

Additionally, we define the operator of *function application* $\Phi_{b=f(a_1, \dots, a_n)}(R)$ which adds a new attribute b to the relation R , based on the function f and the values of the attributes a_1, \dots, a_n .

We will use the notation $R \subseteq (a_1 : T_1, \dots, a_n : T_n)$ to declare that R is a relation with attributes a_1, \dots, a_n from domains T_1, \dots, T_n .

2.2 Abstraction of the XQuery Program

Similarly to the normative definition of the XQuery semantics, we use (abstract) grammar rules of the *core grammar* [14] as the base for the models. A XQuery program is formalized as a forest of abstract syntax trees (AST), one tree for each user-defined function and one for the main expression. Each node of each AST has a (program-wide) unique *address* $E \in \text{ADDR}$; we assume that $\text{ADDR} \subseteq \Sigma$. The set functions $\subseteq \text{ADDR}$ enumerates all roots of ASTs assigned to functions, the node $\text{main} \in \text{ADDR}$ denotes the root of the main expression.

```

declare function local:toc($P as element()) as element()*
{
  for $X in $P/section
  return <section> {
    $X/@* , $X/title , local:toc($X)
  } </section>
};

<toc> {
  for $S in $I/book return local:toc($S)
} </toc>
    
```

Fig. 1. Query 1

As an example, we will use the Use Case TREE – Query 1 from the XQuery Test Suite [15], shown at Fig. 1. Fig. 2 shows the corresponding abstract syntax forest. Node labels are shown as letters left to the nodes.

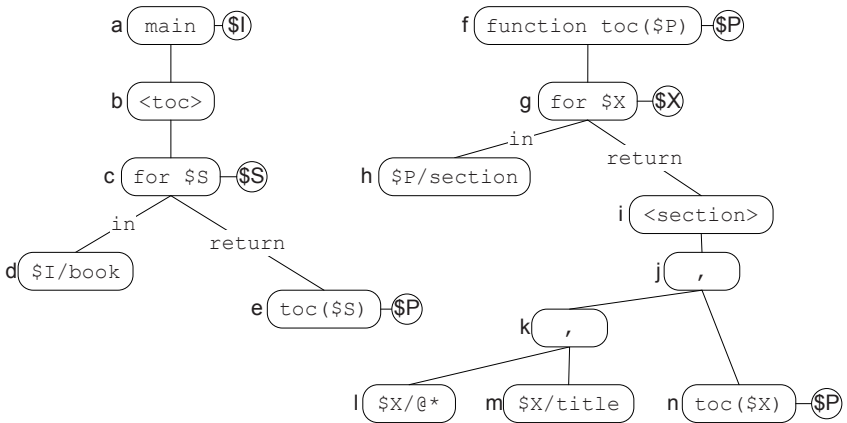


Fig. 2. Query 1 – Forest model

For each AST node E , the set $\text{vars}[E] \subseteq \text{QName}$ contains the names of *accessible variables*. In particular, when $E \in \text{functions}$, $\text{vars}[E]$ contains the names of arguments of the function E , including implicit arguments like the context node.

In the backward propagation algorithm, we will use the following structures derived from the AST: $\text{decl}[C, \mathbf{\$x}]$ is the node where a variable $\mathbf{\$x}$, accessible from a node C , was declared, $\text{calls}[D]$ is the set of all calls to a function D , $\text{actual}[C, \mathbf{\$x}]$ is the actual expression associated to the formal parameter $\mathbf{\$x}$ in the call C , $\text{formal}[F, i]$ is the name of the i -th formal parameter of the function F .

2.3 Common Model Structure

Each model is based on the following data structures:

- $\text{inv}[E]$ is the set of contexts in which the expression $E \in \text{ADDR}$ is evaluated.
- $\text{var}[E, \mathbf{\$x}]$ represents the assignment of the values of the variable $\mathbf{\$x} \in \text{vars}[E]$ to the contexts from $\text{inv}[E]$. Besides user-defined and implicit (\cdot , fn:root) variables, this structure is also used to represent the external documents accessible via the fn:doc function.
- $\text{ex}[E]$ represents the assignment of the result value of the expression E to the contexts from $\text{inv}[E]$.

$\text{var}[E, \mathbf{\$x}]$ and $\text{ex}[E]$ are compounds containing several relations, based on the particular model. We will access these relations using dot notation.

Each model defines the semantics of each (applicable) XQuery operator, using equations over our relational algebra notation. These equations bind together the values of the relations associated to the whole expression and the values associated to its operand. The equations are always formulated as assignments; thus, the model may be viewed as an attribute grammar, having inherited attributes $\text{inv}[E]$ and $\text{var}[E, \mathbf{\$x}]$, and synthesized attributes $\text{ex}[E]$. However, this analogy works only for stand-alone XQuery expressions; when function calls are involved, the definition must be more precise:

Given a program P containing AST nodes $\text{ADDR}_P \subset \text{ADDR}$, *execution* of the program is any assignment to the $\text{inv}[E]$, $\text{var}[E, \mathbf{\$x}]$, and $\text{ex}[E]$ variables that satisfies all the equations of the model. For the main expression $E_P \in \text{ADDR}_P$, the set of executions induces a relation between the values $\text{var}[E_P, \text{doc}_i]$ of input documents $\text{doc}_1, \dots, \text{doc}_n$ and the value $\text{ex}[E_P]$ of program output. The fact that all model constraints are in the form of assignments causes that there is at most one output for a given input. (Note that the proof of this property is not trivial and it is beyond the space limitations of this paper.)

2.4 Canonical Model

The canonical model directly reflects the definition of XQuery semantics. It uses the following relations:

$$\text{inv}[E] \subseteq (i : \text{ADDR}^*, f : \Sigma^*)$$

$$\text{var}[E, \$x].\text{seqa}, \text{ex}[E].\text{seqa} \subseteq (i : \text{ADDR}^*, f : \Sigma^*, s : \Sigma^*, v : \text{ATOMIC})$$

$$\text{var}[E, \$x].\text{seqn}, \text{ex}[E].\text{seqn} \subseteq (i : \text{ADDR}^*, f : \Sigma^*, s : \Sigma^*, t : \text{TI}, n : \Sigma^*)$$

$$\text{var}[E, \$x].\text{env}, \text{ex}[E].\text{env} \subseteq (i : \text{ADDR}^*, f : \Sigma^*, t : \text{TI}, n : \Sigma^*, a : \text{NODEDATA})$$

A `.seqa` relation describes atomic members of sequences, the corresponding `.seqn` relation contains node members of the same sequence (the two portions are interweaved using the sequence identifiers s). The `.env` relation models the tree environment, i.e. the trees in which the nodes of the corresponding sequence are contained. Since the manipulation with the `.seqa` relations is similar to the handling of the `.seqn` relations, we will omit equations for `.seqa` from the definitions shown below, for brevity.

Attribute i represents the call stack that brought the execution to the examined expression E , including it (i.e., $i = j.E$ for some j). f is the stack of sequence identifiers selected by the `for`-clauses throughout the descent along i . The two stacks together form the identification of the dynamic context in which an expression is evaluated. While the XQuery standard defines dynamic context as the set of variable assignments (with some negligible additions), our notion of dynamic context is based on the stack pair that determines the descent through the code to the examined expression, combining both the code path stored in i and the `for`-control variables in f . The key to the sufficiency of this model is the observation that the variable assignment is a function of the stack pair. (Note that this function is not necessarily an injection; therefore, our model may repeatedly evaluate an expression in the same variable assignment. Although it may seem to be a flaw for a declarative language interpreter, it exactly reflects the node identity generation by constructors.)

Attribute s is a sequence identifier; for a given $\langle i, f \rangle$, the set of s values form a prefix-less language.

v is a value of an atomic type as defined by the XQuery standard.

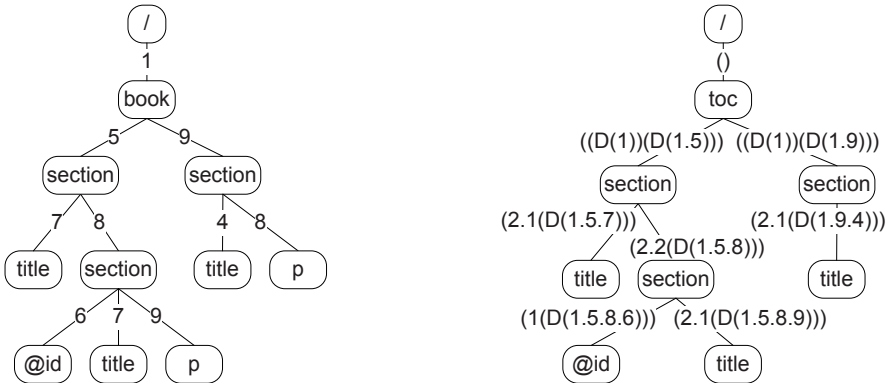


Fig. 3. Query 1 – Sample input and output documents

t is a tree identifier of type $\text{TI} = (i : \text{ADDR}^*, f : \Sigma^*)$; its parts i and f correspond to the environment identification at the moment of tree creation. Note that the trees of external documents shall also have tree identifiers of such structure, like if these trees were created at the start of the program.

n is a node identifier in the form of a Dewey ID. a is a tuple of properties assigned to a node by the XML Data Model, containing node kind, name, typed and string values, etc.

Fig. 3 shows an input document related to the example program at fig. 2; note that text nodes are omitted. The input document is given a unique tree identifier D and encoded using Dewey-based relation passed as the environment of the global variable $\$I$, while the variable value itself is a singleton reference to the document root:

$$\begin{aligned} \text{var}[a, \$I].\text{seqn} &= (a \ \lambda \ \lambda \ D \ \lambda) \\ \text{var}[a, \$I].\text{env} &= \begin{pmatrix} a \ \lambda \ D \ \lambda & \langle \text{document} \rangle \\ a \ \lambda \ D \ 1 & \langle \text{element, book} \rangle \\ a \ \lambda \ D \ 1.5 & \langle \text{element, section} \rangle \\ a \ \lambda \ D \ 1.5.7 & \langle \text{element, title} \rangle \\ a \ \lambda \ D \ 1.5.8 & \langle \text{element, section} \rangle \\ a \ \lambda \ D \ 1.5.8.6 & \langle \text{attribute, id} \rangle \\ a \ \lambda \ D \ 1.5.8.7 & \langle \text{element, title} \rangle \\ a \ \lambda \ D \ 1.5.8.9 & \langle \text{element, p} \rangle \\ a \ \lambda \ D \ 1.5.9 & \langle \text{element, section} \rangle \\ a \ \lambda \ D \ 1.5.9.4 & \langle \text{element, title} \rangle \\ a \ \lambda \ D \ 1.5.9.8 & \langle \text{element, p} \rangle \end{pmatrix} \end{aligned}$$

As the execution of the program proceeds, the function `toc` is called four times - this is expressed in the context set $\text{inv}[f]$ as well as in the model of the variable $\$P$. Note that the variable is always a singleton, therefore, its model contains empty sequence identifiers:

$$\begin{aligned} \text{inv}[f] &= \begin{pmatrix} \text{e.f} & (D(1)) \\ \text{e.n.f} & (D(1))(D(1.5)) \\ \text{e.n.f} & (D(1))(D(1.9)) \\ \text{e.n.n.f} & (D(1))(D(1.5))(D(1.5.8)) \end{pmatrix} \\ \text{var}[f, \$P].\text{seqn} &= \begin{pmatrix} \text{e.f} & (D(1)) & \lambda \ D \ 1 \\ \text{e.n.f} & (D(1))(D(1.5)) & \lambda \ D \ 1.5 \\ \text{e.n.f} & (D(1))(D(1.9)) & \lambda \ D \ 1.9 \\ \text{e.n.n.f} & (D(1))(D(1.5))(D(1.5.8)) & \lambda \ D \ 1.5.8 \end{pmatrix} \end{aligned}$$

Encoding of a sequence is shown at the relation $\text{ex}[j].\text{seqn}$ which corresponds to the value of the concatenation operator at node j . This operator is evaluated three times, returning sequences of two nodes in two cases. Input nodes and constructed nodes are mixed here, which is expressed in the use of different tree identifiers.

$\text{ex}[j].\text{seqn} =$

$$\left(\begin{array}{l} \text{e.j, } (D(1))(D(1.5)), 2.1(D(1.5.7)), D, 1.5.7 \\ \text{e.j, } (D(1))(D(1.5)), 2.2(D(1.5.8)), (\text{e.n.i})((D(1))(D(1.5))(D(1.5.8))), \lambda \\ \text{e.j, } (D(1))(D(1.9)), 2.1(D(1.9.4)), D, 1.9.4 \\ \text{e.n.j, } (D(1))(D(1.5))(D(1.5.8)), 1(D(1.5.8.6)), D, 1.5.8.6 \\ \text{e.n.j, } (D(1))(D(1.5))(D(1.5.8)), 2.1(D(1.5.8.9)), D, 1.5.8.9 \end{array} \right)$$

The value associated to the node j becomes the child list of a constructor; therefore, the tree identifiers t are replaced to obey the required copy semantics and node identifiers n are stripped during subtree extraction. On the other hand, the sequence identifiers s are used as edge labels in the output document – they may be found in the final output of the program shown at Fig. 3.

Most XQuery operators do not change the assignment of variable values. However, in our model, the values carry the i attribute which determines the place where the value is examined. Therefore, when the execution descends through an operator from an AST node E_0 to a node E_1 , the $\text{inv}[E]$ and $\text{var}[E, \$\mathbf{x}]$ values formally change as follows, using an auxiliary operator jmp :

$$\begin{aligned} \text{jmp}_{E_0} &= \rho_{i/j} \pi_{\setminus i} \Phi_{j=rtrim(i).E} \\ (\forall \$\mathbf{x} \in \text{vars}[E_0]) \text{var}[E_1, \$\mathbf{x}].\text{seqn} &= \text{jmp}_{E_1}(\text{var}[E_0, \$\mathbf{x}].\text{seqn}) \\ (\forall \$\mathbf{x} \in \text{vars}[E_0]) \text{var}[E_1, \$\mathbf{x}].\text{env} &= \text{jmp}_{E_1}(\text{var}[E_0, \$\mathbf{x}].\text{env}) \end{aligned}$$

for-expression and function call are an exception from this rule; their jmp and var equations are shown below.

Concatenation – $E_0 = E_1 , E_2$

$$\begin{aligned} \text{ex}[E_0].\text{seqn} &= \rho_{s/r}((\Phi_{r=1.s} \text{jmp}_{E_0}(\text{ex}[E_1].\text{seqn})) \cup (\Phi_{r=2.s} \text{jmp}_{E_0}(\text{ex}[E_2].\text{seqn}))) \\ \text{ex}[E_0].\text{env} &= (\text{jmp}_{E_0}(\text{ex}[E_1].\text{env})) \cup (\text{jmp}_{E_0}(\text{ex}[E_2].\text{env})) \end{aligned}$$

Note that whenever the two environments $\text{ex}[E_1].\text{env}$ and $\text{ex}[E_2].\text{env}$ contain the same tree identifier t , the corresponding tree information is merged via set-union. Since the tree identifier exactly determines the context in which the tree was created, trees having the same identifier must be identical; therefore, applying set-union to tree environments do not alter them anyway.

Node-Set Union – $E_0 = E_1 \text{ union } E_2$

$$\begin{aligned} \text{docorder} &= \Phi_{s=\alpha(t,n)} \\ \text{ex}[E_0].\text{seqn} &= \text{docorder}((\pi_{\setminus s} \text{jmp}_{E_0}(\text{ex}[E_1].\text{seqn})) \cup (\pi_{\setminus s} \text{jmp}_{E_0}(\text{ex}[E_2].\text{seqn}))) \\ \text{ex}[E_0].\text{env} &= (\text{jmp}_{E_0}(\text{ex}[E_1].\text{env})) \cup (\text{jmp}_{E_0}(\text{ex}[E_2].\text{env})) \end{aligned}$$

The **intersection** and **except** operators are modeled similarly, using natural join and set difference instead of set union.

Node Construction – $E_0 = \langle a \rangle \{ E_1 \} \langle /a \rangle$

$$\begin{aligned} \text{ex}[E_0].\text{seqn} &= \Phi_{s=\lambda} \Phi_{t=\alpha(i,f)} \Phi_{n=\lambda} (\text{inv}[E_0]) \\ \text{ex}[E_0].\text{env} &= \Phi_{t=\alpha(i,f)} ((\Phi_{n=\lambda} \Phi_{v=\mathbf{a}} (\text{inv}[E_0]))) \\ &\cup (\pi_{i,f,n,v} \Phi_{n=\alpha(s)}. \text{after}(q,p) \sigma_{\text{prefix}(p,q)} ((\rho_{p/n} \text{jmp}_{E_0} (\text{ex}[E_1].\text{seqn})) \\ &\quad \bowtie (\pi_{i,f,q,v} \rho_{q/n} \text{jmp}_{E_0} (\text{ex}[E_1].\text{env})))))) \end{aligned}$$

Navigation – $E_0 = E_1 / \text{axis} : : *$

$$\begin{aligned} \text{ex}[E_0].\text{seqn} &= \text{docorder } \pi_{i,f,t,n} \sigma_{P(q,n)} \\ &((\pi_{i,f,t,n} \text{jmp}_{E_0} (\text{ex}[E_0].\text{env})) \bowtie (\pi_{i,f,t,q} \rho_{q/n} \text{jmp}_{E_0} (\text{ex}[E_0].\text{seqn}))) \\ \text{ex}[E_0].\text{env} &= (\text{jmp}_{E_0} (\text{ex}[E_0].\text{env})) \bowtie (\pi_{i,f,t} \sigma_{P(q,n)}) \\ &((\pi_{i,f,t,n} \text{jmp}_{E_0} (\text{ex}[E_0].\text{env})) \bowtie (\pi_{i,f,t,q} \rho_{q/n} \text{jmp}_{E_0} (\text{ex}[E_0].\text{seqn}))) \end{aligned}$$

The selection operator is driven by a predicate P applied to node identifiers q (a node from the sequence) and n (a node from its tree environment). The predicate is selected according to the *axis* used in the navigation operator, for instance, the *prefix* predicate for the **descendant-or-self** axis.

For Expression – $E_0 = \text{for } \$y \text{ in } E_1 \text{ return } E_2$

$$\begin{aligned} \text{down}_{E_0} &= \Phi_{g=f.\alpha(s)} \text{jmp}_{E_0} \\ \text{inv}[E_2] &= \rho_{f/g} \pi_{i,g} \text{down}_{E_2} (\text{ex}[E_1].\text{seqn}) \\ \text{var}[E_2, \$y].\text{seqn} &= \Phi_{s=\lambda} \rho_{f/g} \pi_{\setminus f, s} \text{down}_{E_2} (\text{ex}[E_1].\text{seqn}) \\ \text{var}[E_2, \$y].\text{env} &= \rho_{f/g} \pi_{\setminus f} \\ &((\text{down}_{E_2} (\text{ex}[E_1].\text{env})) \bowtie (\pi_{i,f,t,g} \text{down}_{E_2} (\text{ex}[E_1].\text{seqn}))) \\ (\forall \$x \in \text{vars}[E_0] \setminus \{\$y\}) \text{var}[E_2, \$x].\text{seqn} &= \rho_{f/g} \pi_{\setminus f} \\ &((\text{jmp}_{E_2} (\text{var}[E_0, \$x].\text{seqn})) \bowtie (\pi_{i,f,g} \text{down}_{E_2} (\text{ex}[E_1].\text{seqn}))) \\ (\forall \$x \in \text{vars}[E_0] \setminus \{\$y\}) \text{var}[E_2, \$x].\text{env} &= \rho_{f/g} \pi_{\setminus f} \\ &((\text{jmp}_{E_2} (\text{var}[E_0, \$x].\text{env})) \bowtie (\pi_{i,f,g} \text{down}_{E_2} (\text{ex}[E_1].\text{seqn}))) \\ \text{ex}[E_0].\text{seqn} &= \pi_{\setminus g, p, r} \Phi_{s=p.r} \text{jmp}_{E_0} \\ &((\rho_{r/s} \rho_{g/f} (\text{ex}[E_2].\text{seqn})) \bowtie (\pi_{i,f,g,p} \rho_{p/s} \text{down}_{E_2} (\text{ex}[E_1].\text{seqn}))) \\ \text{ex}[E_0].\text{env} &= \pi_{\setminus g} \text{jmp}_{E_0} \\ &((\rho_{g/f} (\text{ex}[E_2].\text{env})) \bowtie (\pi_{i,f,g} \text{down}_{E_2} (\text{ex}[E_1].\text{seqn}))) \end{aligned}$$

Order-by Clause – $E_0 = \text{for } \$y \text{ in } E_1 \text{ order by } E_2 \text{ return } E_3$

$$\begin{aligned} \text{ex}[E_0].\text{seqn} &= \pi_{\setminus g, p, r} \Phi_{s=\alpha(v).p.r} \text{jmp}_{E_0} ((\rho_{r/s} \rho_{g/f} (\text{ex}[E_3].\text{seqn})) \\ &\quad \bowtie (\pi_{i,g,v} \rho_{g/f} \text{jmp}_{E_3} (\text{ex}[E_2].\text{seqn}))) \\ &\quad \bowtie (\pi_{i,f,g,p} \rho_{p/s} \text{down}_{E_3} (\text{ex}[E_1].\text{seqn}))) \end{aligned}$$

The other equations are identical to the equations of the plain for-expression. Note that in the case of multi-variable for-expression with order-by clause, the $\text{ex}[E_0].\text{seqn}$ equation shall handle all the control variables at once.

Where Clause – $E_0 = \text{where } E_1 \text{ return } E_2$

$$\begin{aligned}
\text{inv}[E_2] &= \pi_{i,f} \sigma_{v=true} \text{jmp}_{E_2}(\text{ex}[E_1].\text{seqa}) \\
(\forall \$x \in \text{vars}[E_0]) \text{var}[E_2, \$x].\text{seqn} &= \\
&((\text{jmp}_{E_2}(\text{var}[E_0, \$x].\text{seqn})) \bowtie (\pi_{i,f} \sigma_{v=true} \text{jmp}_{E_2}(\text{ex}[E_1].\text{seqa}))) \\
(\forall \$x \in \text{vars}[E_0]) \text{var}[E_2, \$x].\text{env} &= \\
&((\text{jmp}_{E_2}(\text{var}[E_0, \$x].\text{env})) \bowtie (\pi_{i,f} \sigma_{v=true} \text{jmp}_{E_2}(\text{ex}[E_1].\text{seqa}))) \\
\text{ex}[E_0].\text{seqn} &= \text{jmp}_{E_0}(\text{ex}[E_2].\text{seqn}) \\
\text{ex}[E_0].\text{env} &= \text{jmp}_{E_0}(\text{ex}[E_2].\text{env})
\end{aligned}$$

2.5 Reduced Models

In the operator models described above, often an attribute is not used in the right-hand side of the equations. For instance, all node-set operators ignore the s attribute of the `.seqn` relation of their operands. In the terms of the relational algebra, there is a projection operator that removes the attribute. The projection operator is distributive over other relational algebra operators, provided the operators do not reference attributes removed by the projection. Thus, the projection operator may be moved against the flow of computation until a barrier formed by a set difference, a natural join or a selection operator that references the removed column. The projection operator may also annihilate with a function application operator that defines the removed column.

With the removal of an attribute, the model of an expression is shrunk to a *reduced model*. Thus, a reduced model is generated “automatically” from the canonical model, by applying projection to the relations of the canonical model.

2.6 The CC Model

When a constructed tree is used as a child in the construction of another node, the standard requires that a copy of the tree be made, assigning the newly created tree a new identifier. Therefore, the original tree identifier t is not used except for the binding between a sequence and its tree environment. Due to the use in a natural join operator, the tree identifier t cannot be removed; however, it is no longer required that the tree identifier be computed according to the canonical model. Thus, the canonical tree identifier based on the identification of the place of its creation may be replaced by any identifier that allows the binding between sequences and environments. In the constructor-to-constructor model (*CC model*), the sequence identifier s is used instead of the canonical tree identifier. Moreover, when no navigation takes place on the constructed trees, sequences may contain only the roots of the constructed trees, whose node identifiers are empty and may be omitted. Using the abovementioned simplification, we get the following model:

$$\begin{aligned}
\text{inv}[E] &\subseteq (i : \text{ADDR}^*, f : \Sigma^*) \\
\text{var}[E, \$x].\text{cseq}, \text{ex}[E].\text{cseq} &\subseteq (i : \text{ADDR}^*, f : \Sigma^*, s : \Sigma^*) \\
\text{var}[E, \$x].\text{cenv}, \text{ex}[E].\text{cenv} &\subseteq (i : \text{ADDR}^*, f : \Sigma^*, s : \Sigma^*, n : \Sigma^*, a : \text{NODEDATA})
\end{aligned}$$

Note that the CC model is not derived from the canonical model by attribute removal. Since the model is not able to express all XQuery values (in particular, atomic values), there must be a conversion from the canonical model to the CC model, applied whenever an operator (e.g. navigation) cannot produce the CC model directly. The conversion is necessarily lossy; however, it preserves the information required at the child list of a constructor.

The CC model is applied as follows: Every constructor requires that its child list be presented in the CC model. Operators like concatenation or for-expression work, in parallel and independently, in both the canonical and CCC models. Operators that cannot consume operands in the CC model are specified in the canonical model, followed by a conversion to the CC model. Thus, each value is formally represented twice, allowing its consumer to select the most suitable form.

Unlike the reduced models, the CC model is not derived by projection from the canonical model; however, both the models may be derived by projection from the supermodel composed of the two models. This observation is later used as the formal basis of the static analysis.

3 Static Analysis

Whenever an attribute is not used in an operator, the corresponding projection operator may be moved against the data-flow. In an abstract syntax tree, the projection operators may move downwards, i.e. against the flow of the *ex* relations, or upwards, towards the definition of variables. Since the projection operators may move in both directions, the possibility to shrink a model strongly depends on the context where the expression is evaluated. In a user-defined function, it may depend on the context where the function was called. More precisely, available model reduction depends on the *i* attribute of the *inv* relation (note that the reducibility does not depend on the *f* attribute). Thus, available reduction (i.e. the set of attributes that may be removed) is described by the following functions:

$$\begin{aligned} \text{rex} &: \text{ADDR}^* \rightarrow \text{REQ} \\ \text{rvar} &: \text{ADDR}^* \times \text{QName} \rightarrow \text{REQ} \end{aligned}$$

where $\text{REQ} = (s : \mathcal{B}, t : \mathcal{B}, n : \mathcal{B}, a : \mathcal{B}, v : \mathcal{B}, c : \mathcal{B})$ is a vector of Boolean *requirement flags* named after the attributes of the model relations (the *c* flag stands for the whole CC model). The Boolean flags are assigned the value 1 if the corresponding attribute (or set of attributes) is *required*, 0 if it may be removed. Note that we do not trace the usage of *i* and *f* attributes since they are used in every model equation. Moreover, we use the *t* and *n* flags for the respective attributes in both the *.seqn* and *.env* relations since they are always used simultaneously.

The vector $\text{rex}[i.E]$ contains the requirement flags for the relations in $\text{ex}[E]$ when evaluated in the context of the call stack *i.E*. The vector $\text{rvar}[i.E, \$\mathbf{x}]$ is similarly related to $\text{var}[i.E, \$\mathbf{x}]$.

The $\text{rex}(i)$ and $\text{rvar}[i, \$\mathbf{x}]$ vectors correspond to the behavior of operators on all paths from the invocation *i* to the output of the program. More exactly,

the vectors are determined by a matrix product (in the Boolean algebra) of elementary *requirement matrices* $\text{opm}_{op,k}$ associated to the k -th operand of each operator op on a path; the products associated to paths are then collected by matrix addition, i.e. Boolean OR. The matrix product is computed using the following *requirement matrices* associated to each function/argument pair:

$$\text{rvarm} : \text{functions} \times \text{QName} \rightarrow \overline{\text{REQ}} \times \overline{\text{REQ}}$$

For a function $F \in \text{functions}$ and its formal argument $\$x \in \text{vars}(f)$, the matrix $\text{rvarm}[F, \$x]$ describes the dependence between the requirement flags $\text{rex}[F]$ of the function value and the requirement flags $\text{rvar}[F, \$x]$ of the formal argument $\$x$. The dependence is defined by the following equation in terms of matrix multiplication over the Boolean algebra:

$$\overline{\text{rvar}}[F, \$x] = \text{rvarm}[F, \$x] \cdot \overline{\text{rex}}[F]$$

The lines over $\overline{\text{REQ}} \times \overline{\text{REQ}}$ indicate that a column and a row were added to the matrix. Similarly, $\overline{\text{rvar}}$ and $\overline{\text{rex}}$ were augmented with a Boolean 1. This augmentation forms a trick, known from linear algebra, that allows adding a constant vector during the multiplication by the matrix.

The Boolean matrices $\text{rvarm}[F, \$x]$ are computed during *backward analysis* shown as Algorithm [11](#). The algorithm computes $\text{rexm}[E]$ matrices for each AST node E and $\text{rvarm}[E, \$x]$ matrices for each variable $\$x$ declared by a node E , including formal arguments. Each $\text{rexm}[E]$ matrix corresponds to the requirement effect of the path from E to the root of the function tree. The matrix is determined as the matrix product of matrices corresponding to the operators on the path. Similarly, a $\text{rvarm}[E, \$x]$ matrix corresponds to the requirement transition from the root of the function tree to the references to the variable. If there is more than one reference, their requirement effect is combined using Boolean \vee -operator.

Since recursion may exist among user-defined functions, the backward analysis algorithm computes all the matrices at once, in iterative manner. The matrices corresponding to return values of the functions are initialized with identities, the other expression matrices and all variable matrices are initialized with zeros (see lines 1 to 10 of Algorithm [11](#)). Then, the matrices are repeatedly increased until a stable state is reached. To keep track of changed matrices, the stack `stk` containing code-forest nodes is used. In the main cycle (lines 11 to 46), each node E removed from the stack is examined. If the node is an operator (lines 14 to 19), the matrix of each of its operands is recalculated using the elementary requirement matrix $\text{opm}_{op,i}$. If any change is detected, the operand is scheduled to reexamination using the stack. In this phase, for and let operators are handled as regular operators with two to four operands.

If a reference to a variable is encountered, the expression matrix is added (by Boolean OR) to the variable matrix (see line 21). If the variable matrix changes, the matrix of the expression that initializes the variable is reevaluated, depending on the type of variable: The requirement matrix of a for-control variables (line 25) affects the in-clause through the elementary requirement matrix `forvarm`; a

Algorithm 1. Backward analysis algorithm

```

1: for all  $E \in \text{ADDR}$  do
2:    $\text{rexm}[E] := 0$ 
3:   for all  $\$x \in \text{decl}[E]$  do
4:      $\text{rvarm}[E, \$x] := 0$ 
5:   end for
6: end for
7:  $\text{stk} := \text{empty}$ 
8: for all  $F \in \text{functions}$  do
9:    $\text{rexm}[F] := 1$  ;  $\text{stk.push}(F)$ 
10: end for
11: while not  $\text{stk.empty}$  do
12:    $E := \text{stk.pop}$ 
13:   if  $E \rightarrow \text{op}(E_1, \dots, E_n)$  then
14:     for all  $i \in \{1, \dots, n\}$  do
15:        $N := \text{rexm}[E_i] + \text{opm}_{\text{op}, i} \cdot \text{rexm}[E]$ 
16:       if  $N \neq \text{rexm}[E_i]$  then
17:          $\text{rexm}[E_i] := N$  ;  $\text{stk.push}(E_i)$ 
18:       end if
19:     end for
20:   else if  $E \rightarrow \$x$  then
21:      $D := \text{decl}[E, \$x]$  ;  $N := \text{rvarm}[D, \$x] + \text{rexm}[E]$ 
22:     if  $N \neq \text{rvarm}[D, \$x]$  then
23:        $\text{rvarm}[D, \$x] := N$ 
24:       if  $D \rightarrow \text{for } \$x \text{ in } E_1 \text{ return } E_2$  then
25:          $\text{rexm}[E_1] := \text{rexm}[E_1] + \text{forvarm} \cdot \text{rvarm}[D, \$x]$  ;  $\text{stk.push}(E_1)$ 
26:       else if  $D \rightarrow \text{let } \$x := E_1 \text{ return } E_2$  then
27:          $\text{rexm}[E_1] := \text{rvarm}[D, \$x]$  ;  $\text{stk.push}(E_1)$ 
28:       else
29:         for all  $C \in \text{calls}[D]$  do
30:            $A := \text{actual}[C, \$x]$  ;  $M := \text{rvarm}[D, \$x] * \text{rexm}[C]$ 
31:           if  $M \neq \text{rexm}[A]$  then
32:              $\text{rexm}[A] := M$  ;  $\text{stk.push}(A)$ 
33:           end if
34:         end for
35:       end if
36:     end if
37:   else if  $E \rightarrow F(E_1, \dots, E_n)$  then
38:     for all  $i \in \{1, \dots, n\}$  do
39:        $\$x := \text{formal}[F, i]$ 
40:        $N := \text{rvarm}[F, \$x] * \text{rexm}[E]$ 
41:       if  $N \neq \text{rexm}[E_i]$  then
42:          $\text{rexm}[E_i] := N$  ;  $\text{stk.push}(E_i)$ 
43:       end if
44:     end for
45:   end if
46: end while

```

let-variable (line 27) requirement is copied directly to the defining expression. The requirement matrix of a formal argument of a function (lines 29 to 34) must be propagated to the corresponding actuals in all calls to the function; in this case, the propagation is driven by the matrix product of the variable requirement and the call requirement.

When a function-call node is removed from the stack (lines 38 to 44), requirement matrices of all its actuals must be recalculated, using the same matrix product as in the abovementioned case of propagation from a formal.

The algorithm runs in $O(n^2 \cdot k^4)$ time and $O(n \cdot k^2)$ space where n is the size of the program and k is the size of the augmented requirement vector. In our system, $k = 7$; however, we are presenting the generalized complexity since the algorithm may be used to compute other types of backward-propagated requirement flags.

4 Conclusion

We have presented a group of mathematical models of XQuery evaluation, based on relations whose attributes are strings in a hierarchical alphabet, acting similarly to Dewey identifiers. Since the models use notation borrowed from the relational algebra, they can be more easily understood in the database community than models based on state-machines. The relational algebra axioms also back up any manipulation with the models. The system is open – new models may be added under the same notation and corresponding algorithms can be reused.

The canonical model described in the paper is used to reflect the definition of the XQuery semantics as closely as possible. It would be natural to prove the equivalence between our canonical model and the normative XQuery semantics; unfortunately, some of the most interesting parts of XQuery semantics, namely the effect of the order-by clause and the identity of constructed nodes, are described only informally, although under the title “Formal Semantics” [14]. Therefore, this paper belongs to the rare attempts to define the dark corners of the XQuery semantics more exactly.

The canonical model offers an option of “automatic” demotion whenever the consumers of an expression do not use all attributes of the model. One of the reduced models corresponds to the *unordered* context as defined by the XQuery standard; therefore, the static analysis algorithm described in this paper may also be used for the automatic determination of ordered and unordered contexts in a XQuery program. Methods of ordered/unordered context determination were already described; among them, *column dependency analysis* described in [7] is similar to our approach but based on a different algebra that does not handle user-defined functions.

Besides the reduced models, the CC model is defined to support the lifetime of tree fragments between their creation and their use in the construction of larger trees. This model uses the sequence identifiers as tree identifiers; the tree identifiers are directly transformed into the Dewey-ID-based encoding of the output tree.

Finally, an effective algorithm of static analysis is presented which evaluates the behavior of XQuery code with respect to the information required at any

node of its abstract syntax tree. The algorithm is applicable to any group of models based on the relational algebra.

The output of this algorithm may be interpreted in two ways: Conservatively, to assign the minimal correct mode of evaluation (for instance, the unordered set mode) to every operator in the program; or aggressively, to duplicate the functions in the program so that every call is evaluated exactly in the minimal mode.

The results of the analysis may be used in many ways, depending on the architecture of the XQuery interpreter. The impact of unordered evaluation was already studied for instance in [7].

In the CC-mode of evaluation, the most important factor is the absence of globally-unique tree identifiers, which in turn saves the cost of identifier generation. While it may be considered negligible in DOM-based systems, the effect may be important in implementations in distributed environments or in systems based on translation to SQL. Furthermore, trees constructed in the CC-mode will not be navigated; thus, their implementation may use simpler data structures – for instance, trees without child-to-parent backlinks or even serialized text if the output of the program is to be serialized.

Although the main purpose of the mathematical model is to give a background to the presented algorithm, it may also encourage new approaches in the implementation of XQuery engines. In particular, the CC model may be directly bound to a Dewey-ID-based storage (described, for instance, in [11]).

The models based on relational algebra may also offer new modes of optimization. For instance, the concatenation operator is rewritten using the natural join operator which is commutative; thus, transformations based on reordering of the operands become available.

There are two important omissions from the core XQuery that are not covered by our current model: Positional variables and aggregate functions. Future research will address them, probably inducing new modes of model reduction.

References

1. Boncz, P., Grust, T., van Keulen, M., Manegold, S., Rittinger, J., Teubner, J.: Pathfinder: XQuery—The Relational Way. In: Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.-Å., Ooi, B.C. (eds.) VLDB 2005: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 1322–1325. ACM, New York (2005)
2. Chamberlin, D.: XQuery: Where Do We Go from Here? In: XIMEP 2006, 3rd International Workshop on XQuery Implementation, Experiences and Perspectives. ACM Digital Library, New York (2006)
3. El-Sayed, M., Wang, L., Ding, L., Rundensteiner, E.A.: An Algebraic Approach for Incremental Maintenance of Materialized XQuery Views. In: WIDM 2002: Proceedings of the 4th International Workshop on Web Information and Data Management, pp. 88–91. ACM, New York (2002)
4. Fegaras, L., Dash, R., Wang, Y.: A Fully Pipelined XQuery Processor. In: XIMEP 2006, 3rd International Workshop on XQuery Implementation, Experiences and Perspectives. ACM, New York (2006)

5. Fokoue, A., Rose, K., Siméon, J., Villard, L.: Compiling XSLT 2.0 into XQuery 1.0. In: WWW 2005: Proceedings of the 14th International Conference on World Wide Web, pp. 682–691. ACM, New York (2005)
6. Groppe, S., Böttcher, S., Birkenheuer, G., Höing, A.: Reformulating XPath Queries and XSLT Queries on XSLT Views. Technical report, University of Paderborn (2006)
7. Grust, T., Rittinger, J., Teubner, J.: eXrQuy: Order Indifference in XQuery. In: 2007 IEEE 23rd International Conference on Data Engineering, pp. 226–235. IEEE Computer Society, Los Alamitos (2007)
8. Hidders, J., Michiels, P., Paredaens, J., Vercammen, R.: LixQuery: A Formal Foundation for XQuery Research. SIGMOD Rec. 34(4), 21–26 (2005)
9. Hloušek, P.: XPath, XQuery, XSLT: Formal Approach. PhD thesis, Charles University Prague (2005)
10. Janssen, W., Korlyukov, A., Van den Bussche, J.: On the Tree-Transformation Power of XSLT. Technical report, University of Hasselt (2006)
11. Lu, J., Ling, T.W., Chan, C.-Y., Chen, T.: From Region Encoding to Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In: Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.-Å., Ooi, B.C. (eds.) VLDB 2005: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 193–204. ACM, New York (2005)
12. Pal, S., Cseri, I., Seeliger, O., Rys, M., Schaller, G., Yu, W., Tomic, D., Baras, A., Berg, B., Churin, D., Kogan, E.: XQuery Implementation in a Relational Database System. In: Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.-Å., Ooi, B.C. (eds.) VLDB 2005: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 1175–1186. ACM, New York (2005)
13. Papparizos, S., Wu, Y., Lakshmanan, L.V.S., Jagadish, H.V.: Tree Logical Classes for Efficient Evaluation of XQuery. In: SIGMOD 2004: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 71–82. ACM, New York (2004)
14. XQuery 1.0 and XPath 2.0 Formal Semantics, W3C (2007)
15. XML Query Test Suite, W3C (2007)

Predictive Join Processing between Regions and Moving Objects*

Antonio Corral¹, Manuel Torres¹, Michael Vassilakopoulos²,
and Yannis Manolopoulos³

¹ Dept. of Languages and Computing, University of Almeria, 04120 Almeria, Spain
{acorral,mtorres}@ual.es

² Dept. of Informatics with Applications in Biomedicine,
University of Central Greece, Papasiopoulou 2-4, 35100, Lamia, Greece, and
Dept. of Informatics, Alexander TEI of Thessaloniki, 57400 Greece
mvasilako@ucg.gr

³ Dept. of Informatics, Aristotle University, 54124 Thessaloniki, Greece
manolopo@csd.auth.gr

Abstract. The family of R-trees is suitable for indexing various kinds of multidimensional objects. TPR*-trees are R-tree based structures that have been proposed for indexing a moving object database, e.g. a database of moving boats. Region Quadrees are suitable for indexing 2-dimensional regional data and their linear variant (Linear Region Quadrees) is used in many Geographical Information Systems (GIS) for this purpose, e.g. for the representation of stormy, or sunny regions. Although, both are tree structures, the organization of data space, the types of spatial data stored and the search algorithms applied on them are different in R-trees and Region Quadrees. In this paper, we examine a spatio-temporal problem that appears in many practical applications: processing of predictive joins between moving objects and regions (e.g. discovering the boats that will enter a storm), using these two families of data structures as storage and indexing mechanisms, and taking into account their similarities and differences. With a thorough experimental study, we show that the use of a synchronous Depth-First traversal order has the best performance balance (on average), taking into account the I/O activity and response time as performance measurements.

Keywords: Moving objects, TPR-trees, R-trees, linear quad-trees, query processing, joins.

1 Introduction

The recent advances of technologies in mobile communications and global positioning systems have increased users attention to an effective management of

* Supported by the Almacenes de Datos Espacio-Temporales basados en Ontologias project (TIN2005-09098-C05-03), funded by the Spanish Ministry of Science and Technology.

information on the objects that move in 2-dimensional space. Those moving objects send their current positions, which can be forwarded periodically to the users for different purposes (e.g. making decision, etc.). This position information is spatio-temporal, since spatial locations of objects change with time. A database that stores information for a large number of objects locations changing with time is called a *moving object database*.

Users queries issued on moving object databases can be categorized into two types: *past-time queries* and *future-time queries* [17]. The past-time query retrieves the history of dynamic objects movements in the past, while the future-time query predicts movements of dynamic objects in the future [17]. In this paper, we discuss join processing of future-time queries between regions and moving objects.

Spatial data are collected and stored in two main generic formats, called *vector* and *raster*. The basic unit of spatial data in the *vector* format corresponds to discrete real world features represented by points, lines or polygons. In the *raster* alternative, the basic unit of spatial data takes the form of a square grid cell, embedded within a grid of equally sized *pixels* (picture elements). On the other hand, the spatial access methods can be classified according to the two following approaches. First, *data-driven spatial access methods* are organized by partitioning the set of spatial objects, and the partitioning adapts to the distribution of the objects in the embedding space. An example of this approach is the R-tree. Second, *space-driven spatial access methods* are based on partitioning of the embedding two-dimensional space into cells of specific shapes and sizes, independently of the distribution of the spatial objects (objects are mapped to the cells according to some geometric criterion). An example of this approach is the Quadtree. The books [16] and [12] provide excellent information sources for the interested reader about Quadtrees and R-trees, respectively.

There are a number of variations of the R-tree all of which organize multi-dimensional data objects by making use of the Minimum Bounding Rectangles (MBRs) of the objects. We will concentrate on access methods that have the capability of dealing with anticipated future-time queries of moving objects or points (dynamic point of view). Generally, to support the future-time queries, databases store the current positions and velocities of moving objects as linear functions. Up to now, for processing current and future-time queries, several indexing methods have been proposed belonging to the R-tree family and the TPR*-tree [19] is the most widely-used index structure for predicting the future positions of moving points, which can be used for future-time queries.

The Region Quadtree is a space-driven spatial access method, which is suitable of storing and manipulating 2-dimensional regional data (or binary images). Moreover, many algorithms have been developed based on Quadtrees. The most widely known secondary memory alternative of this structure is the Linear Region Quadtree [16]. Linear Quadtrees have been used for organizing regional data in GIS [16].

The contributions of this paper consist in the following:

1. We present predictive join processing techniques between two different access methods, TPR*-trees for moving objects (vector data) and Linear Region Quadtrees for regions (raster data), in order to answer future-time (predictive) queries appearing in practical applications, like “Retrieve all the boats covering by a storm within 1 hour”. To the best of our knowledge, this is the first study of spatio-temporal joins between different data formats (vector and raster).
2. We distinguish between two types of such future-time queries, depending on the required result: future-time-interval and future-time-parameterized join queries, between vector and raster data.
3. We present a detailed experimental comparison of the alternative methods and highlight the performance winner, for each experimental setting.

The paper is organized as follows. In Section 2, we review the related literature and motivate the research reported here. In Section 3, a brief description of the TPR*-tree and the Linear Region Quadtree are presented. In Section 4, we present the algorithms that perform the predictive join processing between regions and moving objects. In Section 5, a comparative performance study of proposed algorithms is reported. Finally, in Section 6, conclusions on the contribution of this paper and future work are summarized.

2 Related Work and Motivation

In general, the spatial join combines two sets of spatial objects based on a spatial predicate (usually overlap). Recently, an exhaustive analysis of several techniques used to perform a spatial join taking into account a filter-and-refinement approach has been published in [7]. Regarding spatial joins over static spatial objects, we can classify the spatial join methods in three categories, depending on whether the sets of spatial objects involved in the query are indexed or not. When both sets are indexed, the most influential and known algorithm for joining two datasets indexed by R*-trees was presented in [2], where additionally several techniques to improve both CPU and I/O time have been studied. This algorithm follows a Depth-First synchronized tree traversal order. A breadth-first synchronized tree traversal version to reduce I/O cost was presented in [4]. In the case of just one set being indexed, several spatial join algorithms have been proposed in the literature, and the most representative ones are [9,13,11]. In the last category, when both sets are not indexed, the most relevant publications are [14,10,6]. We must highlight that, when both sets are indexed, but with incompatible type of indexes, such as by R-trees (hierarchical and non-disjoint indexing for vector data) and by Linear Region Quadtrees based on B⁺-trees (hierarchical and disjoint indexing for raster data), the only research work that proposes join algorithms between the different data formats is [3]. The authors proposed several algorithms to perform this spatial join, and the most novel uses a complex buffering system and the FD-order [16] to reduce the I/O cost, while searching in the B⁺-tree for the FD-code that overlaps with a point in the R-tree.

From the dynamic point of view, the most representative joins on moving objects have been proposed very recently. In [18], the authors present a set of spatio-temporal queries so-called time-parameterized queries, including the time-parameterized join query, which we will adapt later to our problem setting. In [5], query maintenance algorithms for spatial joins on continuously moving points that support updates were presented. And finally, in [20], the problem of processing continuous intersection join over moving objects, using TPR*-trees, has been addressed.

In practical applications, the need for spatio-temporal predictive joining between different data formats is common. For example, consider Figure 1, where five boats (shapes A-E), along with their moving vectors (arrows) and a storm (gray region) are depicted. At the time instant of Figure 1a (time I = now), only boat B is in the storm. At the time instant of Figure 1b (time II = now+10 minutes), boat B has just exited the storm, while boats C and D have just entered the storm. One possible query is: give me all the boats that will be under the storm for the next 9 minutes and 59 seconds (this is an example of a future-time-interval join and the result is: boat B). Note that we assumed that the resolution of time is one second. Another possible query is: give me all the boats that are under the storm now, the time point when this situation will change and the event that will cause the change of the situation (this is an example of a future-time-parameterized join and the result is: boat B is under the storm now, the situation will change in 10 minutes, because boat B will exit and boats C and D will enter the storm). It should be noted that both queries are predictive, since they refer to the future and they are based on the assumption that the moving vectors of a boats do not change (at least significantly) between subsequent updates of the position of a moving object. However, depending on the application, the frequency of updates of objects positions and the possibility of sudden changes of the movement vectors, the result such queries may be enough accurate. Moreover, it should be noted that the storm data are considered static, at least for time periods quite large in comparison to the update frequency of the objects positions.

Although, the queries described above arise naturally in practical applications, the literature (to the best of our knowledge) does not include any techniques for processing them, perhaps due to the different nature of regional and vector data and the different methods used for storing and indexing them. In this paper, we consider that the regional data (e.g. the storm) are stored and indexed using Linear Region Quadtrees (a common choice in GIS systems). The codes of the quad blocks are stored either in B⁺-trees [16], or in R*-trees (Oracle, in general, recommends using R-trees over quadtrees [8]), for comparison purposes between the two different alternatives. At time periods large enough for significant changes of the regional data, the whole storage and indexing structure is rebuilt. In this paper, we study the situation within one such time period, during which the regional data are considered static. Moreover, we consider that the changing vector data (e.g. moving boats) are stored and indexed using TPR*-trees [19] (the most widely-used index structure for predicting the future positions of

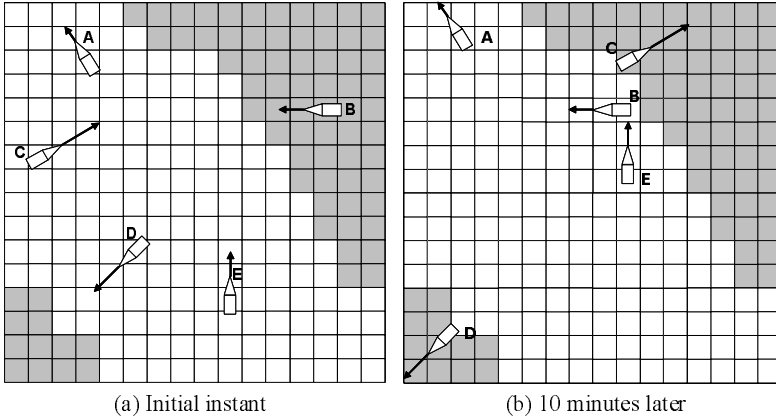


Fig. 1. Five moving boats and a storm at a time instant (a) and 10 minutes later (b)

moving points). Thus, we present and study the first algorithms for processing future-time-interval and future-time-parameterized join queries, between vector and raster data.

3 The Two Access Methods

3.1 The TPR*-Tree

We assume that the reader is already familiar with the R*-tree [1]. The TPR-tree [15] extends the R*-tree, predicts the future locations of moving objects by storing the location and the velocity of each object at a given time point. The locations of moving objects are indexed using CBRs (Conservative Bounding Rectangles) instead of MBRs (Minimum Bounding Rectangles). A CBR is composed of an MBR, representing the region that covers a set of moving objects at a specific time point, and the maximum and minimum moving velocities of the objects within an MBR at each axis (velocity bounding rectangle, VBR). The location of a moving object at any future-time point can be easily predicted with the location and moving velocity stored in a CBR. The predicted region of a node computed by using a current location of an MBR and its maximum and minimum velocities at each axis is defined as a bounding rectangle.

According to [19], a moving object o is represented with (1) an MBR o_R that denotes its extent at reference time 0, and (2) a velocity bounding rectangle (VBR) $o_V = \{o_{V1-}, o_{V1+}, o_{V2-}, o_{V2+}\}$ where o_{Vi-} (o_{Vi+}) describes the velocity of the lower (upper) boundary of o_R along the i -th dimension ($1 \leq i \leq 2$). Figure 2a shows the MBRs and VBRs of 4 objects a, b, c, d . The arrows (numbers) denote the directions (values) of their velocities. For example, the VBR of c is $c_V = \{-2, 0, 0, 2\}$, where the first two numbers are for the X-axis. A non-leaf entry is also represented with an MBR and a VBR. Specifically, the

MBR (VBR) tightly bounds the MBRs (VBRs) of the entries in its child node. In Figure 2a, the objects are clustered into two leaf nodes $N1$ and $N2$, which VBRs are $N1_V = \{-2, 1, -2, 1\}$ and $N2_V = \{-2, 0, -1, 2\}$. Figure 2b shows the MBRs at timestamp 1 (notice that each edge moves according to its velocity). The MBR of a non-leaf entry always encloses those of the objects in its subtree, but it is not necessarily tight.

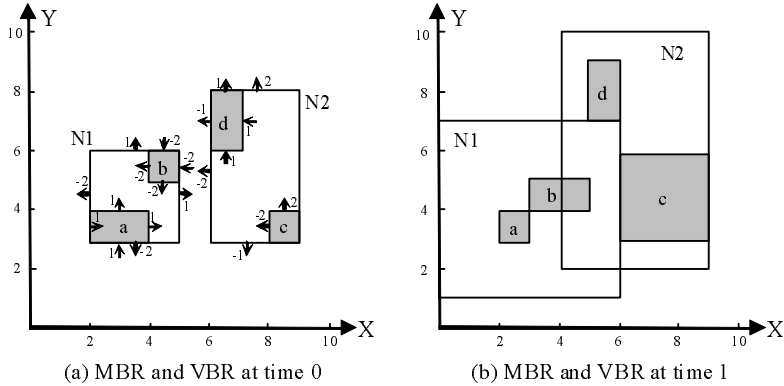


Fig. 2. Entry representation in a TPR*-tree

The TPR*-tree [19] uses a set of improved algorithms to build the TPR-tree and achieves an almost optimal tree. In general, the TPR*-tree basically uses the same structure as the TPR-tree. During the update operations, however, the TPR-tree employs the insertion and the deletion algorithms of the R*-tree as they are, while the TPR*-tree employs modified versions that reflect objects mobility. This makes it possible to improve the performance of updates and retrievals in the TPR*-tree over the TPR-tree. Since the TPR-tree considers the area, circumference, overlapping, and distance of an MBR only at the time of updates of moving objects, it cannot reflect the property that objects move with time. On the other hand, the TPR*-tree performs updates in such a way that it minimizes the area of a sweeping region, which is an extension of the rectangle that corresponds to a node with time after the updates of moving objects.

Taking into account the insertion strategy, the TPR-tree inserts a moving object into such a node whose MBR extension required is minimum at the time of the insertion. On the other hand, the TPR*-tree inserts a moving object into such a node with a minimum extension of the bounding rectangle after the insertion. Traversing from the root to lower-level nodes, their rectangle extensions required for the insertion are computed, and also are stored into a priority queue. And finally, the optimal node for the insertion is the one having the smallest value. With this strategy, the TPR*-tree requires a cost higher than the TPR-tree for updates. However, its compactness of bounding rectangle, it greatly improves the overall query performance.

3.2 Region Quadrees

The Region Quadtree is the most popular member in the family of quadtree-based access methods. It is used for the representation of binary images, that is $2^n \times 2^n$ binary arrays (for a positive integer n), where a 1 (0) entry stands for a black (white) picture element. More precisely, it is a degree four tree with height n , at most. Each node corresponds to a square array of pixels (the root corresponds to the whole image). If all of them have the same color (black or white) the node is a leaf of that color. Otherwise, the node is colored gray and has four children. Each of these children corresponds to one of the four square sub-arrays to which the array of that node is partitioned. We assume here, that the first (leftmost) child corresponds to the NW sub-array, the second to the NE sub-array, the third to the SW sub-array and the fourth (rightmost) child to the SE sub-array. For more details regarding Quadtrees see [16].

Region Quadtrees, as presented above, can be implemented as main memory tree structures. Variations of Region Quadtrees have been developed for secondary memory. Linear Region Quadtrees are the ones used most extensively. A Linear Quadtree representation consists of a list of values, where there is one value for each black node of the pointer-based Quadtree. The value a node is an address describing the position and size of the corresponding block in the image. These addresses can be stored in a efficient structure for secondary memory (such as an B-tree or one of its variations). There are also variations of this representations where white nodes are stored too, or variations which are suitable for multicolor images. Evidently, this representation is very space efficient, although it is not suited too many useful algorithms that are designed for pointer-based Quadtrees. The most popular linear implementations are the FL (Fixed Length), the FD (Fixed Length-Depth) and the VL (Variable Length) linear implementation. For more details regarding FL and VL implementations see [16].

In the rest of this paper, like in [3], we assume that Linear Quadtrees are presented with FD-codes stored in a B^+ -tree or in an R^* -tree. The choice of FD linear representation is not accidental, since it is made of base 4 digits and is thus easily handled using two bits for each digit. Besides, the sorted sequence of FD-codes is a Depth-First traversal of the tree. Since internal and white nodes are omitted, sibling black nodes are stored consecutively in the B^+ -tree or, in general, nodes that are close in space are likely to be stored in the same or consecutive B^+ -tree leaves. This property helps at reducing the I/O cost of join processing. Since in the same quadtree two black nodes that are ancestor and descendant cannot co-exist, two FD-codes that coincide at all the directional digits cannot exist neither. This means that the directional part of the FD-codes is sufficient for building B^+ -tree at all the levels. At the leaf-level, the depth of each black node should also be stored so that images are accurately represented [3]. Since the FD-codes can be transformed in disjoint MBRs we can store the sequence of FD-codes in an R^* -tree, and apply of existing query algorithms over this popular spatial index.

4 Future-Time Join Algorithms

Before join processing, we must create the indexes that store the static region (linear region quadtree stored in a B^+ -tree, or an R^* -tree) and moving points (TPR*-tree). We have decided to store the sequence of FD-codes in an R^* -tree, as an alternative to a B^+ -tree, because Oracle, in general, recommends using R -trees over quadtrees (due to higher tiling levels in the quadtree that cause very expensive preprocessing and storage costs) [8]. Moreover, the correspondence of the spaces covered by the two structures has been established in [3]. Finally, we have to take into account the two types of future-time join queries that we will study in this paper: future-time-interval join and future-time-parameterized join.

Joining the two structures can be carried out following two join processing techniques: (1) multiple queries and (2) synchronized tree traversal. *Multiple queries* technique performs a window query on the TPR*-tree for each FD-code indexed in the B^+ -tree, or in the R^* -tree. And the *synchronized tree traversal* technique follows a Depth-First or Breadth-First order to traverse both structures during the query processing. More specifically, we have designed and implemented the following five algorithms for future-time-interval join and one algorithm for future-time-parameterized join between regions and moving objects.

4.1 Future-Time-Interval Join

The future-time-interval join receives the time-interval of interest ($[T_{st}, T_{ed}]$) and returns the result, valid only during such as time-interval.

B^+ to TPR*-Tree Join (B-TPR). This algorithm follows the multiple queries technique and it descends the B^+ -tree from the root to its leftmost leaf. It accesses sequentially the FD-codes present in this leaf, and for each FD-code it performs a predictive window (MBR of FD-code and VBR, which is 0, since the region is static) query in the TPR*-tree (reporting intersections of this FD-code and elements in the TPR*-tree leaves within $[T_{st}, T_{ed}]$). By making use of the horizontal inter-leaf pointers, it accesses the next B^+ -tree leaf and repeats the previous step. Of course, the reverse alternative (from TPR* to B^+ , i.e. to scan the entries of the TPR*-tree and perform window queries in the B^+ -tree) can be easily implemented, and we expect that the results will be very similar.

R^* to TPR*-Tree Join (R-TPR). Assuming that we store the FD-codes in an R^* -tree, this algorithm follows the multiple queries technique as well, and it traverses recursively the R^* -tree, accessing the MBRs in each node in order of appearance within the node. For the MBR (FD-code) of each leaf accessed, it performs a predictive window (MBR and VBR (it is 0 since the region is static)) query in the TPR*-tree, reporting intersections of this MBR and elements in the TPR*-tree leaves within $[T_{st}, T_{ed}]$. Of course, the reverse alternative (from TPR* to R^* , i.e. to scan the entries of the TPR*-tree and perform window queries in

the R*-tree) can be easily implemented, and we expect that the results will be very similar.

Depth-First Traversal Join (R-TPR-DFJ). This algorithm follows the synchronized tree traversal technique, using a Depth-First order of both trees for overlap join [2]. It is based on the enclosure property of R-tree nodes: if the MBRs of two internal nodes do not overlap, then there can not be any MBRs below them that overlap. In this case, to apply this technique on TPR*-tree we need an additional function (*compute_intersection_period*) to check if two dynamic entries (MBR and VBR) overlap in the required time-interval.

R-TPR-DFJ(nodeR, nodeTPR, Tst, Ted)

```

If nodeR and nodeTPR are leaves
  For each pair of entries (Rentry, TPEntry)
    Save Rentry in RenTPEntry
    If compute_intersection_period(RenTPEntry, TPEntry, Tst, Ted)
      Add TPEntry to the result
Else
  For each pair of entries (Rentry, TPEntry)
    Save Rentry in RenTPEntry
    If compute_intersection_period(RenTPEntry, TPEntry, Tst, Ted)
      nodeRaux = ReadNodeR(Rentry.p)
      nodeTPRaux = ReadNodeTPR(TPEntry.p)
      R-TPR-DFJ(nodeRaux, nodeTPRaux, Tst, Ted)

```

An advanced variant of the algorithm applies a local optimization (because it improves the overlap computation with each node-pair join processing) in order to reduce CPU cost. In particular, when joining two nodes, the overlapping of entries is computed using a plane-sweep technique [2] instead of brute-force nested loop algorithm. In general, the MBRs of each node are sorted on the x-axis, and a merge-like algorithm is carried out, reducing significantly the number of intersection tests. We will call to this variant, *R-TPR-DFJ-PS*.

Breadth-First Traversal Join (R-TPR-BFJ). This algorithm follows the synchronized tree traversal technique, using a Breadth-First order of both trees [4]. The algorithm traverses down the two trees synchronously level by level. At each level, the algorithm creates an *intermediate join index* (IJI) and deploys global optimization techniques (e.g. ordering) to improve the join computation at the next level. It terminates when the IJI is created by joining the leaf entries in the R*-tree with the dynamic leaf entries in the TPR*-tree. Again, we need the function (*compute_intersection_period*) to check if two dynamic entries (MBR and VBR) overlap in the required time-interval. According to [4], we have implemented two orderings of intermediate index join as global optimization: ordering by the sum of the centers (OrdSum) and ordering by center point (OrdCen). In

this case, the MBR of the TPR*-tree is the bounding rectangle that covers the VBR in the required time-interval.

B-TPR FD-Buffer Join (B-TPR-FD). This algorithm follows a particular technique for join processing. It uses a complex buffering system and the FD-order in the B⁺-tree to reduce the I/O cost for join processing between an R*-tree (TPR*-tree) and a FD-linear quadtree stored in a B⁺-tree [3]. In general, the algorithm works as follows: (1) process each entry of the TPR*-tree root in FD-order; (2) read as many FD-codes as possible for the current entry and store them in the FD-buffer, (3) call recursively the join routine for this entry; (4) when the join routine returns, empty the FD-buffer and repeat the previous two steps until the current entry has been completely checked; (5) repeat for the next entry of the TPR*-tree root. On the other hand, The join routine for a TPR*-tree node and the required time-interval works as follows: (1) if the node is a leaf, check intersections at the required time-interval using the *compute_intersection_period* function and return; (2) If not (non-leaf node), for each child of the node that has not been examined in relation to the FD-codes in FD-buffer, call the join routine recursively.

4.2 Future-Time-Parameterized Join (TP-Join)

In general, this type of future-time join does not receive any parameter and it returns (1) the actual result at the time that the query (join) is emitted, (2) the expiry time of the result given in; and (3) the change that causes the invalidation of the result. That is, the answers are in format of triplets (R, T, C) [18].

A static spatial join returns all pairs of objects from two datasets that satisfy some spatial predicate (usually overlap). The join result changes in the future when: (1) a pair of objects, in the current result, ceases to satisfy the join condition, or (2) a pair not in the result starts to satisfy the condition. In general, we denote the *influence time* of a pair of objects (o_1, o_2) as $T_{INF}(o_1, o_2)$, and it the next timestamp that will change the result [18]. The influence time is 1, if a pair will never change the join result, and the expiry time is the minimum influence time. The influence time of two non-leaf entries, $T_{INF}(E_1, E_2)$, should be a lower bound of the $T_{INF}(o_1, o_2)$ of any two objects o_1 and o_2 in the subtrees of the non-leaf entries E_1 and E_2 , respectively.

In general, our join algorithm works as follows: it traverses, in Depth-First order, the two trees (R*-tree and TPR*-tree) simultaneously starting from the two roots. Suppose E_1 and E_2 to be two entries in non-leaf nodes, one from the R*-tree and the other from the TPR*-tree. The traversals go down the subtrees pointed by E_1 and E_2 if one of the following conditions holds: (1) the MBRs of E_1 and E_2 overlap, or (2) $T_{INF}(E_1, E_2)$ is less than or equal to the minimum influence time of all object pairs seen so far (in this case their subtrees may contain object pairs that cause the next result change). Condition (1) finds the current join pairs and condition (2) identifies the next timestamp. The traversals stop when leaf levels are reached for both trees. Notice that to compute T_{INF} we use the *compute_intersection_period* function that returns the time-interval in which the two entries overlap for a given time-interval.

TP-Join(nodeR, nodeTPR)

```

If nodeR and nodeTPR are leaves
  For each pair of entries (Rentry, TPEntry)
    If  $T_{\text{INF}}(\text{Rentry}, \text{TPEntry}) < T$ 
       $C = (\text{Rentry}, \text{TPEntry});$ 
       $T = T_{\text{INF}}(\text{Rentry}, \text{TPEntry});$ 
    Else if  $T_{\text{INF}}(\text{Rentry}, \text{TPEntry}) == T$ 
       $C = C \cup (\text{Rentry}, \text{TPEntry})$ 
    If Rentry overlaps TPEntry
       $R = R \cup (\text{Rentry}, \text{TPEntry})$ 
  Else
    For each pair of entries (Rentry, TPEntry)
      If  $(T_{\text{INF}}(\text{Rentry}, \text{TPEntry}) \leq T)$  or (Rentry overlaps TPEntry)
        nodeRaux = ReadNodeR(Rentry.p)
        nodeTPRaux = ReadNodeTPR(TPEntry.p)
        TP-Join(nodeRaux, nodeTPRaux)

```

Note that all the above algorithms are significantly different from existing R-tree based join algorithms. The special properties of TPR*-trees for query processing have been utilized, as well as, the function *compute_intersection_period()* has been used. Besides, previous algorithms that combine FD-Linear-Quadtrees stored in a B⁺-tree and R*-trees have been adapted for use with TPR*-trees (not a trivial task).

5 Experimental Results

In this section, we have evaluated the performance of our predictive join algorithms over regional data (black-white images of $2^{11} \times 2^{11}$ pixels) and moving points using synthetic (uniform distribution) and real data (24493 populated places of north-america). The regional data correspond to visible spectrums of areas of California (Sequoia data). Notice that, since $n = 11$, an FD-code for such an image requires $2 \times 11 + \lceil \log_2(11 + 1) \rceil = 26$ bits. For the moving objects, each object is associated with a VBR such that on each dimension, the velocity value distribution is uniform in the range $[0,5]$. All experiments were performed on an Intel/Linux workstation with a Pentium IV 2.5 GHz processor, 1 GByte of main memory, and several GBytes of secondary storage, using the gcc compiler. The node size for the tree structures (B⁺-tree, R*-tree and TPR*-tree) is 1 KByte, according to [19]. The performance measurements are: (1) the number of page accesses and (2) the response time (elapsed time) reported in seconds.

Our first experiment seeks the most appropriate LRU buffer size (nodes) for our predictive join algorithms that will be used in the next experiments. We have considered the following configuration: the number of moving objects (synthetic-uniform) is 10000, the query time-interval is $[0, 5]$, and the LRU buffer size is

variable (8, 32, 64, 128 and 512 Kbytes, or nodes). In Figure 3, the results of 6 algorithms are shown. We have to highlight that R-TPR-DFJ-PS was very similar to R-TPR-DFJ; and R-TPR-BFJ1 (OrdSum) obtained very similar results to R-TPR-BFJ2 (OrdCen), for this reason we only show the results of R-TPR-DFJ-PS (R-TPR-DFJ enhanced with the *plane-sweep technique*) and R-TPR-BFJ1. B-TPR and R-TPR are the most I/O-consuming when the buffer sizes are small, but when they are large enough (≥ 32); these algorithms obtain the best behavior (the best is B-TPR, 3112 node accesses for 512 nodes in the LRU buffer). The reason of this excellent I/O behavior is due to the good spatial locality of the TPR*-tree, there is a high probability that in the next predictive window query over the TPR*-tree, a great part of this index remains in the LRU buffer. On the other hand, these algorithms are the most time-consuming, due to the join processing technique, i.e. multiple queries. R-TPR-DFJ-PS shows an excellent behavior with respect to the I/O cost and response time, mainly due to the use of synchronized tree traversal as a join processing technique. R-TPR-BFJ1 does not improve the previous algorithms, due to the high cost of managing the global IJI, although for big LRU buffer size the I/O activity is acceptable. R-TPR-FD is an algorithm designed for reducing the number of node accesses, and for this reason it gets good behavior for this performance measurement, but it consumes a lot of time to return the final result, due to the continuous searches the FD-codes in the B⁺-tree and the management of the FD-buffer. Finally, the TP-joint query gets a similar behavior to the R-TPR-DFJ-PS for I/O activity, but the response time is slightly larger, since this algorithm has to report three answers (R, T, C) and the influence time [18].

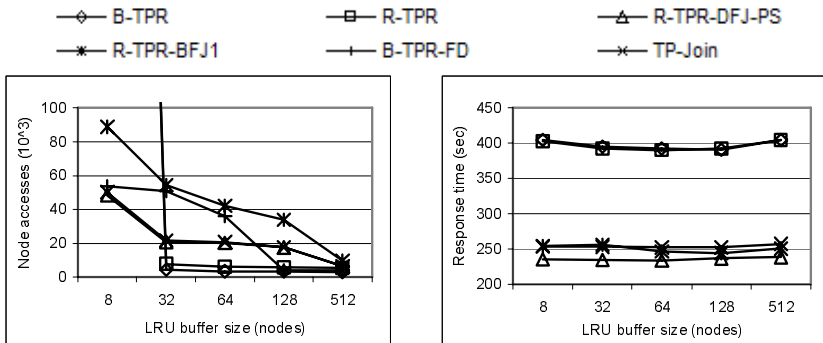


Fig. 3. Performance comparison with respect to the LRU buffer sizes

In the second experiment, we have studied the behavior of the predictive join algorithms when the cardinality of the moving objects datasets varies. We have the following configuration: LRU buffer size is 128 nodes, the query time-interval $[0, 5]$, and the cardinality of the datasets is variable (1000, 10000, 50000 and 100000). Figure 4 shows B-TPR and R-TPR get the best results for this LRU buffer size, although they are time-consuming. B-TPR-FD obtains also good

behavior for I/O activity until the number of moving points is 100000. R-TPR-BFJ1 needs many node accesses, although it needs an acceptable response time. R-TPR-DFJ-PS is the fastest, although it needs more disk accesses than the join algorithms that use multiple queries as the technique for the join processing. TP-Join has similar behavior than R-TPR-DFJ-PS, since they follow the same join processing technique, although the response time is slightly larger.

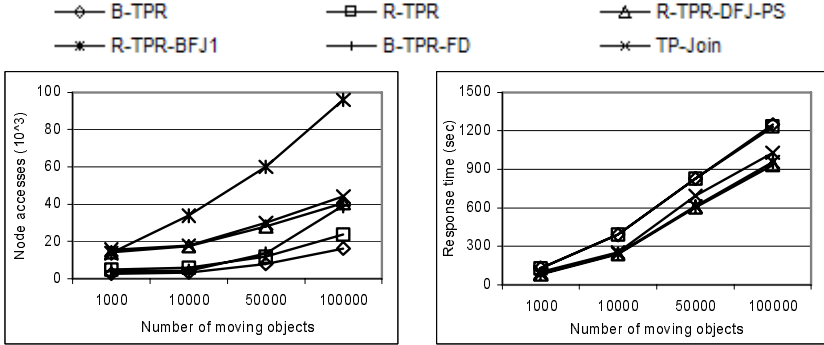


Fig. 4. Performance comparison with respect to the moving objects datasets sizes

In the third experiment, we have compared the behavior of the predictive join algorithms, varying the query time-interval. We have the following configuration: LRU buffer size is 128 nodes, the cardinality of the moving objects dataset is 10000, and the query time-intervals are $[0, 0]$, $[0, 5]$, $[0, 10]$ and $[0, 20]$. Since TP-Join does not receive any query time-interval, the result is not reported. We have to highlight that when the time-interval enlarges, the moving objects cover more space along their movement and the MBRs that cover them grow as well. This fact generates more overlaps between MBRs, and it increments the number of operations necessary to solve the join.

Figure 5 shows again that B-TPR and R-TPR get the best results for the number of node accesses (B-TPR needs less node accesses than R-TRP to perform the same query), although they are time-consuming. Moreover, B-TPR-FD gets interesting results for small query time-intervals, but for larger ones it needs more node accesses. The best performance balance corresponds to R-TPR-DFJ-PS, which consumes a reasonable quantity of node accesses, but it is the fastest for small query time-interval ($[0, 0]$ and $[0, 5]$), but for $[0, 10]$ and $[0, 20]$ sizes it is slightly slower than R-TPR-BFJ1. The join algorithm which uses a Breadth-First traversal order of both trees has a surprising behavior; it is I/O-consuming, but it is the fastest for large query time-intervals. It is mainly due to the applied query processing technique, since it only considers the overlapped elements level by level (there is no backtracking) and when the leaf nodes are reached, the result is reported. Of course, it also needs additional main memory resources to store the IJI.

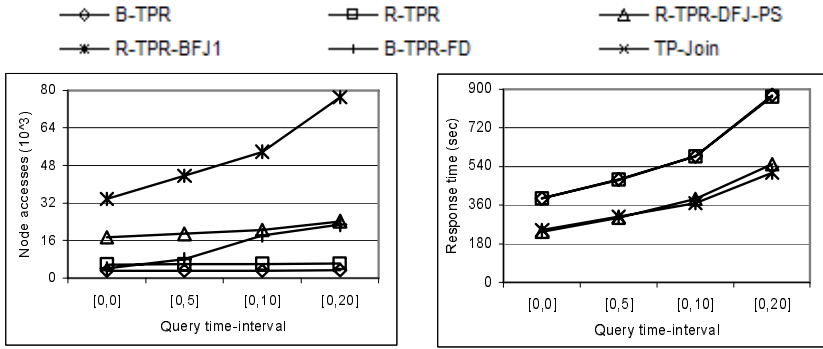


Fig. 5. Performance comparison with respect to the query time-interval sizes

The last results reported here are a summary (representative set) of experiments with real moving object datasets. In general, the tendencies are very similar to the synthetic uniform data. For example, the left chart of Figure 6 is very similar to the left chart of Figure 3, except for the LRU buffer size for B-TPR and R-TPR, starting from which they become the best (≥ 128). Moreover, observe that in the right charts of Figures 5 and 6, the trends are quite similar, where R-TPR-DFJ-PS is the fastest for small query time-interval and for larger ones R-TPR-BFJ1 consumes slightly less time to report the final result; and B-TPR and R-TPR are the most expensive alternatives for response time consumed (R-TPR is slightly faster than B-TPR for the same query).

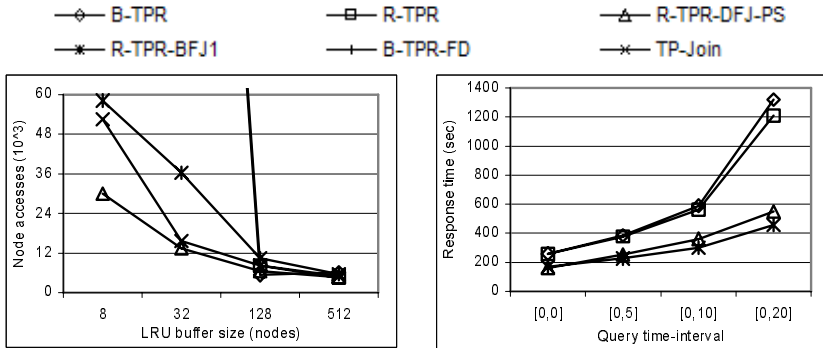


Fig. 6. Performance for real data, for LRU buffer and query time-interval sizes

From the previous performance comparison (for synthetic and real data), the most important conclusions are the following: (1) B-TPR and R-TPR are appropriate when we have available enough resources for buffering. (2) The predictive join algorithms which use a Breadth-First traversal order of both trees (R-TPR-BFJ) have a good behavior for large query time-interval and buffer

sizes, obtaining the best response time. (3) B-TPR-FD reports interesting results with respect to the I/O activity, but it is time-consuming due to the high computational cost of managing the FD-buffer. (4) Finally, the predictive join algorithms which use a synchronous Depth-First traversal order (R-TPR-DFJ-PS and TP-join) have the best performance balance (on average) in all executed experiments, taking into account the I/O activity and response time.

6 Conclusions and Future Work

The contribution of this paper falls within in the study of a spatio-temporal problem that appears in real-world applications: processing of predictive joins between moving objects and regions. To the best of our knowledge, this is the first study of its kind. For this purpose: (1) We have considered two types of future-time join queries: future-time-interval join and future-time-parameterized join. To solve these queries, we have used the TPR*-tree, which is the most widely-used index structure for predicting the future positions of moving points, and the Linear Region Quadtree (FD Linear Quadtree, as pointerless representation) stored in a B^+ -tree, or in R^* -tree. (2) We have proposed several join algorithms between these two indexes, following two join processing techniques: multiple queries and synchronized tree traversal, to solve such future-time join queries. (3) By extensive experimental results, we have shown that the use of a synchronous Depth-First traversal order (R-TPR-DFJ-PS and TP-join) has the best performance balance (on average), considering the I/O activity and response time.

Future research may include the extension of our algorithms to perform continuous intersection joins [20], and the use of moving and/or changing, instead of static, regions.

References

1. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R^* -tree: an Efficient and Robust Access Method for Points and Rectangles. In: SIGMOD Conference, pp. 322–331 (1990)
2. Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient Processing of Spatial Joins Using R -trees. In: SIGMOD Conference, pp. 237–246 (1993)
3. Corral, A., Vassilakopoulos, M., Manolopoulos, Y.: Algorithms for Joining R -Trees and Linear Region Quadtrees. In: Güting, R.H., Papadias, D., Lochovsky, F.H. (eds.) SSD 1999. LNCS, vol. 1651, pp. 251–269. Springer, Heidelberg (1999)
4. Huang, Y.M., Jing, N., Rundensteiner, E.: Spatial Joins Using R -trees: Breadth-First Traversal with Global Optimizations. In: VLDB Conference, pp. 395–405 (1997)
5. Iwerks, G.S., Samet, H., Smith, K.P.: Maintenance of K -nn and Spatial Join Queries on Continuously Moving Points. *TODS* 31(2), 485–536 (2006)
6. Jacox, E.H., Samet, H.: Iterative Spatial Join. *TODS* 28(3), 230–256 (2003)
7. Jacox, E.H., Samet, H.: Spatial Join Techniques. *TODS*, article 7, 32(1), 1–44 (2007)

8. Kothuri, R.K., Ravada, S., Abugov, D.: Quadtree and R-tree Indexes in Oracle Spatial: A Comparison using GIS Data. In: SIGMOD Conference, pp. 546–557 (2002)
9. Lo, M.L., Ravishankar, C.V.: Spatial Joins Using Seeded Trees. In: SIGMOD Conference, pp. 209–220 (1994)
10. Lo, M.L., Ravishankar, C.V.: Spatial Hash-Joins. In: SIGMOD Conference, pp. 247–258 (1996)
11. Mamoulis, N., Papadias, D.: Slot Index Spatial Join. *TKDE* 15(1), 211–231 (2003)
12. Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., Theodoridis, Y.: *R-Trees: Theory and Applications*. Springer, Heidelberg (2006)
13. Papadopoulos, A.N., Rigaux, P., Scholl, M.: A Performance Evaluation of Spatial Join Processing Strategies. In: Güting, R.H., Papadias, D., Lochovsky, F.H. (eds.) *SSD 1999*. LNCS, vol. 1651, pp. 286–307. Springer, Heidelberg (1999)
14. Patel, J.M., Dewitt, D.J.: Partition Based Spatial-Merge Join. In: SIGMOD Conference, pp. 259–270 (1996)
15. Saltinis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the Positions of Continuously Moving Objects. In: SIGMOD Conference, pp. 331–342 (2000)
16. Samet, H.: *Applications of Spatial Data Structures*. Addison-Wesley, Reading (1990)
17. Sistla, A.P., Wolfson, O., Chamberlain, S., Dao, S.: Modeling and Querying Moving Objects. In: ICDE Conference, pp. 422–432 (1997)
18. Tao, Y., Papadias, D.: Time-Parameterized Queries in Spatio-Temporal Databases. In: SIGMOD Conference, pp. 334–345 (2002)
19. Tao, Y., Papadias, D., Sun, J.: The TPR*-tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In: VLDB Conference, pp. 790–801 (2003)
20. Zhang, R., Lin, D., Ramamohanarao, K., Bertino, E.: Continuous Intersection Joins Over Moving Objects. In: ICDE Conference, pp. 863–872 (2008)

Multiple-Objective Compression of Data Cubes in Cooperative OLAP Environments

Alfredo Cuzzocrea

ICAR Inst. and DEIS Dept.

University of Calabria

I-87036 Cosenza, Italy

cuzzocrea@si.deis.unical.it

<http://si.deis.unical.it/~cuzzocrea>

Abstract. Traditional data cube compression techniques do not consider the yet relevant problem of compressing data cubes in the presence of multiple objectives rather than only one (e.g., a given space bound). Starting from next-generation OLAP scenarios where this problem makes sense, such as those drawn by cooperative OLAP environments, in this paper we fulfill this lack via (i) introducing and rigorously formalizing the novel multiple-objective data cube compression paradigm, and (ii) providing an effective solution to this problem by means of a greedy algorithm able to find a sub-optimal solution and, as a consequence, an “intermediate” data cube compressed representation that accommodates a large family of even different OLAP queries, which model the multiple requirements in our proposed framework. Finally, we complete our analytical contribution with a wide experimental analysis on benchmark data cubes.

1 Introduction

The problem of compressing data cubes has a long history in literature (e.g., [30,31]), and it is currently of relevant interest for the Database and Data Warehousing research communities. Basically, this problem deals with the issue of compressing massive-in-size data cubes that make access and query evaluation costs prohibitive. Given an input space bound, the widely-accepted solution to this problem consists in generating a compressed representation of the target data cube, with the goal of reducing the overall data cube size to that bound while admitting loss of information and approximation that are considered irrelevant for OLAP analysis goals (e.g., see [9]). The derived benefits can be summarized in a significant reduction of computational overheads needed to both represent the data cube and evaluate resource-intensive OLAP queries, which constitute a wide query class (e.g., *range-* [29], *top-k* [59], and *iceberg* [21] queries) allowing us to extract useful knowledge from huge amounts of multidimensional data repositories in the vest of summarized information (e.g., *aggregate information*), otherwise infeasible by means of traditional OLTP approaches.

This evidence has given raise to the proliferation of a number of *Approximate Query Answering* (AQA) techniques, which aim at providing *approximate answers* to resource-intensive OLAP queries instead of computing exact answers, as decimal

precision is usually negligible during OLAP query and report activities (e.g., see [9]). AQA techniques have been intensively investigated during the last fifteen years, also achieving important results on both the theoretical and practical planes. Summarizing, AQA techniques propose (i) computing compressed representations of data cubes, and (ii) evaluating (approximate) answers against such representations via ad-hoc query algorithms that, usually, meaningfully take advantages from the *hierarchical nature* of the compressed representation. The latter is, in turn, inherited from the one of the input data cube, modeled in terms of a *lattice of cuboids* [27]. Among all, *histograms*, *wavelets*, and *sampling-based approaches* are the most successful techniques, and they have also inducted several applications in contexts even different than OLAP, like P2P data management (e.g., [25] Sect. 4 discusses some state-of-the-art histogram-based compression techniques, which are close to our work, whereas we remind the reader to the active literature for what regards wavelet- and sampling-based data cube compression techniques, being the latter are outside the scope of this paper).

The idea of introducing multiple-objective computational paradigms in order to deal with complex Database and Data Warehousing research challenges has been considered in few contexts previously, and mostly with respect to requirements defined by *multiple queries* (i.e., simultaneous queries belonging to different query classes – e.g., [52] Among these initiatives, we recall: (i) multiple-query optimization for the view selection and materialization problem in Data Warehousing and OLAP systems [40,53], (ii) multiple-query based data sources integration [33,18,43], (iii) multi-objective query processing in database systems [50,3] and OLAP [36], (iv) multi-objective query processing for specialized contexts such as data aggregation [20] complex mining tasks [34,35] and data stream processing [4,58], and, more recently, (v) skyline query processing [2,4,45], which aims at extracting Pareto distributions from relational data tables according to *multiple preferences*. We review some of these initiatives in Sect. 2. Contrary to the above-listed research contributions, to the best of our knowledge, data cube approximation literature has devoted very little attention to the issue of performing the compression process on the basis of multiple objectives. While this evidence clearly causes the lack of a theoretical background for the problem investigated in this paper, at the same time it gives more relevance to the innovation carried out by our research work, and puts the basis for further research efforts in this field.

Paper Outline. The remaining part of this paper is organized as follows. In Sect. 2, we formalize the statement of the research problem we investigate in this paper. In Sect. 3, we discuss the novel OLAP computational paradigm introduced in this paper, which we name as *multiple-objective data cube compression*. Sect. 4 focuses on related work via considering two different scientific areas that both influence our work: (i) histogram-based data cube compression techniques, and (ii) multi-objective query processing in Database and Data Warehousing systems. Sect. 5 describes our overall multi-objective compression technique for data cubes in cooperative OLAP environments. In Sect. 6, we provide a wide experimental analysis of the technique we propose against two-dimensional benchmark data cubes, and in comparison with state-of-the-art data cube compression approaches. The results of this analysis clearly confirm the validity and the efficiency of our proposal. Finally, in Sect. 7 we derive conclusions of our work and define further research directions in this field.

2 Problem Statement

Formally, a data cube \mathcal{L} is a tuple $\mathcal{L} = \langle C, J, \mathcal{H}, \mathcal{M} \rangle$, such that: (i) C is the *data domain* of \mathcal{L} containing (OLAP) *data cells*, which are the basic aggregate values of \mathcal{L} computed on top of tuples extracted from the external data source S populating \mathcal{L} via SQL aggregation operators such as COUNT and SUM; (ii) J is the set of *dimensions* of \mathcal{L} , i.e. the *functional attributes* (of S) with respect to which tuples in S are aggregated; (iii) \mathcal{H} is the set of *hierarchies* defined on the dimensions of \mathcal{L} , i.e. hierarchical representations of the functional attributes shaped-in-the-form-of general trees; (iv) \mathcal{M} is the set of *measures* of \mathcal{L} , i.e. the *attributes of interest* (of S) whose aggregate values are stored in data cells of \mathcal{L} .

Until now, researchers have mainly been interested in devising efficient solutions for compressing the data domain of the input data cube (i.e., C) by deliberately neglecting the information/knowledge modeled by *logical schemas* (e.g., [57]) and OLAP hierarchies of the data cube. This is a serious limitation since every compression method for databases/data-warehouses/data-cubes should instead carefully consider the *semantics* given by models and schemas that, in the particular OLAP context, represent an effective *plus-value* for the compression of “complex” data cubes, i.e. data cubes defined on top of complex data/knowledge bases. Recent approaches following this direction are [38,13,15].

Apart from matters like the above, more problematic issues appear when the data cube compression process must be accomplished in a *cooperative OLAP environment*, where *multiple* OLAP client applications interact with an OLAP server and cooperate to the achievement of a *common* set of business processes (e.g., like in *Supply Chain Management Systems* (SCMS)). In doing this, client applications access and query the OLAP server according to their *own* specific *aggregation hierarchies*, called *Customized Query Aggregation Hierarchies* (CQAH) and denoted by \mathcal{H}_{Q_d} , to be used during query evaluation, one for each dimension (of interest) of the target data cube. On the practical plane, this means that OLAP queries posed by client applications are defined *along groups* of CQAH, and overall define a *very heterogeneous query class depending on specific knowledge creation, management and delivery processes implemented by different client applications*. The above-described one is a novel application scenario not considered by previous data cube compression approaches, and, as a consequence, it poses novel, previously-unrecognized research challenges. The underlying problem can be formulated as follows.

Definition 1 – The Multiple-Objective Data Cube Compression Problem in Cooperative OLAP Environments. Given (i) a n -dimensional data cube \mathcal{L} , (ii) p OLAP client applications accessing \mathcal{L} to perform cooperative business intelligence processes, (iii) the space bound \mathcal{B} available for housing the compressed representation of \mathcal{L} , denoted by $\tilde{\mathcal{L}}$, and, for each dimension d of \mathcal{L} , (iv) the set of p CQAH, denoted by $\mathcal{W}_d = \{\mathcal{H}_{Q_{d,0}}, \mathcal{H}_{Q_{d,1}}, \dots, \mathcal{H}_{Q_{d,p-1}}\}$ ¹, each one defined by an OLAP client application for

¹ For the sake of simplicity, assume that CQAH in \mathcal{W}_d have all the same depth.

that dimension, the multiple-objective data cube compression process consists in finding an intermediate compressed representation of \mathcal{L} , $\tilde{\mathcal{L}}$, such that the average query error experimented by OLAP client applications is minimized in most cases, based on a best effort approach.

3 Multiple-Objective Data Cube Compression: A Novel OLAP Computational Paradigm

In the application scenario we investigate, for each dimension of the target data cube, an OLAP client application builds its particular CQAH by means of iteratively grouping *leaf members* of the original hierarchy of the target data cube in a *bottom-up* manner, on the basis of specific application goals. As possible alternatives to this model, a client application could build its CQAH by means of a given ad-hoc procedure (e.g., business-process-driven), or as the result of *Data Mining* (DM) activities, such as *classification* and *clustering*, implemented on top of advanced DM tools running on the original hierarchies of the target data cube. It should be noted that this aspect makes our approach *orthogonal* with respect to any method used to generate CQAH, and, consequentially, suitable to be easily integrated with even complex intelligent techniques for handling OLAP hierarchies (e.g., [38,13,15]).

The above-described one can be reasonably intended as an *application-driven process*, meaning that client applications build their proper CQAH according to their *interest* for specific *portions* of the data cube that, in turn, are driven by specific decision-support-oriented subjects of analysis [27]. We highlight that this analysis model can occur frequently in traditional OLAP environments as well as in cooperative OLAP environments, since decision-support-tasks typically adhere to a departmental, subject-oriented strategy [27], and implicitly define a *partitioned organization* of the whole data domain of the target data cube. On the other hand, it should be noted that any other *arbitrary* process used to generate CQAH (e.g., a random approach that is not application-driven) could involve in gross errors. In fact, arbitrary processes could easily originate circumstances such that, in the final compressed representation of the data cube, ranges of multidimensional data for which the target OLAP client application has low interest could be represented with a great level of detail, and, contrary to this, *critical-for-the-application* ranges of multidimensional data (i.e., ranges for which the target OLAP client application has high interest) could be instead represented with a low level of detail, thus decreasing the accuracy of retrieved approximate answers and the effectiveness of the overall (cooperative) OLAP-based business process.

In a cooperative OLAP environment like the one investigated in this paper, being impossible to generate a *valid-for-all* data cube compression in the presence of multiple requirements (i.e., a data cube compression able to accommodate a large family of even different OLAP queries), the final compressed representation of the target data cube must be obtained as the result of an *intermediate representation* originated by meaningfully combining different CQAH defined by multiple client applications, i.e. different goals of multiple client applications. In fact, it is a matter of fact noticing that (i) aggregations given by the original hierarchies of the target data cube could not fit the goals of an even large sub-set of the available applications, and that (ii) this

phenomenon gets worse when the number of client applications grows (in other words, scalability issues are also relevant for a cooperative OLAP application scenario).

This gives raise to a novel OLAP computational paradigm: the multi-objective data cube compression, which deals with the problem of compressing data cubes according to *multiple goals* defined by *multiple requirements* rather than only one like in traditional schemes (e.g., a given space bound – see [11]).

From a theoretical point of view, the *nature* of multiple requirements strongly depends on the specific application context. As a meaningful instance, in our case of interest requirements are represented by different CQAH through which client applications access and query the target OLAP server, thus by the need for accomplishing (i.e., answering efficiently) a large family of even different OLAP queries. While in this research effort we thus consider hierarchies of queries as multiple requirements, in another research effort [13] we instead consider the *nature* (i.e., the *typology*) of queries as multiple requirements. Under a broader vision, both solutions could be integrated in a unique, comprehensive *multiple-objective approximate OLAP query engine*.

Indeed, queries are popular entities used to model different requirements in database and data-warehouse multiple-goal application scenarios (e.g., [52]), as they play a critical role with respect to the effectiveness of the overall underlying computational model. In fact, it should be noted that if CQAH of a given client application are very “different” from aggregations of the final compressed data cube (e.g., with respect to the structure and size of intermediate ranges), gross approximation errors are obtained as traditional *linear interpolation* techniques (i.e., approximation techniques based on the so-called *Continuous Values Assumption* (CVA) [8]) fail (e.g., see [7,11]), and the meant data cube compression becomes ineffective in practice. On the other hand, it is a matter of fact noticing that, in our specific application context, an *arbitrary* compression of the data cube could easily origin the undesired situation in which some client applications could be favorite by the compression process (i.e., retrieved approximate answers have a high degree of accuracy) whereas some other applications could not be favorite (i.e., retrieved approximate answers have a low degree of accuracy). As a consequence, our idea of generating an “intermediate” compressed representation of the target data cube makes sense, as, on the average, a *fair* final compressed representation is obtained, i.e. retrieved approximate answers have acceptable/good accuracy *in most cases*.

Similarly to the nature of requirements, the generation of the intermediate compressed representation strongly depends on the particular application context. Without going into details, in our approach we make use of a *greedy algorithm*, called `computeMObHist`, which is able to generate the intermediate compressed representation of the input data cube \mathcal{L} within the available space bound \mathcal{B} . The output of the compression process is represented by the compressed data cube, $\tilde{\mathcal{L}}$, and its in-memory-representation is a *hierarchical multidimensional histogram*, denoted by $\text{MOB-Hist}(\mathcal{L})$, particularly suitable to evaluate (yet introducing approximation) resource-intensive OLAP queries such as range-SUM queries, which, if evaluated against the original data cube \mathcal{L} , would introduce excessive computational overheads intolerable with interactive decision-support-oriented scenarios (e.g., see [11]).

For the sake of simplicity, models, algorithms and experiments proposed in this paper consider two-dimensional data cubes, which are an interesting case of popular cubes. Obviously, our proposed technique is valid for more general multidimensional data cubes, and scalability issues should be accurately investigated in a further research effort. As mentioned above, at the query layer we consider range-queries, which are a popular class of OLAP queries allowing us to extract summarized knowledge from massive data cubes on the basis of popular SQL aggregation operators like COUNT and SUM.

4 Related Work

As stated in previous Sections, two main research areas have influenced our work: histogram-based data cube compression techniques, and multiple-objective query processing in Database and Data Warehousing systems. In the following we focus the attention on both of them.

Histograms. Histograms have been extensively studied and applied in the context of *Selectivity Estimation* [36,40,45,49], and are effectively implemented within the core layers of commercial systems (e.g., Oracle Database, IBM DB2 Universal Database, Microsoft SQL Server) to query optimization purposes. In statistical databases [39,54], histograms represent a method for approximating probability distributions derived from real-type or integer-type attribute values. They have also been extensively used in data mining activities, intrusion detection systems, scientific databases and so forth, i.e. in all those applications which (i) operate on huge numbers of detailed records, and (ii) extract useful knowledge from condensed information consisting of summary data (e.g., aggregate data). Histograms can reach a surprising efficiency and effectiveness in approximating the actual distributions of data starting from summarized information. This has led the research community to investigate the use of histograms in the fields of *DataBase Management Systems* (DBMS) [1,3,24,32,36,40,45,48], OLAP systems [7,9,16,17,47], and *Data Stream Management Systems* (DSMS) [22,23,55]. From this heterogeneity, it follows an astounding histogram capability of being suitable for different application contexts, as highlighted by Ioannidis [31].

Apart from selectivity estimation, compressing multidimensional domains (such as general relations, e.g., [48], or OLAP data cubes, e.g. [57]) is another basic problem addressed by histograms. Here, the solution consists in approximating the *joint data distribution* of multiple attributes, obtained by jointly combining the distinctive distributions of singleton attributes. To this end, a popular and conventional approach is based on the well-known *Attribute-Value Independence* (AVI) assumption [50], according to which any query Q involving a set of attributes can be answered by applying Q on each attribute singularly. This approach is theoretically reasonable, but it has been recognized as source of gross errors in practice (e.g., [19,45,48]). To cope with this problem, histograms use a small number of mutually disjoint blocks, called buckets, to *directly* approximate the joint data distribution. Then, for each bucket, histograms store some aggregate information computed over the corresponding range of values, like the sum of values (given by the SQL aggregation operator SUM), or the

number of occurrences in that range (given by the SQL aggregation operator COUNT), such that this information retains a certain *summarizing content*.

However, even if reasonable, this approach introduces serious limitations when applied to real-life multidimensional databases and data cubes. Indeed, constructing histograms is much harder for two dimensions than for the one-dimensional case, as recognized in [42]. This problem gets worse in the case in which the dimension number increases, and becomes a problematic bottleneck in real-life data-intensive systems where corporate multidimensional databases and data cubes with more than 100 dimensions can be found. From this evidence, various and heterogeneous alternatives to the problem of computing multidimensional histograms have been proposed in literature, each of them based on one or more properties of data distributions characterizing the input data domain. Conventional approaches take into consideration statistical and error-metrics-based properties of data distributions. Other approaches expose instead *greedy solutions*, as traditional approaches introduce excessive computational overheads on highly-dimensional data domains.

In the following we focus the attention on the main representative instances of the histogram class, which are also close to our work: *Equi-Depth* [40], *MHist* [48], *Min-Skew* [1], *GenHist* [24], and *STHoles* [6] histograms.

Given a d -dimensional data domain \mathcal{D} (e.g., a data cube), *Equi-Depth histogram* $H_{E-D}(\mathcal{D})$ [40] is built as follows: (i) fix an ordering of the d dimensions; (ii) set $\alpha \approx |\mathcal{D}|^{\frac{1}{d}}$ root of desired number of buckets; (iii) initialize $H_{E-D}(\mathcal{D})$ to the input data distribution of \mathcal{D} ; (iv) for each k in $\{0, 1, \dots, |\mathcal{D}|-1\}$ split each bucket in $H_{E-D}(\mathcal{D})$ in α equi-depth partitions along d_k ; finally, (v) return resulting buckets to $H_{E-D}(\mathcal{D})$. This technique is affected by some limitations: fixing α and the dimension ordering can result in poor partitions, and, consequently, there could be a limited level of *bucketization* that, in turn, involves low quality of $H_{E-D}(\mathcal{D})$ in its general goal of approximating \mathcal{D} .

MHist histogram [48] overcomes the performance of *Equi-Depth*. *MHist* build procedure depends on the parameter p (specifically, such histogram is denoted by *MHist-p*). Contrary to the previous technique, at each step, the bucket b_i in a *MHist* histogram $H_{MH}(\mathcal{D})$ containing the dimension d_k whose *marginal* is the most in need of partitioning is chosen, and it is split along d_k into p (e.g., $p = 2$) buckets. Experimental results shown in [48] state that *MHist* overcomes AVI and *Equi-Depth*.

Min-Skew histogram was originally proposed in [1] to tackle the problem of selectivity estimation of *spatial data* in *Geographical Information Systems* (GIS). *Min-Skew* is more sophisticated than *MHist*. The main idea behind a *Min-Skew* histogram $H_{M-S}(\mathcal{D})$ is to follow the criterion of minimizing the *spatial skew* of the histogram by performing a *Binary Space Partitioning* (BSP) via recursively dividing the space along one of the dimensions each time. More formally, each point in the space of a given GIS instance is associated to a *spatial density*, defined as the number of *Minimal Bounding Rectangles* (MBR) that contain such a point. When performing the partition, each bucket b_i is assigned the spatial skew s_i , defined as follows: $s_i = \sum_{j=0}^{n_i-1} (f_j - \bar{f})^2 / n_i$, where (i) n_i is the number of points contained within b_i , (ii) f_j is the *spatial frequency* of the j -th point within b_i , and (iii) \bar{f} represents the *average frequency* of all the points within b_i . The total skew S of $H_{M-S}(\mathcal{D})$ is defined as

follows: $S = \sum_{i=0}^{B-1} n_i \cdot s_i$, where (i) B is the total number of buckets, (ii) s_i is the spatial skew associated with bucket b_i , and (iii) n_i is the number of points of b_i . The construction technique of $H_{M-S}(\mathcal{D})$ tries, at each step, to minimize the overall spatial skew S of the histogram by selecting (i) a bucket to be split, (ii) a dimension of the multi-dimensional space along which split, and (iii) a splitting point along that dimension such that the overall spatial skew computed after the split is smaller than the one computed at the previous step (i.e., before the current split).

[24] proposes *GenHist histogram*, whose key idea is the following. Given an histogram H with h_b buckets on an input data domain \mathcal{D} , a *GenHist* histogram $H_{GH}(\mathcal{D})$ is built via finding n_b overlapping buckets on H , such that n_b is an input parameter. To this end, the technique individuates the number of distinct regions that is much larger than the original number of buckets h_b , thanks to a greedy algorithm that considers *increasingly-coarser grids*. At each step, this algorithm selects the set of cells I of highest density, and moves enough randomly-selected points from I into a bucket in order to make I and its neighbors “close-to-uniform”. Therefore, the novelty of the proposal consists in defining a truly multidimensional splitting policy, based on the concept of *tuple density*.

[6] proposes a kind of multidimensional histogram based on the analysis of the query-workload against it: the *workload-aware histogram*, which they call *STHoles*. Rather than an arbitrary overlap, a *STHoles* histogram $H_{ST}(\mathcal{D})$ allows bucket nesting, thus achieving the definition of the so-called *bucket tree*. Query-workload is handled as follows: the query result stream \mathcal{Q}^R to be analyzed is intercepted and, for each query Q_i belonging to \mathcal{Q}^R and for each bucket b_i belonging to the current bucket tree, the number $|Q_i \cap b_i|$ is counted. Then, “holes” in b_i for regions of different *tuple density* are “drilled” and “pulled out” as children of b_i . Finally, buckets of similar densities are merged in such a way to keep the number of buckets constant. $H_{ST}(\mathcal{D})$ makes use of a tree-like in-memory-data-structure, since the parent-child relationship in a tree is comparable to the nesting relationship, and the sibling-sibling relationship is represented by buckets nested within the same external bucket. The construction algorithm of $H_{ST}(\mathcal{D})$ does not take into account the original data set; indeed, the needed information is instead gathered via inspecting the target query-workload and query feedbacks. This amenity makes $H_{ST}(\mathcal{D})$ *self tunable*, i.e. adaptable to updates and modifications that can occur in the original data cube. On the basis of this approach, a relevant amount of the total storage space available for housing the histogram is invested in representing “heavy-queried regions”, thus providing a better approximation for such regions, whereas a fewer storage space is reserved to “lowly-queried regions”, admitting some inaccuracy for such regions (however, tolerable in OLAP context [9]). A thorough set of experimental results on both real-life and synthetic data sets demonstrates that *STHoles* overcomes performance of *Equi-Depth*, *MHist*, and *GenHist*; in addition to this, in [6] authors also show that, on the DBMS Microsoft SQL Server, query-workload analysis overheads introduced by *STHoles* are very low, less than 10 % of the overall DBMS throughput.

Multiple-Objective Query Processing in Database and Data Warehousing Systems. As highlighted in Sect. 1, multiple-objective query processing in Database and

Data Warehousing Systems count sporadic initiatives, and therefore this one can be reasonably considered as a novel and, thus, challenging research field. In the following, we review some of them. Much work has been done in the context of the view selection and materialization problem, which aims at efficiently computing data cubes from the relational data source by efficiently exploiting the cuboid lattice [27]. [52,53] introduce the basic *multi-query optimization problem* in the view selection and materialization context as related to *transient views* rather than *permanent views* like in conventional approaches [27]. Transient views refer to views that are temporarily materialized, e.g. during the evaluation of complex OLAP queries against very large Data Warehouses. [40] devises an innovative solution for the view selection problem via proposing an integrated methodology taking into consideration the selection of both transient and permanent views in a combined manner. Indeed, this aspect can be reasonably intended as innovative with respect to previous similar research experiences. Multiple-query requirements for transient views are in [40] accommodated by means of a greedy algorithm that runs on the so-called AND-OR/DAG, first introduced by Gupta in [25], which model the hierarchies of SPJ operations that allow us to derive a certain view from given base relations. In some sense, [40], which can be reasonably considered as a previous significant research contribution to our research, confirms the benefits of our approach that, although for a different context, still make use of a greedy solution to compute the final compressed data cube. [3] deals with the problem of extracting Pareto distributions of database objects from relational database systems. This problem is related to ranking database query results, top- k and skyline query processing. In [3], the focus is on how to handle multiple conflicting selection criteria over queries, and the solution is represented by a multi-objective retrieval algorithm that improves the naïve Pareto-based solution. Contrary to previous approaches, this algorithm limits the additional number of scans that still must be done in conventional approaches in order to finally select the database objects belonging to the optimal solution. Skyline query processing [2,4,45] is one step before the latter research work, and deals with the problem of extracting tuple bags that satisfy multiple user preferences from a relational data source. In this case, existent approaches make extensive use of the Pareto-based solution, targeted to relational query processing. Finally, [36] considers the special case of optimizing multiple queries for OLAP data cubes, and [20] deals with the problem of supporting multiple-objective query processing in data aggregation by means of *joint optimization models* and *gathered statistics over data*.

5 Multiple-Objective Compression of Data Cubes

The multiple-objective compression process we propose is accomplished by means of the following multi-step approach: **(1)** for each dimension d of \mathcal{L} , for each level ℓ of d starting from the bottom (i.e., according to a bottom-up strategy), generate the ordered union of members at level ℓ of CQAH in \mathcal{W}_d , thus obtaining the so-called Multiple-Objective Level (MOL) at level ℓ , denoted by MOL_ℓ – let r be the (common) depth of CQAH in \mathcal{W}_d , first step finally generates, for each dimension d of \mathcal{L} , a set of r MOLs, denoted by $\text{MOL}_d(\mathcal{L}) = \{\text{MOL}_0, \text{MOL}_1, \dots, \text{MOL}_{r-1}\}$, and, in turn, the union set

obtained from latter sets, denoted by $\mathcal{MOL}(\mathcal{L}) = \{\mathcal{MOL}_0(\mathcal{L}), \mathcal{MOL}_1(\mathcal{L}), \dots, \mathcal{MOL}_{n-1}(\mathcal{L})\}$; **(2)** for each dimension d of \mathcal{L} , hierarchically merge MOLs in $\mathcal{MOL}_d(\mathcal{L})$ according to their original references, plus an additional, artificial aggregation ALL used to obtain a unique general tree rather than a list of trees, thus obtaining the so-called Multiple-Objective Aggregation Hierarchy (MOAH) for the dimension d , denoted by \mathcal{H}_{M_d} , and, in turn, the whole set of MOAHs for \mathcal{L} , denoted by $\mathcal{MW}(\mathcal{L}) = \{\mathcal{H}_{M_0}, \mathcal{H}_{M_1}, \dots, \mathcal{H}_{M_{n-1}}\}$, one for each dimension d of \mathcal{L} ; **(3)** for each level ℓ (recall that we assume hierarchies having all the same depth), generate a Generalized Partition (GP) of \mathcal{L} at level ℓ , denoted by $\mathcal{G}_\ell(\mathcal{L})$, such that buckets in $\mathcal{G}_\ell(\mathcal{L})$ are obtained via (i) projecting axes along members of MOAHs in $\mathcal{MW}(\mathcal{L})$ at level ℓ , and (ii) storing the sum of items they contain – the collection of GPs (one for each level ℓ of \mathcal{L} , denoted by $\mathcal{MOB}\text{-Set}(\mathcal{L})$, constitutes the “sketch” of $\mathcal{MOB}\text{-Hist}(\mathcal{L})$, meaning that from $\mathcal{MOB}\text{-Set}(\mathcal{L})$ we finally obtain $\mathcal{MOB}\text{-Hist}(\mathcal{L})$ according to our greedy algorithm `computeMOBHist` that realizes what we call the refinement process (described next); **(4)** (algorithm `computeMOBHist`) for each level ℓ , obtain from $\mathcal{G}_\ell(\mathcal{L})$ of $\mathcal{MOB}\text{-Set}(\mathcal{L})$ the so-called Multiple-Objective Partition (MOP) of \mathcal{L} at level ℓ , denoted by $\mathcal{MOP}_\ell(\mathcal{L})$, via merging buckets in $\mathcal{G}_\ell(\mathcal{L})$ with the criterion that the final partition of $\mathcal{G}_\ell(\mathcal{L})$ must be able to fit, at level ℓ , all the different query requirements of OLAP client applications, according to a best effort approach; **(5)** return $\mathcal{MOB}\text{-Hist}(\mathcal{L})$.

In the following, we focus our attention on algorithm `computeMOBHist`, which is the most important task of our technique. From the description above, note that, due the particular process generating the GP at level ℓ , the so-obtained buckets are all the buckets that would allow us to provide, at level ℓ , approximate answers having the highest accuracy for all OLAP applications, as these buckets cover, for each application, the desired CQAH. The same for the other GPs of $\mathcal{MOB}\text{-Set}(\mathcal{L})$. Obviously, it is not possible to materialize all the buckets of all the GPs of $\mathcal{MOB}\text{-Set}(\mathcal{L})$, due to the constraint posed by the space bound \mathcal{B} . If this would be the case, we finally would obtain the histogram $\mathcal{MOB}\text{-Hist}(\mathcal{L})$ as corresponding to $\mathcal{MOB}\text{-Set}(\mathcal{L})$ directly, i.e. as a full materialization of $\mathcal{MOB}\text{-Set}(\mathcal{L})$. Therefore, we adopt the strategy of obtaining the MOP $\mathcal{MOP}_\ell(\mathcal{L})$ as a novel partition over buckets in $\mathcal{G}_\ell(\mathcal{L})$. We also superimpose the constraint of obtaining a number of buckets in $\mathcal{MOP}_\ell(\mathcal{L})$ lower than the number of buckets in $\mathcal{G}_\ell(\mathcal{L})$ (i.e., $|\mathcal{MOP}_\ell(\mathcal{L})| < |\mathcal{G}_\ell(\mathcal{L})|$). This allows us to reduce the overall final size of $\mathcal{MO}_\ell(\mathcal{L})$ and, as a consequence, the overall final size of $\mathcal{MOB}\text{-Hist}(\mathcal{L})$, while, at the same time, ensuring approximate answers with acceptable/good accuracy. The above-described one is the main goal of algorithm `computeMOBHist`, which allows us to finally find a sub-optimal partition of each $\mathcal{G}_\ell(\mathcal{L})$ (i.e., $\mathcal{MOP}_\ell(\mathcal{L})$) able to accommodate, as much as possible, the multiple query requirements posed by OLAP client applications, according to a best effort approach. In order to adequately accomplish the above-illustrated task, `computeMOBHist` introduces a global strategy and a local strategy. The first one deals with the problem of how to explore the overall hierarchical search (bucket) space represented by

$\mathcal{MOB}\text{-Set}(\mathcal{L})$. The second one deals with the problem of how to explore the search (bucket) space represented by a given $\mathcal{G}_\ell(\mathcal{L})$ at level ℓ .

First, consider the latter one, which properly realizes the main greedy criterion used to obtain a $\mathcal{MOP}_\ell(\mathcal{L})$ from a given $\mathcal{G}_\ell(\mathcal{L})$. Since we need to accommodate multiple queries shaped by even-very-different CQAH, we introduce a global average relative error metrics, denoted by $\overline{\mathcal{E}}_\ell(\mathcal{L})$, and, thanks to a greedy strategy, we perform those aggregations of buckets (of $\mathcal{G}_\ell(\mathcal{L})$) into novel buckets (of $\mathcal{MOP}_\ell(\mathcal{L})$) that, overall, allow us to achieve the greatest benefit towards the minimization of $\overline{\mathcal{E}}_\ell(\mathcal{L})$. $\overline{\mathcal{E}}_\ell(\mathcal{L})$ is defined as follows:

$$\overline{\mathcal{E}}_\ell(\mathcal{L}) = \frac{1}{|\mathcal{MOP}_\ell(\mathcal{L})|} \cdot \sum_{k=0}^{|\mathcal{MOP}_\ell(\mathcal{L})|-1} \mathcal{E}_{rel}(B_k) \quad (1)$$

such that, for each bucket B_k in $\mathcal{MOP}_\ell(\mathcal{L})$, $\mathcal{E}_{rel}(B_k)$ models the *relative error* due to evaluating the query that corresponds to B_k . $\mathcal{E}_{rel}(B_k)$ is defined as follows:

$$\mathcal{E}_{rel}(B_k) = \frac{|\mathcal{A}(B_k) - \tilde{\mathcal{A}}(B_k)|}{\mathcal{A}(B_k)} \quad (2)$$

where $\mathcal{A}(B_k)$ and $\tilde{\mathcal{A}}(B_k)$ are the exact and the approximate answers to the query corresponding to B_k , respectively. $\mathcal{A}(B_k)$ is evaluated against the input data cube \mathcal{L} at level ℓ , whereas $\tilde{\mathcal{A}}(B_k)$ is instead evaluated on the *current* MOP $\mathcal{MOP}_\ell(\mathcal{L})$.

A critical point is represented by the greedy criterion used to select the *actual* bucket. In this respect, we inherit the strategy adopted by traditional multidimensional histograms (e.g., *MHist* [48]), which aim at obtaining final buckets that store uniform data via minimizing the *skewness* (i.e., the asymmetry – e.g., see [9]) among buckets themselves. This, in turn, has beneficial effects on the accuracy of approximate answers computed against the histogram, as widely recognized (e.g., see [9]). On the basis of this main intuition, at each step of the local computation of `computeMOB-Hist`, we *greedily select the most uniform bucket*, said B_U , among buckets of the overall bucket space of $\mathcal{G}_\ell(\mathcal{L})$. Then, we check the value of metrics $\overline{\mathcal{E}}_\ell(\mathcal{L})$ on the bucket set composed by buckets of the *current* partition $\mathcal{MOP}_\ell(\mathcal{L})$ plus the “new” bucket B_U to test whether a benefit is obtained or not (i.e., test if a minimization of $\overline{\mathcal{E}}_\ell(\mathcal{L})$ is obtained). If the latter is the case, we add B_U to $\mathcal{MOP}_\ell(\mathcal{L})$, otherwise we flag B_U in order to be further processed. Then, by meaningfully exploiting the clustered nature of OLAP data [8], we assert that *neighbor buckets* of B_U , denoted by $\mathcal{N}(B_U)$, have *quasi-similar homogeneity with respect to the one of B_U* . Therefore, to compression purposes, we adopt the strategy of *merging* B_U and buckets in $\mathcal{N}(B_U)$ with the aim of finally obtaining a *larger* bucket with still a suitable degree of homogeneity, having fixed a threshold value V_U that limits the maximal difference between the homogeneity of B_U and those of buckets in $\mathcal{N}(B_U)$. If the maximal difference is

within the threshold V_U , then B_U and buckets in $\mathcal{N}(B_U)$ are finally merged into a novel bucket of $\mathcal{MOP}_\ell(\mathcal{L})$. Otherwise, they are separately materialized.

To meaningfully support the process described above, given a candidate bucket B of $\mathcal{G}_\ell(\mathcal{L})$, we adopt as homogeneity definition the *greatest quadratic distance* from the average value of *outliers* in B , meaning that the less is such a distance, the more is the homogeneity of B . This approach is modeled by the function $unif(B)$, defined as follows:

$$unif(B) = \frac{1}{\max_{i \in Out(B)} \{0, |B[i] - AVG(B)|^2\}} \quad (3)$$

such that (i) $Out(B)$ is the set of outliers of B , (ii) $B[i]$ is the value of the i -th item of B , and (iii) $AVG(B)$ is the average value of items in B . Note that we make use of the second power to ensure the convergence of the definition. Outlier detection (e.g., [16,17]) is a non-trivial engagement. Without going into detail, in our approach outlier detection can be implemented by means of any of the method proposed in literature (conventional or not), so that this aspect of our work is orthogonal to the main contribution. When a bucket and its neighbors must be merged in a singleton bucket, said B_M , we impose the criterion of obtaining B_M as a *hyper-rectangular bucket* instead of an arbitrary bucket (i.e., a bucket with irregular shape). In doing this, based on geometrical issues, we simply “throw away” protruding parts of merged neighboring buckets in such a way as to obtain a “maximal”, “internal” hyper-rectangular bucket. In turn, the protruding bucket parts are then materialized as new buckets of the GP $\mathcal{G}_\ell(\mathcal{L})$, and then the same process is iterated.

For what regards the second aspect of `computeMOBHist` (i.e., the global strategy), we adopt an *in-depth visit* of $\mathcal{MOB}\text{-Set}(\mathcal{L})$ starting from the aggregation ALL (i.e., from the corresponding GP at level 0, $\mathcal{G}_0(\mathcal{L})$), meaning that when a merged bucket B_M is obtained in the GP $\mathcal{G}_\ell(\mathcal{L})$ at level ℓ , we hierarchically move down to the GP $\mathcal{G}_{\ell+1}(\mathcal{L})$ at level $\ell + 1$, and consider the collection of buckets of $\mathcal{G}_{\ell+1}(\mathcal{L})$ contained by B_M , and so on. When the leaf level GP is reached, we re-start from the aggregation ALL. As said before, this process is bounded by the storage space \mathcal{B} .

Without going into details, it should be noted that the in-depth visit approach is in favor of the idea of accommodating a large family of OLAP queries rather than OLAP queries referred to a particular level of the logical hierarchy underlying $\mathcal{MOB}\text{-Hist}(\mathcal{L})$. Basically, the above-described one defines a *top-down* approach in the computation of $\mathcal{MOB}\text{-Hist}(\mathcal{L})$. The rationale of this way to do comes from arguing that in a typical OLAP scenario, client applications are mainly interested in querying OLAP data at granularities different from the lowest one (e.g., see [14,15]). Finally, pseudo-code of `computeMOBHist` is listed in Fig. 1. The meaning of the code syntax can be easily derived from the previous description of the algorithm.

Algorithm `computeMOBHist`

Input: The collection of GPs $\mathcal{MOB}\text{-Set}(\mathcal{L})$; the space bound \mathcal{B} ; the threshold value V_U .

Output: The multiple-objective histogram $\mathcal{MOB}\text{-Hist}(\mathcal{L})$.

begin


```

 $\mathcal{M} \leftarrow \emptyset;$ 
 $\ell \leftarrow 0;$ 
 $MO\acute{b}\text{-Hist}(\mathcal{L}).add(\mathcal{M}, \ell);$ 
 $\mathcal{G} \leftarrow \emptyset;$ 
 $\mathcal{E} \leftarrow \emptyset;$ 
 $\mathcal{E}' \leftarrow \emptyset;$ 
 $B \leftarrow \emptyset;$ 
 $B_U \leftarrow \emptyset;$ 
 $B_M \leftarrow \emptyset;$ 
 $mergeVector \leftarrow \emptyset;$ 
 $mergeVector.initialize();$ 
 $\mathcal{G} \leftarrow MO\acute{b}\text{-Set}(\mathcal{L}).get(\ell);$  //  $\ell$  is already equal to 0
 $boundingMDRegion \leftarrow \mathcal{G}.getRegion();$ 
while ( $B > 0$ ) do
   $\mathcal{G} \leftarrow MO\acute{b}\text{-Set}(\mathcal{L}).get(\ell);$ 
   $\mathcal{M} \leftarrow MO\acute{b}\text{-Hist}(\mathcal{L}).get(\ell);$ 
   $\mathcal{E} \leftarrow getErrorMetrics(\mathcal{G}, \mathcal{M});$ 
   $B_U \leftarrow findMaxUnifBucket(\mathcal{G});$ 
   $\mathcal{M}.add(B_U);$ 
   $B_U.flagAsVisited();$  // this avoids that current bucket  $B_U$  can be selected
                        // as the most uniform at next WHILE cycles
   $\mathcal{E}' \leftarrow getErrorMetrics(\mathcal{G}, \mathcal{M});$ 
  if ( $\mathcal{E}' \geq \mathcal{E}$ ) then
     $\mathcal{M}.remove(B_U);$ 
     $B_U.unflagAsVisited();$ 
  else
     $B \leftarrow getNextNeighBucket(B_U, boundingMDRegion);$ 
    while ( $abs(unif(B) - unif(B_U)) \leq V_U$ ) do
       $mergeVector.add(B);$ 
       $B \leftarrow getNextNeighBucket(B_U, boundingMDRegion);$ 
    end;
     $B_M \leftarrow merge(mergeVector);$ 
     $\mathcal{M}.add(B_M);$ 
     $B_M.flagAsVisited();$ 
     $B \leftarrow B - computeOccupancy(MO\acute{b}\text{-Hist}(\mathcal{L}), \mathcal{M}, B_U, B_M);$  // retrieves the overall space size
                                                                    // needed to represent the new
                                                                    // buckets  $B_U$  and  $B_M$  within the
                                                                    // current MOP  $\mathcal{M}$  of
                                                                    //  $MO\acute{b}\text{-Hist}(\mathcal{L})$ 
  end;
   $\ell \leftarrow \ell + 1;$  // in-depth visit of  $MO\acute{b}\text{-Set}(\mathcal{L})$ 
   $MO\acute{b}\text{-Hist}(\mathcal{L}).add(\mathcal{M}, \ell);$  // if the MOP  $\mathcal{M}$  at level  $\ell$  of  $MO\acute{b}\text{-Hist}(\mathcal{L})$  already
                                // exists, this is a NULL operation
   $boundingMDRegion \leftarrow B_M;$ 
  if ( $\ell \geq MO\acute{b}\text{-Set}(\mathcal{L}).getDepth()$ ) then
     $\ell \leftarrow 0;$ 
     $\mathcal{G} \leftarrow MO\acute{b}\text{-Set}(\mathcal{L}).get(\ell);$  //  $\ell$  is already equal to 0
     $boundingMDRegion \leftarrow \mathcal{G}.getRegion();$ 

```

```

end;
end;
return MOB-Hist(L);
end;
    
```

Fig. 1. Algorithm computeMOBHist

6 Experimental Assessment

In this Section, we describe the experimental analysis of our proposed technique, and discuss derived results.

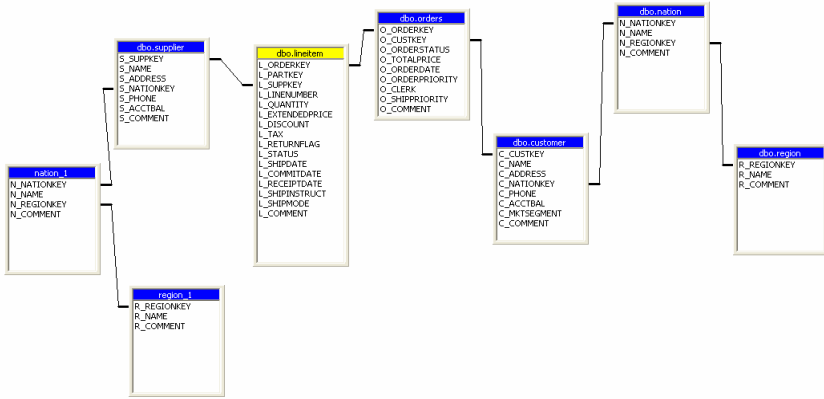


Fig. 2. Two-dimensional data cube on the benchmark data set TPC-H

As regards the data layer of our experimental framework, we engineered two different kinds of data cubes. The usage of different classes of data cubes allowed us to submit our proposed technique to a comprehensive and “rich” experimental analysis, and, as a consequence, carefully test its performance. Particularly, we considered *benchmark data cubes*, which allow us to test the effectiveness of our technique under the stressing of an in-laboratory-built input, and to evaluate our technique against competitor ones on “well-referred” data sets that have been widely used in similar research experiences. We also conducted experiments on *real* and *synthetic data cubes*. However, for space reasons, we show experimental results on benchmark data cubes, which, typically, are more probing than other ones.

In more detail, we considered two popular benchmarks: *TPC-H* [56] and *APB-1* [44], which are both well-known in the Database and Data Warehousing research communities. By exploiting data generation routines made available at the respective benchmark Web sites, we built benchmark two-dimensional data cubes by means of the well-known *Microsoft Analysis Services 2000* OLAP platform. In more detail, from the benchmark data set *TPC-H*, we built a $2,000 \times 2,000$ two-dimensional data cube (see Fig. 2) populated with 4M data cells and having as dimensions the attributes: (i) *C_Address*, belonging to the dimensional table *dbo.Customer* and linked to the fact table *dbo.Lineitem* through the dimensional table *dbo.Orders*, and (ii)

$S_Address$, belonging to the dimensional table $dbo.Supplier$. From the benchmark data set $APB-1$, we built a $2,000 \times 2,000$ two-dimensional data cube (see Fig. 3) populated with 4M data cells and having as dimensions the attributes: (i) $Class$, belonging to the dimensional table $dbo.Product$, and (ii) $Month$, belonging to the dimensional table $dbo.Time$.

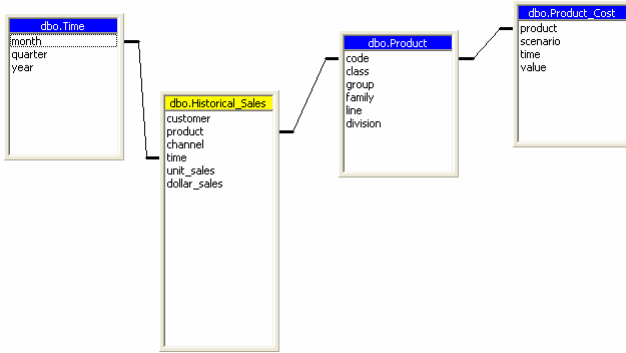


Fig. 3. Two-dimensional data cube on the benchmark data set $APB-1$

As regards the input of our experimental study, we considered *random populations* of synthetic range-SUM queries, modeled in our experimental framework by the set Q_S . For these queries, CQAH have been also obtained randomly with the constraint of “covering” the target data cube *as much as possible* (i.e., obtaining GPs having a large number of buckets) at, however, a “reasonable” granularity. *Query Selectivity* $\|\bullet\|$ is the controlled parameter used to cost the complexity needed to evaluate queries in Q_S (e.g., see [11]).

As regards the outcomes of our experimental study, given a population of synthetic range-SUM queries Q_S , we introduce the *Average Relative Error* (ARE) between exact and approximate answers to queries in Q_S , defined as in (2).

In our experimental study, we compared the performance of our proposed technique against the following well-known histogram-based techniques for compressing data cubes: *MinSkew* [1], *GenHist* [24], and *STHoles* [6]. In more detail, having fixed the space budget \mathcal{B} (i.e., the storage space available for housing the compressed data cube), we derived, for each comparison technique, the configuration of the input parameters that respective authors consider the best in their papers. This ensures a *fair* experimental analysis, i.e. an analysis such that each comparison technique provides its *best* performance. Furthermore, for all the comparison techniques, we set the space budget \mathcal{B} as equal to the $r\%$ of $size(\mathcal{L})$, being r the *compression ratio* and $size(\mathcal{L})$ the total occupancy of the input data cube.

Fig. 4 shows experimental results related to the percentage variation of ARE on both benchmark data cubes with respect to the selectivity of queries in Q_S . This allows us to measure the *quality* of compression techniques, i.e. their capability of introducing low query approximation errors. Fig. 5 shows the results for the same experiment when

ranging r on the interval $[5, 20]$ (i.e., \mathcal{B} on the interval $[5, 20]$ % of $size(\mathcal{L})$), and fixing the selectivity of queries to $\|Q\| = 750 \times 700$. This allows us to measure the *scalability* of compression techniques, which is a critical aspect in OLAP (e.g., see [9]).

Our experimental analysis, which has been conducted on two-dimensional benchmark data cubes, has clearly demonstrated the benefits deriving from applying our proposed data cube compression technique in the probing context of cooperative OLAP environments. In fact, not only our technique has shown a good performance with respect to the quality of approximate answers, but also it has revealed a low dependency on the compression ratio, i.e. a good scalability with respect to a “growing” input. It is a matter of fact noticing that both properties are relevant in dealing with massive data cubes processed within cooperative OLAP environments.

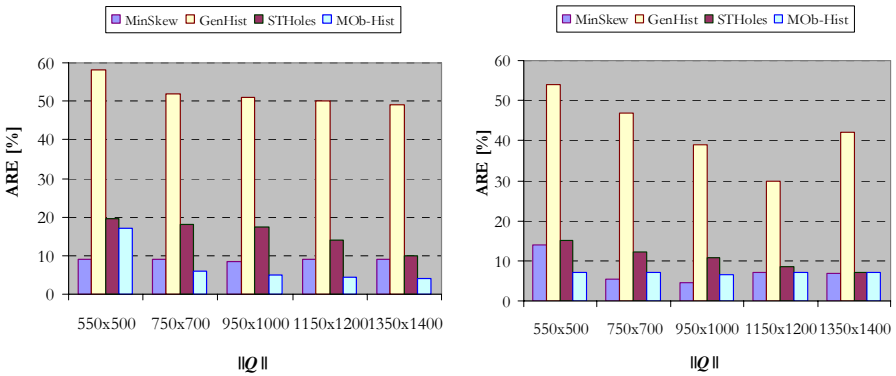


Fig. 4. ARE vs query selectivity $\|Q\|$ on the benchmark data cubes TPC-H (left) and APB-1 (right) with $r = 10$ %

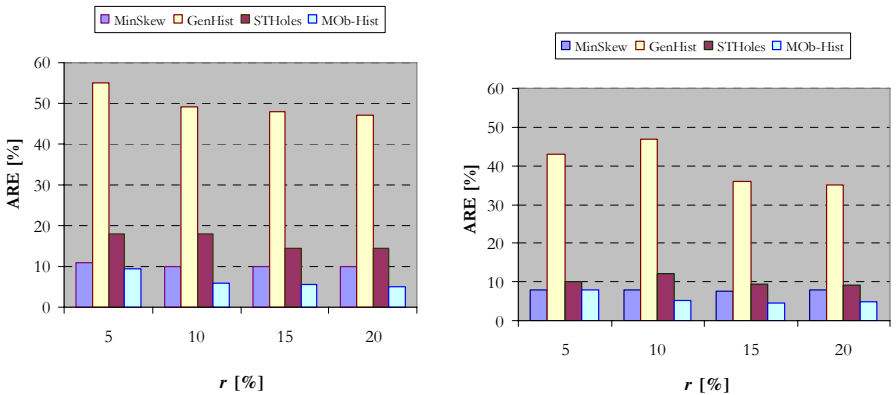


Fig. 5. ARE vs compression ratio r on the benchmark data cube TPC-H (left) and APB-1 (right) with $\|Q\| = 750 \times 700$

7 Conclusions and Future Work

In this paper, we have presented an innovative data cube compression technique that focuses on the context of cooperative OLAP environments, which play a leading role in next-generation advanced information systems. This particular environment gives rise to the definition of the so-called multiple-objective data cube compression paradigm, which is another contribution of this paper. Further, a secondary contribution of this paper is represented by the experimental analysis of the proposed multiple-objective data cube compression technique, which has demonstrated the limitations of traditional histogram-based data cube compression techniques and, symmetrically, the benefits deriving from our proposed technique in dealing with cooperative OLAP environments. Future work is mainly oriented to the issue of dealing with more complex OLAP aggregation predicates (beyond the traditional ones, i.e. SUM, COUNT, AVG), which, applied to multiple queries, pose novel and previously-unrecognized research challenges.

References

1. Acharya, S., et al.: Selectivity Estimation in Spatial Databases. *ACM SIGMOD*, 13–24 (1999)
2. Agrawal, R., Wimmers, E.L.: A Framework for Expressing and Combining Preferences. *ACM SIGMOD* (2000)
3. Balke, W.-T., Güntzer, U.: Multi-Objective Query Processing for Database Systems. *VLDB* (2004)
4. Börzsönyi, S., et al.: The Skyline Operator. *IEEE ICDE* (2001)
5. Bowman, I.T., Salem, K.: Optimization of Query Streams using Semantic Prefetching. *ACM TODS* 30(4) (2005)
6. Bruno, N., et al.: STHoles: A Multidimensional Workload-Aware Histogram. *ACM SIGMOD*, 211–222 (2001)
7. Buccafurri, F., et al.: A Quad-Tree based Multiresolution Approach for Two-Dimensional Summary Data. *IEEE SSDBM*, 127–140 (2003)
8. Colliat, G.: OLAP, Relational, and Multidimensional Database Systems. *ACM SIGMOD Record* 25(3), 64–69 (1996)
9. Cuzzocrea, A.: Overcoming Limitations of Approximate Query Answering in OLAP. *IEEE IDEAS*, 200–209 (2005)
10. Cuzzocrea, A.: Providing Probabilistically-Bounded Approximate Answers to Non-Holistic Aggregate Range Queries in OLAP. *ACM DOLAP*, 97–106 (2005)
11. Cuzzocrea, A.: Improving Range-Sum Query Evaluation on Data Cubes via Polynomial Approximation. *Data & Knowledge Engineering* 56(2), 85–121 (2006)
12. Cuzzocrea, A.: Accuracy Control in Compressed Multidimensional Data Cubes for Quality of Answer-based OLAP Tools. *IEEE SSDBM*, 301–310 (2006)
13. Cuzzocrea, A.: Top-Down Compression of Data Cubes in the Presence of Simultaneous Multiple Hierarchical Range Queries. In: An, A., Matwin, S., Raś, Z.W., Ślęzak, D. (eds.) *Foundations of Intelligent Systems. LNCS (LNAI)*, vol. 4994, pp. 361–374. Springer, Heidelberg (2008)
14. Cuzzocrea, A., et al.: A Hierarchy-Driven Compression Technique for Advanced OLAP Visualization of Multidimensional Data Cubes. In: Tjoa, A.M., Trujillo, J. (eds.) *DaWaK 2006. LNCS*, vol. 4081, pp. 106–119. Springer, Heidelberg (2006)

15. Cuzzocrea, A., et al.: Semantics-aware Advanced OLAP Visualization of Multidimensional Data Cubes. *International Journal of Data Warehousing and Mining* 3(4), 1–30 (2007)
16. Cuzzocrea, A., et al.: Answering Approximate Range Aggregate Queries on OLAP Data Cubes with Probabilistic Guarantees. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) *DaWaK 2004*. LNCS, vol. 3181, pp. 97–107. Springer, Heidelberg (2004)
17. Cuzzocrea, A., Wang, W.: Approximate Range-Sum Query Answering on Data Cubes with Probabilistic Guarantees. *Journal of Intelligent Information Systems* 28(2), 161–197 (2007)
18. Doan, A., Levy, A.Y.: Efficiently Ordering Plans for Data Integration. *IEEE ICDE* (2002)
19. Faloutsos, C., Kamel, I.: Relaxing the Uniformity and Independence Assumptions Using the Concept of Fractal Dimension. *Journal of Computer and System Sciences* 55(2), 229–240 (1997)
20. Fan, J., Kambhampati, S.: Multi-Objective Query Processing for Data Aggregation. *ASU CSE TR* (2006)
21. Fang, M., et al.: Computing Iceberg Queries Efficiently. *VLDB*, 299–310 (1998)
22. Guha, S., et al.: Histogramming Data Streams with Fast Per-Item Processing. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) *ICALP 2002*. LNCS, vol. 2380, pp. 681–692. Springer, Heidelberg (2002)
23. Guha, S., et al.: Data Streams and Histograms. *ACM STOC*, 471–475 (2001)
24. Gunopulos, D., et al.: Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes. *ACM SIGMOD*, 463–474 (2000)
25. Gupta, H.: Selection of Views to Materialize in a Data Warehouse. *ICDT*, 98–112 (1997)
26. Gupta, A., et al.: Approximate Range Selection Queries in Peer-to-Peer Systems. *CIDR* (2003), <http://www-db.cs.wisc.edu/cidr/cidr2003/program/p13.pdf>
27. Harinarayan, V., et al.: Implementing Data Cubes Efficiently. *ACM SIGMOD*, 205–216 (1996)
28. Han, J., Kamber, M.: *Data Mining: Concepts and Techniques*. Morgan Kauffmann Publishers, San Francisco (2000)
29. Ho, C.-T., et al.: Range Queries in OLAP Data Cubes. *ACM SIGMOD*, 73–88 (1997)
30. Ioannidis, Y.: Universality of Serial Histograms. *VLDB*, 256–267 (1993)
31. Ioannidis, Y.: The History of Histograms (abridged). *VLDB*, 19–30 (2003)
32. Ioannidis, Y., Poosala, V.: Histogram-based Approximation of Set-Valued Query Answers. *VLDB*, 174–185 (1999)
33. Ives, Z.G., et al.: An Adaptive Query Execution System for Data Integration. *ACM SIGMOD* (1999)
34. Jin, R., et al.: A Framework to Support Multiple Query Optimization for Complex Mining Tasks. *MMDM* (2005)
35. Jin, R., et al.: Simultaneous Optimization of Complex Mining Tasks with a Knowledgeable Cache. *ACM SIGKDD* (2005)
36. Kalnis, P., Papadias, D.: Multi-Query Optimization for On-Line Analytical Processing. *Information Systems* 28(5) (2003)
37. Kooi, R.P.: *The Optimization of Queries in Relational Databases*. PhD Thesis, Case Western Reserve University (1980)
38. Koudas, N., et al.: Optimal Histograms for Hierarchical Range Queries. *ACM PODS*, 196–204 (2000)
39. Malvestuto, F.: A Universal-Scheme Approach to Statistical Databases Containing Homogeneous Summary Tables. *ACM Transactions on Database Systems* 18(4), 678–708 (1993)
40. Mistry, H., et al.: Materialized View Selection and Maintenance using Multi-Query Optimization. *ACM SIGMOD* (2001)

41. Muralikrishna, M., DeWitt, D.J.: Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries. *ACM SIGMOD*, 28–36 (1998)
42. Muthukrishnan, S., et al.: On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. *ICDT*, 236–256 (1999)
43. Nie, Z., Kambhampati, S.: Joint Optimization of Cost and Coverage of Query Plans in Data Integration. *ACM CIKM*, 223–230 (2001)
44. OLAP Council: Analytical Processing Benchmark 1, Release II (1998), http://www.symcorp.com/downloads/OLAP_CouncilWhitePaper.pdf
45. Papadias, D., et al.: An Optimal and Progressive Algorithm for Skyline Queries. *ACM SIGMOD* (2003)
46. Poosala, V.: Histogram-based Estimation Techniques in Database Systems. PhD Thesis, University of Wisconsin-Madison (1997)
47. Poosala, V., Ganti, V.: Fast Approximate Answers to Aggregate Queries on a Data Cube. *IEEE SSDBM*, 24–33 (1999)
48. Poosala, V., Ioannidis, Y.: Selectivity Estimation Without the Attribute Value Independence Assumption. *VLDB*, 486–495 (1997)
49. Poosala, V., et al.: Improved Histograms for Selectivity Estimation of Range Predicates. *ACM SIGMOD*, 294–305 (1996), 1996
50. Roy, P., et al.: Efficient and Extensible Algorithms for Multi-Query Optimization. *ACM SIGMOD* (2000)
51. Selinger, P., et al.: Access Path Selection in a Relational Database Management System. *ACM SIGMOD*, 23–34 (1979)
52. Sellis, T.: Multiple-Query Optimization. *ACM TODS* 13(1) (1988)
53. Sellis, T., Ghosh, S.: On the Multiple-Query Optimization Problem. *IEEE TKDE* 2(2) (1990)
54. Shoshani, A.: OLAP and Statistical Databases: Similarities and Differences. *ACM PODS*, 185–196 (1997)
55. Thaper, N., et al.: Dynamic Multidimensional Histograms. *ACM SIGMOD*, 428–439 (2002)
56. Transaction Processing Council: TPC Benchmark H (2006), <http://www.tpc.org/tpch/>
57. Vassiliadis, P., Sellis, T.: A Survey of Logical Models for OLAP Databases. *ACM SIGMOD Record* 28(4), 64–69 (1999)
58. Wang, S., et al.: State-Slice: A New Paradigm of Multi-Query Optimization of Window-Based Stream Queries. *VLDB* (2006)
59. Xin, D., et al.: Answering Top-k Queries with Multi-Dimensional Selections: The Ranking Cube Approach. *VLDB*, 463–475 (2006)

Reference Management in a Loosely Coupled, Distributed Information System

Matthias Grossmann, Nicola Hönle, Daniela Nicklas, and Bernhard Mitschang

Universität Stuttgart, Institute of Parallel and Distributed Systems,
Universitätsstraße 38, 70569 Stuttgart, Germany
{grossmann,hoenle,nicklas,mitsch}@ipvs.uni-stuttgart.de

Abstract. References between objects in loosely coupled distributed information systems pose a problem. On the one hand, one tries to avoid referential inconsistencies like, e.g., dangling links in the WWW. On the other hand, using strict constraints as in databases may restrict the data providers severely. We present the solution to this problem that we developed for the Nexus system. The approach tolerates referential inconsistencies in the data while providing consistent query answers to users. For traversing references, we present a concept based on return references. This concept is especially suitable for infrequent object migrations and provides a good query performance. For scenarios where object migrations are frequent, we developed an alternative concept based on a distributed hash table.

1 Introduction

References between objects in distributed data sets can be found in many domains: foreign keys in distributed databases, references over XML documents, e.g., using XLinks [1], document links in peer-to-peer databases, or hyperlinks in the World Wide Web. There are two main challenges to address in approaches for reference management: the avoidance of referential inconsistencies (i.e., dangling references, Fig. 1a), and the bi-directional traversal of the references (Fig. 1b).

For both challenges, a wide spectrum of solutions exists that differ in the quality of consistency they ensure and in the assumptions they make about the behavior of the participating data sets, and thus, how closely or loosely they are coupled.

Relational databases use foreign key constraints and referential actions to ensure referential integrity among database tables. These tables are closely coupled, and the grade of consistency is high. On the other side, in the World Wide Web, dangling references between web pages are very frequent, since different web servers are loosely coupled and no referential integrity is guaranteed.

The retrieval of referenced objects in centralized databases is trivial. In distributed databases, catalogs store the information which objects are stored at

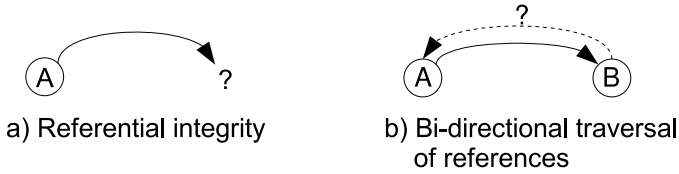


Fig. 1. Reference management

which node. With a growing number of participating nodes, the management of such catalogs becomes inefficient. Again, the World Wide Web uses URLs that contain the location of the objects which allows retrieval in constant time¹. No central catalog is needed. However, traversing references in the reverse direction is not possible and the migration of objects from one data provider to another requires the change of all references to this object.

In the Nexus project [2], we developed a federated architecture for the distributed management of large-scale context models. The architecture contains a large number of independent data providers, which are solely responsible for their local data sets. We allow references between objects stored on different data providers. Our solution for the two challenges stated above, which we present in this paper, is an approach that may be valuable for other distributed information systems as well.

Since we want to allow a high grade of autonomy for the data providers, we cannot implement strict referential actions over different data sets. To be more precise, a data provider A should be in full charge of its data set, i.e., no action of another data provider B should force A to delete or change an object in A 's data set. So, our approach allows referential inconsistencies, but is able to produce consistent query answers over the data sets. The federated system answers queries using a best effort semantic: If the result set would contain an object that has a dangling reference (like in Fig. 1a), the object should be returned, unless the user specifies a strict referential integrity semantic for the query, e.g. specifying return data from the referenced object. For the retrieval of objects and the bi-directional traversal of references, we use in our current implementation locators similar to URLs that contain the location of the storing data provider. However, we also show how our approach can be realized using distributed hash tables (DHTs).

The remainder of this paper is organized as follows: In Sect. 2, we give an overview of the related work. Section 3 contains an introduction to the basic concepts of the Nexus system. We present our solution for the problem of ensuring referential integrity in Sect. 4, and for the problem of the retrieval of objects in Sect. 5. For the second problem, we also give an alternative solution based on DHTs in Sect. 6, which is less efficient, but is better suited for dynamic scenarios with frequent object migrations.

¹ Aside from the overhead caused by DNS lookups when the location is not given as IP address.

2 Related Work

Integrating data from independent providers (peers) is the main objective of peer-to-peer (P2P) computing. In most P2P systems like Piazza [3] or the system envisioned in [4], the peers are more independent from each other than Nexus' data providers. Their data sets are self-contained and do not contain inter-peer references, and their schemas may differ heavily from each other. Consequently, the publications focus on schema mediation. Hyperion [5] supports Event-Condition-Action (ECA) rules, which can be used to enforce inter-peer consistency constraints. This approach would aim at keeping the complete data consistent, thus limiting the peer autonomy, which we want to avoid.

ECA concepts are also often used for maintaining integrity in federated or multidatabase systems. The approach taken in [6] is especially interesting with respect to our scenario, as it allows a fine-grained specification of constraints and as it accepts temporary violations of the constraints. So-called data dependency descriptors are used to specify source and target object sets for a constraint, an expression defining the constraint, a declaration when the constraint has to be satisfied and actions to be taken to restore a consistent state. In the Nexus context, this is a feasible approach for two different data providers to mutually agree on a way to restore referential integrity between their data sets, thus to some extent preserving their autonomy, but it does not provide consistent query answers while the constraints are violated and involves organizational overhead as a reference between objects of two different data providers requires the definition of a data dependency descriptor.

Gupta and Widom present in [7] a concept for reducing the overhead for checking integrity constraints in a distributed database. Under certain conditions, the compliance of a data manipulation operation with an integrity constraint referring to data stored at a remote node can be inferred from data stored at the local node. The paper considers arbitrary integrity constraints, but for the special case of referential integrity constraints, the concept is straightforward. Consider at node n_r the insertion of an object r referencing an object o stored at node n_o . If n_r already stores an object referencing o , this ensures that the insertion of o complies to the referential integrity constraint and a lookup for o at n_o is needless. We cannot use this concept in our scenario for two reasons. First, as outlined in Sect. 1, we have to accept inconsistencies in the data, so the presence of an object referencing o does not necessarily imply the presence of o , and second, we have to contact n_o anyway to enable the bi-directional traversal of references (cf. Sect. 5.1).

In [8] and [9], the problem of finding consistent answers to queries on an inconsistent database is discussed. The solution is based on repaired versions of the database: A repair of a database is a minimally altered version of the database which satisfies the integrity constraints. There may be more than one repair for a given database and a set of integrity constraints. The consistent answer to a query is the intersection of the results of the query on all repairs of the inconsistent database. Applying this approach to our referential integrity problem is possible, but implies a limitation contradicting our best effort concept. One

repair for a referential inconsistent database is to remove the object containing the invalid reference, hence, no query result will contain this object. We will discuss this further in Sect. 4.

Referential integrity and bi-directional references are well-known problems in the WWW and they have become more important with the advent of the XLink specification. To overcome these problems, link services have been proposed, e.g., by Davis in [10] and by Ciancarini et al. for XLinks in [11]. Davis replaces the URLs in the anchor elements in documents by unique document and content part names, and the address of the server storing the document is part of an entry for the reference in the link service. This approach is similar to the one we present in Sect. 6, however Davis does not suggest an implementation for the link service. Davis also classifies open hypermedia systems with respect to their approaches for dealing with dangling references. According to this classification, our approach most closely resembles to the ‘loosely coupled’ category with the additional and important feature of ensuring consistent query results.

A link service taking care of referential integrity was developed as part of the Aquarelle project [12]. It relies on the data providers to send update messages when they move or delete a document, so it cannot handle server or network failures.

3 The Nexus Platform

The Nexus platform is an open architecture for various context aware applications. It is based on a common, object-oriented context model, the so-called Augmented World Model (AWM). This context model is used to integrate context from various sources and to provide an abstraction for different context aware applications. It is used to model context data in different areas, like geographical data, dynamic sensor data, infrastructural context, or links to related information (e.g., HTML documents). The AWM can be easily extended with new object types for new types of applications.

For requesting AWM data, Nexus provides its own query language, the Augmented World Query Language (AWQL). An AWQL query mainly consists of a Boolean expression over predicates on attributes of objects, which determines which objects are included in the result set. Result sets are represented in the Augmented World Modeling Language (AWML). AWML treats objects as plain containers for attributes, and an object can contain more than one instance of a single attribute. The type of an object also is a normal attribute of the object. This allows for representing inconsistencies caused by different representations of the same real-world objects at different data sources. See [13] and [14] for more detailed presentations of AWQL, AWML and the AWM.

Nexus provides also basic services and value added services to provide this context information to applications. The architecture of the Nexus platform is divided into three tiers (cf. Fig. 2):

- The *service tier* contains different context servers that hold the context data, gathered by sensors or modeled by data domain experts of the different

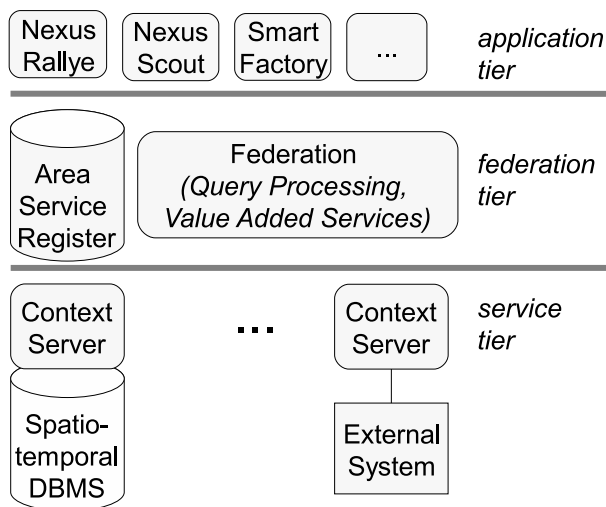


Fig. 2. Nexus Platform Architecture

context data providers. There are various realizations of context servers: e.g., for map data, a DBMS with spatial extensions can be used [2]. The global AWM is realized as a virtual integration of these local context models at the federation tier.

- The *federation tier* integrates context information. It holds numerous functional components that process the raw context data to provide more expressive data, and a registry for different context servers, the area service register (ASR). This registry contains information about the object types stored in each context server and its spatial service area that is defined as a polygon which covers the locations of all objects stored in that context server. The federation service uses this information to allow inserting, querying, and modifying context data over different context servers. Value added services offer additional functionality for convenience with proprietary interfaces (e.g., a map service to generate maps, or a navigation service for routing).
- Context-aware applications are located at the *application tier* and use the underlying services. Within the Nexus project, several context aware applications have been developed, like NexusScout, an advanced location based information system [15], NexusRallye, an explorative mobile game [16], or prototypes within the Smart Factory scenario where context awareness is applied to a production environment [17].

We think of the Nexus platform as a spatial analogon to the World Wide Web, i.e., like web servers, it is easy to set up and operate independent context servers with their own data sets within our platform.

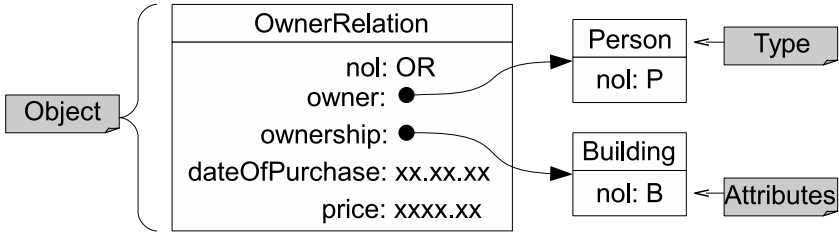


Fig. 3. Reference between a building and its owner

In such a scenario, there are various ways to use references between objects, e.g., references between buildings and their owners (Fig. 3) as well as topological relationships between objects like *overlaps* or *inside*. With different persons or organizations developing their own context models, multiple representations of the same real world objects are likely. To give applications the chance to get all possible information about a certain object, we use references between its multiple representations.

There are approaches to find multiple representations in geographic data sets automatically, e.g., for street data [18]. Figure 4 shows an example for multiple representations of nodes and edges in a road network, given by two different data sets. E.g., the nodes 2_A and 2_B of the two data sets are similar enough to interpret them as a multiple representation of the same real world junction object, in consideration of their geographic position and distance as well as the number of associated edges, so a multiple representation reference object *MR* will be introduced.

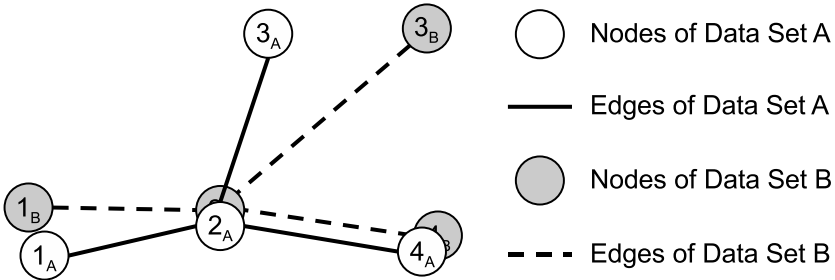


Fig. 4. Multiple representations in a road network

For modeling these references, we use the so-called Nexus Object Locator (NOL), a global identifier of every object within the Nexus platform. Reference object types in the AWM have attributes with NOL values, as shown in Fig. 5. The NOL consists of two parts, a globally unique object identifier and the address of the context server storing this object. The address allows Nexus components

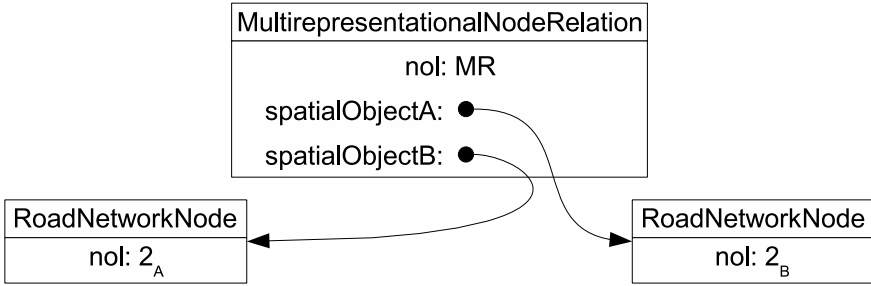


Fig. 5. A reference object and its referenced objects

to retrieve a complete object when they know its NOL. Without additional means, references can only be traversed in one direction, i.e. in Fig. 5 from the `MultirepresentationalNodeRelation` to the `RoadNetworkNodes`.

AWQL includes simple predicates to express queries, but no complex constructs like joins. Therefore the described references cannot be interpreted directly in the platform so far. We plan to introduce an operator in AWQL, so that referenced and referencing objects are included in result sets as well as the actual queried objects. The query results should correspond to our best effort principle, i.e., if the result set would contain an object that has a dangling reference, the object should be returned. For example, a query for objects of the owner relation (Fig. 3) should return all owner relation objects, no matter whether referenced person objects or building objects are currently available or not, because the additional information in the attributes of the owner relation objects should not be lost. This only holds if no data from a building or person object is required.

For the following discussion, we give a clearer distinction of the terms data provider and context server. Context server refers to an infrastructure component, a computer storing Nexus objects and providing them to the federation tier. Data providers are organizations, e.g., companies or municipalities, owning one or more context servers, which they use to host the objects they want to provide to the Nexus system. For the remainder of this paper, we will mostly use the term context server, but of course, the requirement of context server autonomy results from the fact that two different context servers may be owned by two different, autonomous organizations.

4 Referential Integrity

As stated in Sect. 1, there are two main challenges to address in reference management. In this section, we discuss our approach to handle referential inconsistencies. In the next section, we present our approach for enabling bi-directional traversal of references.

In this section, we will refer to a simplified example shown in Fig. 6. A context server S_1 stores an object of type O with the object ID o_1 . The context server

S_2 , owned by a different data provider, stores an object of type R (e.g., MR in Fig. 5) with the object ID o_2 and an attribute r referencing o_1 . Also for simplicity, we will use two general query classes, one for objects of a given type (O or R query type), and one for joins based on a reference attribute ($R * O$ query type). Note that although most practical applications of references are more complex than this example—the applications mentioned in Sect. 3 require two referenced objects—the underlying concepts for managing the references is the same, only the number of references increases. Here we decided to use this primitive example for our further explanations and discussions.

There are two major difficulties in ensuring referential integrity of the data stored by Nexus’ context servers. Databases typically enforce referential integrity of the data by referential constraints. They can either reject changes to the data (ON DELETE RESTRICT, ...) or trigger additional changes to restore a consistent state (ON DELETE CASCADE, ...). Regarding the scenario in Fig. 6, the former would forbid S_1 to delete o_1 , while the latter would force S_2 to delete or alter o_2 when S_1 deletes o_1 . Both variants of constraints violate the principle of autonomous context servers.

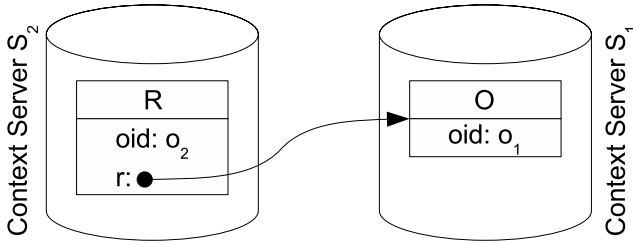


Fig. 6. A relation referencing an object of a different context server

The second difficulty is that in a large-scale distributed information system, we cannot assume that in the normal case the data of every server is accessible. Typically, there will be several context servers which are temporarily or permanently inaccessible, e.g., because of system failures or network partitioning, without obeying some sign off protocol, which would the system allow to restore a consistent state. Following the best effort principle, a query in this case should not fail, but instead generate a result set based on the data currently available.

For these reasons, the approach of continuously maintaining referential integrity of the complete data set is unfeasible for Nexus. Instead, we accept (temporary) referential inconsistencies in the data and leave the decision on how and when inconsistencies are resolved to the data providers. The best way to do this depends on the exact reason for the inconsistency and the kind of data. If the inconsistency is caused by a temporary failure of S_1 , no action is required. If o_1 has been permanently deleted, the action depends on the value of o_2 for its owner. For objects, e.g., consisting merely of references and being generated automatically by some kind of crawler analyzing objects of the Augmented World, the owner will probably use

some data cleaning algorithm to automatically remove o_2 . If o_2 contains additional valuable attributes, the owner may decide to only remove or change the reference.

Although we cannot guarantee consistency for the data stored, we want to make guarantees for the query result. Defining referential integrity for query result sets is not as straightforward as for the complete data set. Referring to Fig. 6, an application may query for only object o_2 . Although the complete data set is consistent, the result set would be inconsistent in the sense that the object referenced by $o_2.r$ is not included in the result set.

As outlined in Sect. 2, a solution based on repairs of the inconsistent database has already been proposed in [8] and [9]. Unfortunately, applying this solution to our scenario would violate the best effort principle. When o_1 is deleted, there are basically two different repairs. The first is to remove o_2 also, the second is to add an O , a virtual object created just for conveying the data into a consistent state². The consistent query result is defined as the intersection of the results of the query evaluated against all repairs. Thus, a query result can never contain o_2 , as it is not included in the result of querying the first repair (the one removing o_2) of the database.

Therefore, we select an approach based on the idea of possible answers in [9]. Instead of taking all repairs into account, we choose one repair depending on the query. A query requesting R s retrieves o_2 (the repair chosen is to assume the existence of an O), while a query for $R * O$ returns the empty set (the repair chosen is to delete o_2). This satisfies the best effort principle while avoiding to include (randomly) generated objects in the result sets. Note that when using two subsequent queries, first for R s and subsequently for the referenced O s, one actually sees the inconsistency. However, as the Nexus system does not provide isolation, this can happen even if we were able to continuously maintain integrity in the data.

Table 1 summarizes the query results for our best effort approach on the left and the query results according to [8] and [9] on the right. In terms of the relational algebra, in our approach the O query would roughly correspond to $\pi_O(R \bowtie O)$, the R query to $\pi_R(R \bowtie O)$ and the $R * O$ query to $R \bowtie O$.

5 Locating Objects

The second problem in managing references is the bi-directional traversal of references. In this section, we present our return references approach, the commit protocol used for managing return references and the results of an experiment measuring the additional overhead caused by the management of return references.

5.1 Return References

To be able to process queries of the $R * O$ type, two different access paths are necessary. The first one is used for locating object o_1 when object o_2 is given. As

² As we do not have a finite domain, there is in fact an infinite number of possible virtual O s.

Table 1. Query results

data	query results					
	best effort			8 and 9		
	<i>O</i>	<i>R</i>	<i>R * O</i>	<i>O</i>	<i>R</i>	<i>R * O</i>
\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
$\{o_1\}$	$\{o_1\}$	\emptyset	\emptyset	$\{o_1\}$	\emptyset	\emptyset
$\{o_2\}$	\emptyset	$\{o_2\}$	\emptyset	\emptyset	\emptyset	\emptyset
$\{o_1, o_2\}$	$\{o_1\}$	$\{o_2\}$	$\{o_1, o_2\}$	$\{o_1\}$	$\{o_2\}$	$\{o_1, o_2\}$

the object ID o_1 is the value of the attribute $o_2.r$, this is a primary key lookup. The second access path is the inversion—locating object o_2 when object o_1 is given—i.e., it retrieves all objects o where $o.r = o_1$.

Primary key lookups are an important capability of information systems, and Nexus already supports them by NOLs (cf. Sect. 3). The return references approach incorporates this concept, thus providing the first access path.

To implement the second access path, each context server maintains an additional table as shown for S_1 in Fig. 7. For each reference to an object stored by S_1 , this table contains an entry with the NOL of the referencing object and the ID of the referenced object. This information is only used internally by the system and never included in result sets delivered to applications.

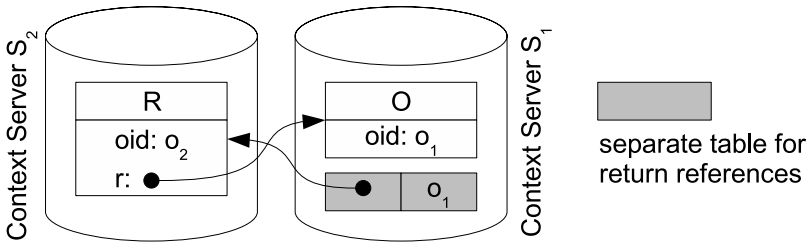


Fig. 7. Context servers using return references

Using return references can also cause referential inconsistencies when the referencing object (o_2 in Fig. 7) is deleted. Because return references do not appear in result sets, no extra procedure is required to hide them from the users, and the inner join used for $R * O$ queries prevents objects from being wrongly included in result sets because of inconsistent return references. As for the inconsistencies described in Sect. 4, the decision on when and how to resolve inconsistencies in return references is up to the data providers.

NOLs and return references provide an efficient means for managing links between objects—the respective context server can be located in constant time—at the cost of flexibility. As the context server’s address is part of every NOL and

every return reference, transferring an object from one context server to another is rather complicated. But, this is considered a very seldom needed operation.

5.2 Distributed Commit Protocol

When a context server inserts or deletes an object containing references to other objects, a protocol is required to instruct the context servers hosting the referenced objects to create or delete the corresponding return references. As we have to cope with inconsistent references and return references anyway, the protocol does not have to guarantee not to create any inconsistencies, but it should try to avoid them. However, for ensuring the correct result for the $R*O$ query type, it is required that for every pair of referencing and referenced object a return reference is inserted. We use a modification of the 2PC protocol tailored to these requirements, which tolerates server failures or network partitioning and reduces the overhead compared to fault tolerant extensions of the 2PC protocol. In the following, we will focus on the return references approach, where the context server inserting the referencing object acts as the coordinator and the context servers hosting the referenced objects are the cohorts.

For inserting an object o referencing a set of other objects o_1, \dots, o_k stored at S_1, \dots, S_k the requirements are:

- If the insertion of o succeeds, the return references at S_1, \dots, S_k *must* be present. This ensures that, given an o_i , o can always be retrieved.
- If at least one of o_1, \dots, o_k is not present, the insertion of o *should* fail.
- If the insertion of o fails, none of the return references at S_1, \dots, S_k *should* be present.

The algorithms for the coordinator and the cohorts work as follows:

Coordinator algorithm

- 1: begin local transaction
- 2: insert o
- 3: send requests for insertion of return references to all S_i s
- 4: **repeat**
- 5: wait for reply or timeout
- 6: **until** k success or one failure message received or timeout exceeded
- 7: **if** k success messages received **then**
- 8: send commit messages to all S_i s
- 9: local commit
- 10: **else** ▷ at least one S_i failed or did not reply, coordinator aborts
- 11: send abort messages to all S_i s
- 12: local abort
- 13: **end if**

Cohort algorithm

- 1: receive request for inserting return reference
- 2: **if** referenced object present and insertion of return reference succeeds **then**

```

3:  reply success
4:  wait for commit or abort request or timeout
5:  if abort then
6:      delete previously inserted return reference
7:  else      ▷ the coordinator succeeded or did not reply, cohort commits
8:  end if
9: else
10:  reply failure
11: end if

```

According to the requirements, the coordinator follows an ‘if in doubt, abort’ concept (Line 10), while the cohorts do an ‘if in doubt, commit’ (Line 7) to ensure that the first requirement is fulfilled. The cohort locally commits the insert of a return reference in Line 2 and uses a compensate operation in Line 6 when the coordinator aborts, to prevent the database from rolling back the insert operation when the cohort crashes at Line 4.

The requirement for deleting o is that all corresponding return references at the S_i s *should* be deleted. For this, no special commit protocol is required, the coordinator just deletes o and sends delete messages to the cohorts.

5.3 Experimental Evaluation

For the experimental evaluation, we used two different servers, both equipped with dual UltraSPARC III 1.2GHz CPUs and 6GB resp. 8GB of RAM. Both have a local DB2 V8 installation. As the commit protocol tolerates network failures, we can use UDP messages for the reference management. On one server, we inserted object sets containing a varying number of objects, each one referencing two objects stored on the other server. Figure 8 shows the runtime for the insert operation against the number of objects inserted. To assess the extra overhead caused by the reference management, i.e., the communication between the two

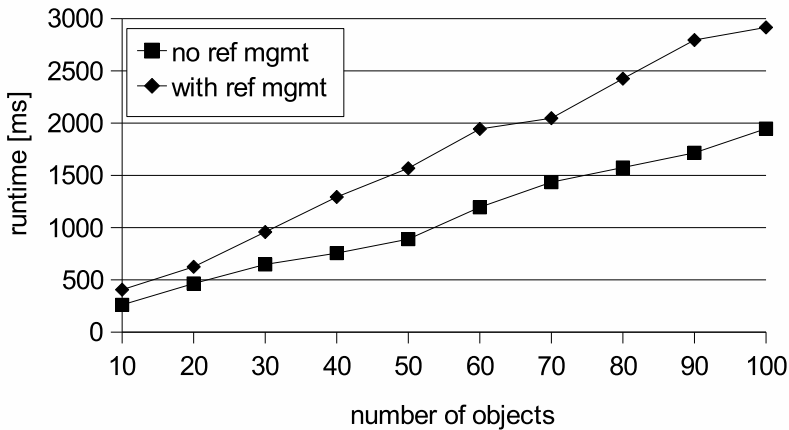


Fig. 8. Runtime overhead for insert operations caused by reference management

servers and insertions of return references, the diagram also contains the runtime of the same insert operation without using reference management. For the data sets we used for the evaluation, the additional overhead is between 34% and 76%. We expect the additional overhead for a typical mix of queries and inserts to be considerably lower, because inserts are less frequent than queries and only a small percentage of objects contains references. The additional overhead is acceptable given the benefit of the additional functionality provided by the return references. For example, queries retrieving multiple representations of a given object modeled as in Fig. 5 cannot be processed at all without return references.

6 DHT as a Distributed Global Index

An alternative approach to return references is using some sort of distributed global index, e.g., a storage system built on top of a distributed hash table (DHT). DHTs are large tables of key-value pairs, distributed among many hosts, which allow the retrieval of the value for a given key in typically $\log n$ steps. This approach not only provides the functionality of return references, but also replaces the locator part of NOLs, so that in all places where Nexus currently uses NOLs, plain object IDs (i.e. NOLs without server part) are sufficient.

The most basic variant of this approach is shown in Fig. 9. The object IDs serve as keys in the global index, the associated values are sets containing the context server hosting the object and the IDs of all objects referencing this object. Locating object o_1 given object o_2 requires one lookup in the index, typically at the complexity $O(\log n)$, where n is the total number of objects in the Nexus system. The reverse operation—locating object o_2 given object o_1 —requires two index lookups. Compared to return references, queries and inserts are more expensive with the global index approach, but it has the advantage that object migrations between context servers only requires the update of a single index entry.

Some DHT storage systems are unsuitable for implementing an updatable index as required in our scenario. PAST, the storage system built on top of Pastry, regards its entries as read-only content [19]. Consequently, the keys are not provided by users but calculated by PAST as a hash value of the entry. As PAST

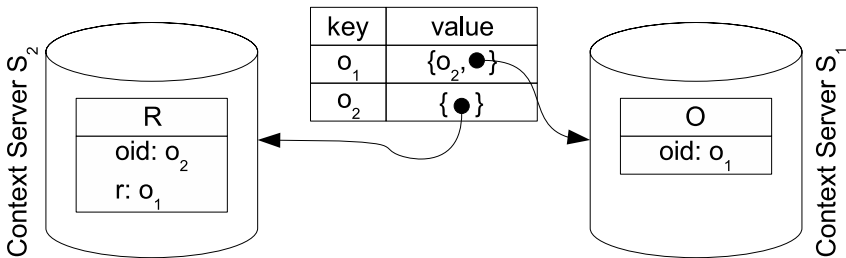


Fig. 9. Global distributed index

replicates entries to increase the availability, extending the implementation to support updates is complicated.

OpenDHT [20] is a more feasible alternative with respect to our requirements. It allows the applications to specify the keys. OpenDHT does not provide an update function, but this functionality can be emulated by remove and put operations. This can cause lost updates when several context servers concurrently modify the same entry while using an index structure as in Fig. 9. This problem can be solved by exploiting the feature of OpenDHT that keys do not have to be unique, so instead of storing $(o_1, \{o_2, S_1\})$, we store $\{(o_1, o_2), (o_1, S_1)\}$. As the context server storing o_1 is the only one to modify the entry (o_1, S_1) and S_2 is the only one to modify the entry (o_1, o_2) or (o_2, S_2) , no concurrent updates arise.

The commit protocol presented in Sect. 5.2 can also be used for the DHT approach, in this case the DHT is the cohort.

Because object migrations are rare and with the DHT approach, the processing of queries is more expensive than with return references, we focus on the return references approach.

7 Conclusion

In this paper, we discussed the problem of managing references in a distributed information system providing a high level of autonomy of the participating data providers. This problem is twofold: First, a concept of referential integrity is needed, which maintains the autonomy of the data providers while providing a consistent view on the data for the users. Second, the infrastructure has to support bi-directional traversal of references, i.e. finding the referenced object given the referencing object and finding the referencing object given the referenced object.

For the first problem, we presented a concept based on consistent query answers, i.e., we accept referential inconsistencies in the data but ensure consistent query answers, for which we adopted a special definition of referential integrity. For the second problem, we developed an extension for the context servers of the Nexus system, which for each reference contained in an object inserts a return reference at the context server hosting the referenced object. For inserting return references, the context servers follow a modified 2PC protocol, which guarantees that for every pair of referencing and referenced object a return reference is inserted even if a context server fails during the insert operation. The concepts we developed can be useful beyond the Nexus system, e.g., they can be used to implement a link service for the WWW.

We measured the time consumed by insert operations using the extended context server and compared it to the original context server without reference management. The additional overhead caused by the reference management component is below 80%, which seems reasonable given the fact that the original insert operation does not involve any network communication. So return references add an important functionality to the Nexus system at affordable costs. As we assume that object migrations are rare, the return references approach is also superior to the approach using a DHT.

Acknowledgments. This work was funded by the Center of Collaborative Studies *Nexus: Spatial World Models for Context-aware Applications* (Deutsche Forschungsgemeinschaft (DFG) grant SFB 627).

References

1. DeRose, S., Maler, E., Orchard, D.: XML Linking Language (XLink) Version 1.0 (2001), <http://www.w3.org/TR/2001/REC-xlink-20010627/>
2. Grossmann, M., Bauer, M., Hönle, N., Käppeler, U.P., Nicklas, D., Schwarz, T.: Efficiently managing context information for large-scale scenarios. In: 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005), pp. 331–340 (2005)
3. Tatarinov, I., Ives, Z.G., Madhavan, J., Halevy, A.Y., Suci, D., Dalvi, N.N., Dong, X., Kadiyska, Y., Miklau, G., Mork, P.: The Piazza peer data management project. SIGMOD Record 32(3), 47–52 (2003)
4. Chen, Y., Davidson, S.B., Zheng, Y.: Constraints preserving schema mapping from XML to relations. In: Fernandez, M.F., Papakonstantinou, Y. (eds.) Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002), pp. 7–12 (2002)
5. Arenas, M., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R.J., Mylopoulos, J.: The hyperion project: from data integration to data coordination. SIGMOD Record 32(3), 53–58 (2003)
6. Rusinkiewicz, M., Sheth, A.P., Karabatis, G.: Specifying interdatabase dependencies in a multidatabase environment. IEEE Computer 24(12), 46–53 (1991)
7. Gupta, A., Widom, J.: Local verification of global integrity constraints in distributed databases. In: Buneman, P., Jajodia, S. (eds.) Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 49–58. ACM Press, New York (1993)
8. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 68–79. ACM Press, New York (1999)
9. Fuxman, A., Fazli, E., Miller, R.J.: Conquer: Efficient management of inconsistent databases. In: Özcan, F. (ed.) Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 155–166. ACM, New York (2005)
10. Davis, H.C.: Referential integrity of links in open hypermedia systems. In: HYPERTEXT 1998. Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia: Links, Objects, Time and Space – Structure in Hypermedia Systems, pp. 207–216. ACM, New York (1998)
11. Ciancarini, P., Folli, F., Rossi, D., Vitali, F.: XLinkProxy: external linkbases with XLink. In: Proceedings of the 2002 ACM Symposium on Document Engineering, pp. 57–65. ACM, New York (2002)
12. Rizk, A., Sutcliffe, D.C.: Distributed link service in the aquarelle project. In: Bernstein, M., Carr, L., Østerbye, K. (eds.) Hypertext 1997, The Eighth ACM Conference on Hypertext, pp. 208–209. ACM, New York (1997)
13. Nicklas, D., Mitschang, B.: On building location aware applications using an open platform based on the Nexus Augmented World Model. Software and System Modeling 3(4), 303–313 (2004)

14. Hönle, N., Käppeler, U.P., Nicklas, D., Schwarz, T., Grossmann, M.: Benefits of integrating meta data into a context model. In: 3rd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops), pp. 25–29. IEEE Computer Society, Los Alamitos (2005)
15. Nicklas, D., Grossmann, M., Schwarz, T.: NexusScout: An advanced location-based application on a distributed, open mediation platform. In: VLDB 2003, pp. 1089–1092. Morgan Kaufmann, San Francisco (2003)
16. Nicklas, D., Hönle, N., Moltenbrey, M., Mitschang, B.: Design and implementation issues for explorative location-based applications: The NexusRallye. In: Iochpe, C., Câmara, G. (eds.) VI Brazilian Symposium on Geoinformatics, pp. 167–181. INPE (2004)
17. Westkämper, E., Jendoubi, L., Eissele, M., Ertl, T.: Smart factory—bridging the gap between digital planning and reality. In: Proceedings of the 38th CIRP International Seminar on Manufacturing Systems, pp. 44. CIRP (2005)
18. Volz, S.: An iterative approach for matching multiple representations of street data. In: Hampe, M., Sester, M., Harrie, L. (eds.) Proceedings of the JOINT IS-PRS Workshop on Multiple Representations and Interoperability of Spatial Data, Hannover, pp. 101–110 (self-published, 2006)
19. Druschel, P., Rowstron, A.I.T.: PAST: A large-scale, persistent peer-to-peer storage utility. In: Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, pp. 75–80. IEEE Computer Society, Los Alamitos (2001)
20. Rhea, S.C., Godfrey, B., Karp, B., Kubiawicz, J., Ratnasamy, S., Shenker, S., Stoica, I., Yu, H.: OpenDHT: a public DHT service and its uses. In: Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 73–84. ACM, New York (2005)

On Top- k Search with No Random Access Using Small Memory

Peter Gurský¹ and Peter Vojtáš²

¹ University of P.J.Šafárik, Košice, Slovakia

² Charles University, Prague, Czech Republic

`peter.gursky@upjs.sk`, `peter.vojtas@mff.cuni.cz`

Abstract. Methods of top- k search with no random access can be used to find k best objects using sorted lists of attributes that can be read only by sorted access. Such methods usually need to work with a large number of candidates during the computation. In this paper we propose new methods of no random access top- k search that can be used to compute k best objects using small memory. We present results of experiments showing improvement in speed depending on ratio of memory size and data size. Our system outperforms other also when the total number of attributes is much bigger than number of query attributes (varying with user).

Keywords: Top- k search, no random access, TA-sorted variants, data much bigger than memory, user query.

1 Introduction

Top- k search became popular with growing use of the web services and increasing datasets sizes. Users are usually interested in few best objects rather than a big list of objects as a result. Top- k algorithms usually follow two main goals. Firstly, they minimize the number of source data to be processed i.e. they find the correct top- k objects using only a part of data. Secondly, the algorithms try to minimize the number of accesses to the sources and the computation time as well.

1.1 Related Work

The main stream of top- k search algorithms [1,2,3,4,5,6,7,10,13,14] consider to have objects with m attributes and to have values of each attribute stored in an independent structure. Each structure can be placed on a remote computer. Algorithms consider two types of access to the structures: sorted access and random access. Using sorted access we obtain pairs of object identifier and attribute value in sorted order, ordered by the attribute value (or more sophisticated - ordered by user preference to the attribute values). Random access allows us to obtain concrete attribute value for a specified object identifier. The *Best position algorithm* [1] works with the objects' positions in the ordered list in addition. All

these algorithms compute the overall object value using a monotone combination function.

According to the discussion in [6], the mentioned algorithms can be divided into following two groups. The first group uses both sorted and random access to the source structures, the second group uses the sorted access only. In this paper we concentrate on solutions using sorted access only. These algorithms are useful, for example, when the sources are streams of data or the incremental results of other algorithms/programs. We will show a significant improvement against previous no random access approaches.

The first algorithm using only the sorted access was *Stream-combine* [10]. The *NRA (No random access) algorithm* [6] presents stronger stop condition. Both approaches are time consuming and require a lot of memory in comparison to the size of the source data.

Other top- k search algorithms use rich indexation and multidimensional search [21,22] or prepared ranked views [12] to support arbitrary combination functions.

Another possible view of recent research distinguishes: one branch of research generalizes types of data to uncertain (see e.g. [18,17]) or to XML data (see e.g. [19]). The other branch focuses on top- k query optimization. This is the way we are going to contribute too.

1.2 Motivation

Our motivation is in an infrastructure serving top- k querying for different users over web extracted and/or retrieved data (it is a part of our research in the project NAZOU (see <http://nazou.fiit.stuba.sk>)). Our motivating example deals with a job seeking user. One would select a job according to job offer (attributes such as salary, location, job category, position type, employee experience, desired education level etc), another would like to have information about pollution in the city and pollen calendar. Maybe a part of the salary is paid in cash and a part in mutual funds, then it would be interesting to know some parameters of this part of salary, e.g. trend analysis. If the jobs assumes moving to another place, one would like to know the situation on real estate market with a variety of attributes. Such data can consist of several dozens of attributes (in our example it is about 50 to 60 attributes) and can be downloaded from the web and/or extracted from the web. Some domains of attributes have a complicated structure (points in a metric space, hierarchical tree of a graph). For these types of domains the suitability of the attribute values to a user is a complex problem, that can be solved effectively using a stream producing program. A typical user's job search is based on a small part of these data, mostly on 5 to 7 attributes. Nevertheless, our system should be ready to serve different users using different sets of attributes.

As a result of this, our interest is in the study of top- k querying over big number of data (exceeding the size of memory) extracted from the web and stored indexed in a big number of structures.

1.3 Main Contribution

In this paper we present a novel view on methods of top- k search with *NRA* - No Random Access (aka. *TA*-sorted variants) focusing on different users querying small number of attributes from a big number of web extracted data in a large number of attributes.

- Our first contribution includes three novel versions of *NRA* algorithm - the *3P-NRA* algorithm and its 2 variants using small memory and organizing overflow data on disk by hash partitioning and B+tree.
- Next contribution is own implementation of these methods with light weight indexes. The point is, that managing a big number of indexes in a transactional database can be very costly and that a database overhead can influence results substantially. We consider only few insertions and deletions (no update) in comparison to the number of queries (read only), so we do not need the full power of a RDBMS.
- Last contribution are extensive experiments in different directions. We measure time efficiency of several top- k algorithms from table scan of joined data to several variants of our new *3P-NRA* algorithm with different ratio of memory and data. As we show in section 5 the *3P-NRA* algorithm outperforms the previous *NRA* algorithm significantly. The algorithm *3P-NRA with hash partitioning* with very low memory/data ratio works well with only a small time increase in comparison to the fastest *3P-NRA*.

2 3P-NRA Algorithm

In this section we present the modification of *NRA* algorithm [6] named *3P-NRA* (3-phased no random access) proved to be better than *NRA*. The first version of our *3P-NRA* algorithm was presented in the NAZOU project proceedings [8].

In the rest of this paper we use the following notation. Value m represents the number of attributes of objects or the number of sources. An arbitrary object is denoted as $X = (x_1, \dots, x_m)$, where x_1, \dots, x_m are called scoring values. Each x_i represents a user preference to the real value of the i -th attribute of X . Scoring values are numbers from interval $[0, 1]$, where 1 means strong preference to the real value and 0 represents the unwanted real value of the attribute. Let $V(X) = \{i_1, \dots, i_n\} \in \{1, \dots, m\}$ be a subset of known attributes x_{i_1}, \dots, x_{i_n} of X , we define $W_V(X)$ (or shortly $W(X)$ if V is known from context) to be minimal (worst) possible value of the combination function F for the object X . We assume that F is a monotone combination function. We compute $W_V(X)$ such that we substitute each missing attribute by 0. For example if $V(X) = \{1, \dots, g\}$ then $W_V(X) = F(x_1, \dots, x_g, 0, \dots, 0)$.

Analogously we define maximal (best) possible value of the combination function F for object X as $B_V(X)$ (or shortly $B(X)$ if V is known from context). Since we know that sorted access returns values in descending order, we can substitute for each missing value the corresponding value from the vector

$u = (u_1, \dots, u_m)$, where u_1, \dots, u_m are the last scoring values seen from each source. For example if $V(X) = 1, \dots, g$ then $B_V(X) = F(x_1, \dots, x_g, u_{g+1}, \dots, u_m)$.

The real value of object X is $W(X) \leq F(x_1, \dots, x_m) \leq B(X)$. Note that unseen objects during the computation (no values are known) has $B(X) = F(u_1, \dots, u_m)$. The value $\tau = F(u_1, \dots, u_m)$ is well known as the threshold value.

In algorithms we use the top- k list T ordered by the worst value. The object in T with the smallest worst value is labeled T_k . In the (unordered) set C we store the candidates with the worst value smaller or equal to $W(T_k)$ but with the best value larger than $W(T_k)$. These are the objects with a hope to get into T later. We call the objects in C as candidates.

We implemented C as a hash table with object identifiers as a key. To recognize the sources with all known values between objects in $T \cup C$ we maintain the number of missing values for each source.

For the comparison with our 3P-NRA algorithm we present the original NRA [6](#) in our notation first:

Input: k, F, m sources

Output: k ranked objects if exist

$T = \emptyset, C = \emptyset$

Repeat the following:

Do the sorted access in parallel to all sources.

For every object X seen under sorted access do

Compute $W(X)$ and $B(X)$

If $|T| < k$ then put X to T

Else If $W(X) > W(T_k)$ then

If $X \notin T$ move T_k to C , put X to T

Else put X to C

For every object $Y \in C, Y \neq X$ compute $B(Y)$

If Y is no more relevant (i.e. $B(Y) \leq W(T_k)$)

remove Y from C

If $|C| = 0$ return T and exit;

The most time consuming part of *NRA* algorithm is the last cycle. In the following *3P-NRA* algorithm we drastically decrease the number of best value computation, moreover we take the unknown values of the candidates into account.

Input: k, F, m sources

Output: k ranked objects if exist

$T = \emptyset, C = \emptyset$

Phase 1

Do the sorted access in parallel to all sources.

```

For every object  $X$  seen under sorted access compute  $W(X)$  and do
  If  $|T| < k$  then put  $X$  to  $T$ 
  Else If  $W(X) > W(T_k)$  then
    If  $X \notin T$  move  $T_k$  to  $C$ , put  $X$  to  $T$ 
    Else put  $X$  to  $C$ 
If  $W(T_k) \geq$  threshold  $\tau$  goto Phase 2
repeat Phase 1
Phase 2
Do the sorted access in parallel to the sources for which there
are unknown values for objects in  $C$  and  $T$ .
For every object  $X$  seen under sorted access do
  If  $X \notin T \cup C$  ignore it
  Else If  $B(X) \leq W(T_k)$  remove  $X$  from  $C$ 
    If  $|C| = 0$  return  $T$  and exit
    If  $W(X) > W(T_k)$  and  $X \notin T$  move  $T_k$  to  $C$ , put  $X$  to  $T$ 
If  $(W(T_k)$  increased) OR (threshold  $\tau$  decreased) then
  heuristic  $H$  can choose to go to Phase 3
repeat Phase 2
Phase 3
For every object  $X \in C$  compute  $B(X)$ ;
  If  $X$  is no more relevant ( $B(X) \leq W(T_k)$ ) remove  $X$  from  $C$ 
If  $|C| = 0$  return  $T$  and exit; otherwise goto Phase 2.

```

Phase 1 works similarly to the standard *NRA* algorithm with the exception of the threshold test and the computation of best values. The best values do not need to be computed in Phase 1 because each object X seen in Phase 1 has its best value bigger than $W(T_k)$ and no pruning of the candidates can be done.

The heuristic H in *3P-NRA* algorithm can be used to skip an expensive computation of Phase 3. On the other hand, if H always chooses to do the Phase 3 the *3P-NRA* algorithm is proved to be instance optimal. The instance optimality of *NRA* means, that if *NRA* finds top k objects using y sorted accesses, then there are no algorithms reading the sources only by sorted access, that can find top k objects using less than $\frac{y}{m}$ sorted accesses (see [6]).

Theorem 1. *Let F to be a monotone combination function, then algorithm 3P-NRA correctly finds top- k objects.*

Proof. Let $S(X) = F(x_1, \dots, x_m)$. Assume that *3P-NRA* algorithm ended its computation at position $z = (z_1, \dots, z_m)$ (i.e. algorithm did z_i sorted accesses to the i -th source) and returned objects X_1, \dots, X_k . Let $X \notin \{X_1, \dots, X_k\}$. We will show that $B(X) \leq W(T_k)$.

Note that the $W(T_k)$ cannot decrease during the computation and in the whole algorithm we removed from C only the objects with best value smaller or equal to actual $W(T_k)$. Thus for each such object X holds that its best value is smaller or equal to the $W(T_k)$ at the end of the computation, more formally for every removed object X from C holds that $S(X) \leq B(X) \leq W(T_k)$.

Next we need to consider the objects ignored in Phase 2. These are the objects not seen in Phase 1. Let X is such object with scores x_1, \dots, x_m and let v_1, \dots, v_m are equal to values u_1, \dots, u_m at the end of Phase 1. It holds that $x_i \leq v_i$ for each $i \in \{1, \dots, m\}$. From the monotonicity of combination function we can see that $S(X) = F(x_1, \dots, x_m) \leq B(X) = F(v_1, \dots, v_m)$. The last equality goes from the fact that X was not seen in Phase 1. From the condition at the end of Phase 1 we know that $W(X_j) \geq F(v_1, \dots, v_m)$ for each $j \in \{1, \dots, k\}$ thus $B(X) \leq W(T_k)$.

Note that the fact that we can skip the sources where all attributes between objects in T and C follows from the observations that we can ignore all unseen objects. From such sources we cannot retrieve interesting objects anymore. \square

Theorem 2. *Let F to be a monotone combination function. If heuristic H always chooses to go to Phase 3, then algorithm 3P-NRA makes at most the same number of sorted accesses as NRA algorithm i.e. 3P-NRA algorithm is instance optimal.*

Proof. We can see that Phase 1 access the sources in the same way as NRA [6] algorithm as well as Phase 2 + Phase 3 together unless the skipped sorted accesses to useless sources.

All we need to show is that algorithm *NRA* cannot stop earlier than the Phase 1 does. Algorithm *NRA* ends when there are no more relevant objects out of the list T i.e. $B(X) \leq W(T_k)$ for all $X \notin T$. Phase 1 ends when $F(u_1, \dots, u_m) \leq W(T_k)$. Observe that $F(u_1, \dots, u_m)$ is the best value for all unseen objects. We need to wait until $F(u_1, \dots, u_m)$ decreases to have $W(T_k) \geq F(u_1, \dots, u_m)$ because we have to be sure that there is no object with the best value greater than $W(T_k)$. \square

The improvements of the 3P-NRA algorithm in contrast to *NRA* [6] are the following:

- New objects are considered in phase 1 only. Other objects are ignored.
- Many computations of the best values are omitted.
- After acquisition of all unknown values of any attribute between objects in $T \cup C$ the algorithm stops the work with the corresponding source (no more sorted accesses to the source will be done). This feature decreases the number of disk accesses significantly.
- A good choice of heuristic H can yield a massive speedup of the algorithm, however it can slightly increase the number of disk accesses according to H .

In our tests in section 5 we use the heuristic that goes to phase 3 only each 1000th loop of phase 2. In the tests we show orders of magnitude speedup against algorithms without heuristic.

3 3P-NRA with B+Tree

Analyzing the 3P-NRA algorithm, we can see that the maximal number of candidates in C is at the end of phase 1. After that moment, the algorithm ignores

all unseen objects and the number of candidates decreases until $C = \emptyset$. In the worst case the size of C can reach the number of all objects [6]. Although it is an unusual situation, in practice the size of C can easily reach one half of the source data. If the source data are big in comparison to the available memory, we need to consider putting a part of candidates on disk. We will call them the overflow candidates.

In the memory version of the $3P$ - NRA algorithm we maintain candidates in a hash table with the object identifier as a key. We need to access the hash table after every access to the sources to join the score value with the appropriate object or to identify that the object is not a candidate. The straightforward solution is to store the overflow candidates in a simple heap file, but the search of candidates will be highly ineffective with the growing size of the file.

To eliminate the number of disk accesses we implemented our version of B+tree with inner nodes maintained in memory and leaves stored on disk. Therefore we usually need 1 or 2 I/O with each sorted access: one to identify the object to join with and one I/O to store the joined object. The second I/O can be omitted in phase 2, if it is an unseen object or its best value is lower than $W(T_k)$. Time to time we need an extra I/O to split a leaf when the number of objects in the leaf overflows the page size. Seeing that only one instance of the algorithm accesses the B+tree, we skip the transactional features of standard relational database to improve the performance.

The algorithm works like $3P$ - NRA until the memory size is reached. Then we need to search list T followed by the search in the main-memory hash table for every pair of object identifier and score value from the sources to find the object to update. If the object is not found in memory we continue with the search in the B+tree (see Figure 1). The whole update procedure *after the size of hash table reaches the allowed maximum (max)* works as follows:

```

Let  $X$  be an object found by sorted access,  $C$  be the hash table
and  $B$  be the B+tree
If  $X \in T \cup C$  update  $X$  and exit to main procedure
If  $X \in B$ 
    If doing Phase 1 update  $X$  in  $B$ 
    If doing Phase 2
        If the size of  $C < max$  update  $X$ , move  $X$  to  $C$  and remove
         $X$  from  $B$ 
        Else update  $X$  in  $B$ 
Else If doing Phase 1 store  $X$  to  $B$ 
    If doing Phase 2 ignore  $X$ 

```

We need to do update using this procedure until the number of objects in B+tree decreases to zero.

In Phase 3 we process the data first from hash table and then from B+tree. As in the update procedure we prefer to move candidates from B+tree to hash table if there is a space. The more scarcely the heuristic H in phase 2 does the

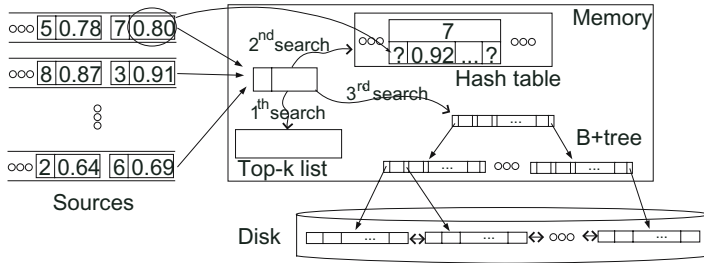


Fig. 1. Managing data in 3P-NRA with B+tree

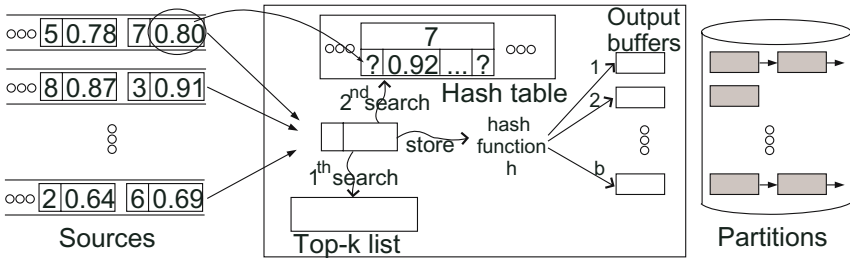


Fig. 2. Managing data in NRA with Hash partitioning

phase 3 the higher reduction of the candidates is. Instead of removing the objects one by one from B+tree it is usually faster to use bulk loading to create a new B+tree.

The main advantage of this method is that it can work with very small memory. All we need to preserve in memory is the top- k list and inner nodes of B+tree i.e. no hash table. The main disadvantage is its speed as we show in section 5. It is caused by many I/Os to disk.

4 3P-NRA with Hash Partitioning

The weak spot of the previous method is a high number of I/Os to the leaves of B+tree. To decrease the I/Os, the updating or joining of the objects need to be a less frequent process.

In this method we send the overflow candidates directly to the partitions as soon as they come from the sources. This principle is similar to the hash join technique [16]. We have to be aware of the time of matching the partitions. We still want to minimize the number of accesses to the sources instead of processing all data as in the standard hash join. We can use different heuristics (denoted as H_{disk}) to estimate the suitable moment of matching.

Algorithm *3P-NRA with hash partitioning* illustrated in Figure 2 works as follows:

Input: k , F , m sources, $M = \#$ pages needed to store all objects from the sources

$max = \#$ available pages without pages needed by T

$h =$ hash function for partitioning (different from hash function used in the memory hash table)

Output: k ranked objects if exist

$T = \emptyset$, $C = \emptyset$

$b = M/max$; //counting the number of partitions

Create b output buffers

$maxC = max - b$ //counting the number of pages for C

Phase 1

Do the sorted access in parallel to all sources.

If $size(C) < maxC$ do the Phase 1 as in 3P-NRA

Else

For every object X seen under sorted access do

If $X \in T \cup C$ then update X and compute $W(X)$

If $(X \notin T \text{ and } W(X) > W(T_k))$ move T_k to C and X to T

Else add X to the buffer page $h(X.id)$ //flushed as page fills

heuristic H_{disk} can choose to do the disk phase

If $W(T_k) \geq$ threshold τ goto Phase 2

repeat Phase 1

Phase 2

If there are no objects in partitions on disk

do the Phase 2 as in 3P-NRA

Else

For every object X seen under sorted access do

If $X \in T \cup C$ then update X and compute $W(X)$ and $B(X)$

If $B(X) \leq W(T_k)$ remove X from C

Else

If $(X \notin T \text{ and } W(X) > W(T_k))$ move T_k to C and X to T

Else add X to the buffer page $h(X.id)$ //flushed as page fills

heuristic H_{memory} can choose to do the Phase 3

heuristic H_{disk} can choose to do the Disk phase

repeat Phase 2

Phase 3

For every object $X \in C$ do

If X is no more relevant (i.e. $B(X) \leq W(T_k)$) remove X from C

If $|C| = 0$ and no objects in partitions on disk return T and exit; otherwise goto Phase 2.

Disk phase (illustrated in Figure 3):

Flush all output buffers to the partitions on disk.

Assign output buffers with the new partitions and use them to store candidates from C and flush them.

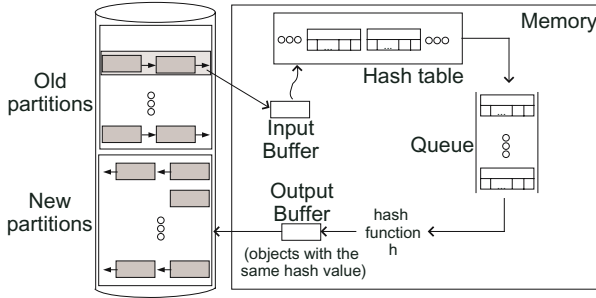


Fig. 3. Disk phase in NRA with Hash partitioning

Release the memory taken by the output buffers.

Queue $Q = \emptyset$

For each old partition P do

 Release as many objects from Q to new partitions as needed to allocate C such that $capacity(C) = size(P)$

 For each object X update or add X in C

 If doing Phase 2 then

 For every object $X \in C$ do

 If X is no more relevant ($B(X) \leq W(T_k)$) remove X from C

 Move every object from C to Q

If $(size(Q) + size(objects\ on\ disk) + size(output\ buffer)) \leq max$
 load all objects to C

Else

 release as many objects from Q to new partitions as needed to allocate C with size $maxC$ and then move the remaining objects from Q back to C

The minimum required amount of memory is $size(T) + \sqrt{f \cdot M}$, where M is the size needed to store all joined objects from the sources and f is a fudge factor used to capture the increase in size between the data in hash table and real size needed by hash table (usually $f = 1,3$). Because we want to maximize the memory available for the hash table we compute the number of partitions as $b = M/max$, thus the maximal size of the partition will be the size of whole available memory.

The heuristic H_{memory} is equivalent to the heuristic H in $3P-NRA$ algorithm. The main purpose of H_{memory} is to skip a large number of the best value computations. The good choice of the heuristic H_{disk} can decrease the number of Disk phases. Our effort is to reduce the number of disk phases to one or two and to keep the number of sorted accesses on the low level i.e. we want to use (much) less I/O than the hash join itself.

We have tried several versions of the heuristic H_{disk} . The heuristic, which we call the *Small buffer heuristic*, shows the best results and we use it also in

our experiments. Let $sizeP$ be the size of all partitions on disk together with the cumulative size of the output buffers. In the Disk phase we want to put all candidates from the partitions back to the memory to end the computation with standard $3P-NRA$ algorithm. We expect that a part of the data on disk has its best value worse than $W(T_k)$ and will be deleted. Suppose that objects in C can be stored in A pages, then the fill factor in C is $p = \frac{A}{maxC} \%$ i.e. the algorithm deleted $(100 - p)\%$ of data in C using phases 2 and 3. Our hypothesis is that the similar proportion of data can be deleted also from the data in partitions and we can expect that about $p \cdot sizeP$ will remain as candidates. We have $max - A$ pages to store the candidates from the disk partitions. Thus the *Small buffer heuristic* chooses to do the disk phase if:

$$max - A > p \cdot sizeP = \frac{A}{maxC} \cdot sizeP$$

5 Experiments

This section reports the experiments with our algorithms over artificial data. The experiments were conducted on a PC having Intel Pentium M 2GHz CPU and 512 MB RAM running on Windows XP. We minimized the number of other processes during the tests to keep a stable test environment. Algorithms are implemented in java. The size of disk pages was set to 4 kB. Memory restriction was obtained via input parameters of top- k search methods.

In our tests we used artificial data from 3 different datasets. Each dataset consists of 5 attributes x_1, \dots, x_5 . Each attribute has values in range $[0, 1]$. Dataset 1 and Dataset 2 were randomly generated with exponential and Gaussian distributions respectively. Attributes in Dataset 3 were generated as a concatenation of two Gaussian distributions with standard deviation 0.05 and means in 0.25 and 0.75 respectively. Presented results are average times from all datasets. Each dataset has different versions with sizes 1 MB, 10 MB, 100 MB and 1GB. In each experiment we used 5 sources and fixed combination function $F(X) = 3x_1 + 2x_2 + x_3 + 2x_4 + 2x_5$. From the database point of view we can say that we do a partial join of 5 "tables".

In the first experiment we compare the original NRA algorithm from [6], then $3P-NRA$ algorithm with the heuristic H choosing to do Phase 3 of the algorithm every 1000th loop of Phase 2 and finally the instance optimal version of $3P-NRA$ denoted as $3P-NRA_{io}$. We used 10 MB of input data and each algorithm had enough time to hold the candidates. We tested the time of the computation of different result size. Results are shown in table [1]. In table [2] there is the comparison of the same algorithms over 100 MB of data.

We can see that omitting the big number of phases 3 in $3P-NRA$ improves the computational time significantly. The advantage of $3P-NRA_{io}$ algorithm - ignoring the sources for which we have no missing values in $C \cup T$ - improves the computational time very slightly.

In the next experiment we fixed the data size to 100 MB and the number of retrieved objects (k) to 10. We compared our algorithms $3P-NRA$ with $B+tree$

Table 1. Computation time in milliseconds using different k (memory algorithms, data size 10MB)

k	1	5	10	20
<i>NRA</i>	275	497	653	899
<i>3P-NRA_{io}</i>	238	451	626	886
<i>3P-NRA</i>	69	115	117	148

Table 2. Computation time in seconds using different k (memory algorithms, data size 100MB)

k	1	5	10	20
<i>NRA</i>	160	242	416	675
<i>3P-NRA_{io}</i>	159	231	397	661
<i>3P-NRA</i>	5,47	7,51	7,72	9,28

and *3P-NRA with hash partitioning* with the use of restricted size of memory to store the candidates. Results are shown in table 3. The first column labeled $\frac{1}{1}$ means enough memory to hold all candidates. The last column labeled $\frac{1}{250}$ is undefined for *3P-NRA with hash partitioning* because the output buffers used for the partitioning need more space than the available memory in our implementation (theoretical minimum over 100 MB data with fudge factor 1, 3 of the memory hash table is approximately $\frac{1}{277} = 360$ kB).

Table 3. Computation time in seconds using different memory size (data size 100MB)

$\frac{\text{memory}}{\text{data size}}$	$\frac{1}{1}$	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{50}$	$\frac{1}{75}$	$\frac{1}{100}$	$\frac{1}{125}$	$\frac{1}{150}$	$\frac{1}{175}$	$\frac{1}{200}$	$\frac{1}{225}$	$\frac{1}{250}$
Hash partitioning	7,71	7,74	7,67	7,74	7,65	7,80	7,64	7,90	8,31	8,74	10,39	undef.
B+ tree	7,72	10,34	82,4	168	188	203	213	230	240	245	246	244

Our last experiment compares the algorithms over different data sizes. In table 4, we show the computation time of *3P-NRA* algorithm and both disk versions of the *3P-NRA* algorithm with $\frac{1}{50}$ of the available memory size in comparison to the size of the source data. We did not do the test with *3P-NRA* over 1GB data because of the small physical memory. The last row shows the computation over joined data from the sources in one table of MySQL with 6 columns: the first column holds the object identifiers and the other five the attribute scores. We observe that simple table scan over joined data used by MySQL is much faster than joining (a part of) 5 input sources using any of our methods. On the other hand, if the sources are a small subset of the attributes or attributes have complicated structure, the joined table is not an usual state. Comparing top- k search with the database join of more tables is our nearest future work.

Table 4. Computation time in seconds over different size of data

data size	1 MB	10 MB	100 MB	1GB
3P-NRA	0,092	0,217	7,724	—
Hash partitioning	0,094	0,828	7,745	133
B+ tree	2,954	26,08	168	2125
table scan	0,082	0,296	2,325	19,72

The time taken by any of NRA-like algorithms increases with the size of the sources more rapidly than in the case of table scan. This is caused by the complexity of partial join computation against the simple table scan.

6 Discussion

In this paper we have studied methods of top- k search with no random access which can be used to find k best objects using sorted lists of attributes that can be read only by sorted access. Such methods usually need to work with a huge set of candidates during the computation and handling of candidates is an important issue.

Our motivation comes from querying a big repository with data extracted and/or retrieved from the web relevant to job search. Such data typically contain several dozens of different attributes and single user query typically uses a constant fraction of them, nevertheless all of them are used by some user.

We have introduced new methods for top- k search with sorted access and implementation of own experimental environment. Novel versions of *NRA* algorithm were tested with different data organization (hash partitioning and B+tree).

First result of our research is, that the problem of top- k complexity has more dimensions as expected in the beginning.

As it was expected, the complexity is directly proportional to the ratio of the size of data and the size of memory (the data to memory parameter d),

$$d = D/M = \frac{\text{size of data}}{\text{size of memory}},$$

although for different methods this dependency can be stronger or weaker.

New observation is that the complexity of top- k querying for systems with indexes based on RDBMS depends on the ratio of number of all attributes and (average) number of query attributes (the attributes parameter a)

$$a = AA/AQA = \frac{\text{No. all attributes}}{\text{avg. No. query attributes}}.$$

For the number of users u , it is proportional to a^u . *AQA* corresponds to the average value of m used in algorithms. Our system uses a light weight proprietary disk data structures. This serves only to top- k search, we do not need full

functionality of a transactional multiuser RDBMS. Maintenance is proportional to AA , nevertheless it depends on the number of users only linearly, with $u * AA$. Usually

$$a^u \gg u * AA.$$

This was not taken into account in any previous research. Several parameters of repositories like speed of incremental insert or speed of incremental join are also relevant. Our implementation offers incremental insertions.

In the project NAZOU (see <http://nazou.fiit.stuba.sk>), we use the sorted access to process the output from different methods, which produce data with respect to user preferences to attribute values. There are different methods over various index structures for different types of attributes: ordinal, nominal, hierarchical and metric attributes. To obtain the combination function for different users, our system uses learning user preferences from a sample of objects (see [11]) or a recommendation for a similar user based on it.

Considering, that users are interested only in a part of the attributes and that attribute domains can be complex structures, the new methods of top- k search become very useful.

Supported by Czech projects 1ET100300517, 1ET100300419 and MSM-0021620838 and Slovak projects NAZOU and VEGA 1/3129/06.

References

1. Akbarinia, R., Pacitti, E., Valduriez, P.: Best Position Algorithms for Top-k Queries. In: VLDB (2007)
2. Bast, H., Majumdar, D., Schenkel, R., Theobald, M., Weikum, G.: IO-Top-k: Index-Access Optimized Top-k Query Processing. In: VLDB (2006)
3. Balke, W., Güntzer, U.: Multi-objective Query Processing for Database Systems. In: VLDB (2004)
4. Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In: ICDE (2002)
5. Chang, K.C.C., Hwang, S.W.: Minimal probing: Supporting expensive predicates for top-k queries. In: SIGMOD (2002)
6. Fagin, R., Lotem, A., Naor, M.: Optimal Aggregation Algorithms for Middleware. In: ACM PODS (2001)
7. Gurský, P., Lencses, R., Vojtáš, P.: Algorithms for user dependent integration of ranked distributed information. In: TCGOV (2005)
8. Gurský, P., Šumák, M.: Top-k aggregator. In: Tools for Acquisition, Organisation and Presenting of Information and Knowledge, project proceedings (2006)
9. Gurský, P., Horváth, T., Novotný, R., Vaneková, V., Vojtáš, P.: UPRE: User preference based search system. In: IEEE/WIC/ACM Web Intelligence (2006)
10. Güntzer, U., Balke, W., Kiessling, W.: Towards efficient multi-feature queries in heterogeneous environments. In: ITCC (2001)
11. Horváth, T., Vojtáš, P.: Ordinal Classification with Monotonicity Constraints. In: Proc. 6th Industrial Conference on Data Mining ICDM (2006)
12. Hristidis, V., Papakonstantinou, Y.: Algorithms and Applications for answering Ranked Queries using Ranked Views. VLDB Journal 13(1) (2004)

13. Ilyas, I., Aref, W., Elmagarmid, A.: Supporting top- k join queries in relational database. In: VLDB (2003)
14. Ilyas, I., Shah, R., Aref, W.G., Vitter, J.S., Elmagarmid, A.K.: Rank-aware query optimization. In: SIGMOD (2004)
15. Li, C., Chang, K., Ilyas, I., Song, S.: RankSQL: Query algebra and optimization for relational top- k queries. In: SIGMOD (2005)
16. Ramakrishnan, R., Gherke, J.: Database management systems, 3rd edn. McGraw-Hill, New York (2003)
17. Soliman, M.A., Ilyas, I.F., Chang, K.C.C.: Top- k Query Processing in Uncertain Databases. In: Proc. ICDE (2007)
18. Re, C., Dalvi, N.N., Suciu, D.: Efficient Top- k Query Evaluation on Probabilistic Data. In: Proc. ICDE (2007)
19. Theobald, M., Schenkel, R., Weikum, G.: An Efficient and Versatile Query Engine for TopX Search. In: VLDB (2005)
20. Yu, H., Hwang, S., Chang, K.: Enabling Soft Queries for Data Retrieval. Information Systems. Elsevier, Amsterdam (2007)
21. Xin, D., Han, J., Chang, K.: Progressive and Selective Merge: Computing Top- K with Ad-Hoc Ranking Functions. In: SIGMOD (2007)
22. Zhang, Z., Hwang, S., Chang, K., Wang, M., Lang, C., Chang, Y.: Boolean + Ranking: Querying a Database by K -Constrained Optimization. In: SIGMOD (2006)

IDFQ: An Interface for Database Flexible Querying

Mohamed Ali Ben Hassine and Habib Ounelli

Department of Computer Sciences, Faculty of Sciences of Tunis,
El Manar 1, Campus Universitaire, 2092, Tunis, Tunisia
{mohamedali.benhassine, habib.ounelli}@fst.rnu.tn

Abstract. In traditional database management systems, imprecision has not been taken into account so one can say that there is some sort of lack of flexibility. The main cause is that queries retrieve only elements which precisely match to the given Boolean query. Many works were proposed in this context. The majority of these works are based on Fuzzy logic. In this paper, we discuss the flexibility in databases by referring to the Formal Concept Analysis theory. We propose an environment based on this theory which permits the flexible modelling and querying of a database with powerful retrieval capability. The architecture of this environment reuses the existing structure of a traditional database and adds new components (Metaknowledge Base, Context Base, Concept Base, etc.) while guaranteeing interoperability between them.

Keywords: Databases, Flexible queries, Formal Concept Analysis, Concept Lattice.

1 Introduction

The majority of existing information systems deals with crisp data through crisp database systems [1] and in traditional database management systems (DBMS) imprecision has not been taken into account, so one can say there is some sort of lack of flexibility. The main cause is that queries retrieve only elements which precisely match to the given Boolean query. That is, an element only belongs to the result if the query is true for this element. Many works are proposed to resolve this limitation. The majority of these works are based in Fuzzy theory which has been identified as a successful technique for modeling imprecise and vague data and also for effective data retrieval [2].

In other side, concepts are necessary for expressing human knowledge. Therefore, the process of discovering knowledge in databases (DB) benefits from a comprehensive formalization of concepts which can be activated to communicatively represent knowledge coded in DB. Formal Concept Analysis (FCA) [3] offers such a formalization by mathematizing concepts that are understood as units of thought constituted by their extension and intention. In this paper, we will discuss the flexibility in DB while referring to the theory of FCA. In fact, FCA is a method for data analysis, knowledge representation and information

management invented by Rudolf Wille in the early 80s [4] but until now the querying of DB based on FCA theory was rarely used. The few works which treat this problem did not present a theoretical framework [5,6,7].

This work presents some ideas to integrate the elements of FCA theory in the flexible queries handling within the field of modern architectures of DBMS. Exactly, it is proposed a DB architecture based on the Client/Server approach with the requirements that it must join to resolve the different aspects implicated in the treatment of information (contexts, concepts, linguistic terms, etc.). This architecture is implemented by a module called IDFQ (An Interface for Database Flexible Querying) which reuses the existing structure of relational DB and adds new components (MKB, CXB, CPB, Relaxable layer) to model with a new look (in the shape of context, concept lattice, etc.) the existing data and to make interoperability between them.

The rest of this paper is organized as follows. Section 2 presents the basic concepts of FCA. Section 3 presents the related work about the architectures dealing with flexible queries. Section 4 presents the core of the paper: The IDFQ. Section 5 presents a general description of flexible querying processing in IDFQ illustrated by an example. Finally, Section 6 concludes the paper and suggests some future works.

2 Background

2.1 Formal Concept Analysis

Formal Concept Analysis (FCA) was introduced by Wille [4] and represents an approach towards the extraction of highly similar groups of objects from a collection O of objects described by a set of attributes A . The paradigm occurred within the lattice theory [3]: the attributes considered represent binary features, i.e., with only two possible values, present or absent. In this framework, the discovered groups represent the closed sets of the Galois connection induced by a relation R on the couple O and A . For formalizing this understanding it is necessary to specify the objects and attributes which shall be considered in fulfilling a certain task. Therefore, Formal Concept Analysis starts with the definition of a formal context. For the mathematical foundations the interested reader is referred to Ganter, Wille (1996, 1999).

2.2 Formal Context, Formal Concept and Concept Lattices

Definition 1. A *formal context* is a triple $\mathbb{K} = (O, A, R)$, where O is a finite set of elements called objects, A a finite set of elements called attributes and R is a binary relation defined between O and A (i.e. $R \subseteq O \times A$). If $o \in O$ and $a \in A$ are in relation R , $(o, a) \in R$ is read “object o has attribute a ”.

Definition 2. Given a subset $X \subseteq O$ of objects from a context (O, A, R) , let $'$ an operator that produces the set X' of their common attributes for every set $X \subseteq O$ of objects to know which attributes from A are common to all these objects:

$$X' := \{a \in A \mid \forall o \in X : (o, a) \in R\}$$

Dually, for a subset $Y \subseteq A$ of attributes, Y' denotes the set consisting of those objects in O that have all the attributes from Y :

$$Y' := \{o \in O \mid \forall a \in Y : (o, a) \in R\}$$

These two operators are called the **Galois connection** of (O, A, R) . These operators are used to determine a formal concept.

Definition 3. A **formal concept** of a formal context (O, A, R) is a pair (X, Y) with $X \subseteq O, Y \subseteq A, X' = Y$ and $Y' = X$. The sets X and Y are called the **extent** and the **intent** of the formal concept (X, Y) , respectively. The subconcept-superconcept relation is formalized by:

$$(X1, Y1) \leq (X2, Y2) :\iff X1 \subseteq X2 (\iff Y1 \supseteq Y2)$$

The set of all formal concepts of a context \mathbb{K} together with the order relation \leq is always a complete lattice, called the **concept lattice** of \mathbb{K} and denoted by $\underline{\mathcal{B}}(\mathbb{K})$

Figure 1 depicted an example of a formal context and its associated lattice.

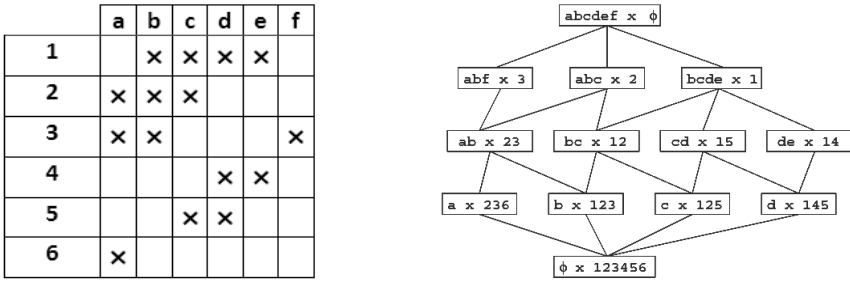


Fig. 1. Formal Context and its Concept lattice

2.3 Scaling

In many applications, objects may not just have attributes or not, but different values for the attributes. This is modelled by a many-valued context. A many valued context may not only have crosses (i. e., yes/no) as entries, but values of attributes pairs. It can be seen as a table of a relational database with the column containing the objects being a primary key.

Definition 4. A many-valued context is a tuple $\mathbb{K} := (O, A, (\mathbb{W}_a)_{a \in A}, I)$ where O is a set of objects, A a set of attributes, each W_a a set of possible values for the attribute $a \in O$, and $I \subseteq O \times \{(a, w) \mid a \in A, w \in W_a\}$ a relation with $(o, a, w_1) \in I, (o, a, w_2) \in I \Rightarrow w_1 = w_2$. $(o, a, w) \in I$ is read “object o has value w for attribute a ”.

A *conceptual scale* is a one-valued context which has objects possible values of the DB attributes. It used to extract the relevant information from the many-valued context such that a concept lattice can be generated. The choice of the attributes of the scale is purpose-oriented and reflects the understanding of an expert of the domain.

Definition 5. A *conceptual scale for a subset $B \subseteq A$ of attributes* is a (one-valued) formal context $\mathbb{S}_B := (\mathbb{O}_B, \mathbb{A}_B, \mathbb{I}_B)$ with $G_B \subseteq \times_{a \in B} W_a$. The realized scale $\mathbb{S}_B(\mathbb{K})$ is defined by $\mathbb{S}_B(\mathbb{K}) := (\mathbb{O}_B, \mathbb{A}_B, \mathbb{I}_B)$ with $(o, n) \in J$ if and if there exists $w = (w_a)_{a \in B} \in O_B$ with $(o, a, w) \in I$, for $a \in B$, and $(w, n) \in I_B$.

The idea is to replace the attribute values in W_A which are often too specific by binary, more general attributes which are provided in A_B . The many-valued context is realized as a table in a relational DB.

3 Related Work

3.1 Databases and Flexible Querying

One of the features of the human reasoning is that it may use imprecise or incomplete information. Moreover, in the real world, there exists a lot of this kind of data. Hence, we can assert that in our everyday life we use several linguistic terms to express abstract concepts such as *young*, *old*, *cold*, *hot*, and so forth. In this scenario, the classic queries treated by the majority of the existing DBMS retrieve only elements which precisely match to the given Boolean criterias. That is, an element only belongs to the result if the criterias are true for this element. The flexible querying of a DB, contrary to the Boolean classic querying, allows the users to use of linguistic terms and to express their preferences to better qualify the data that they wish to get [2]. An example of flexible query, is “*list of the young and well paid employees*”. This query contains the linguistic labels “*young*” and “*well paid*”. Many works are proposed to treat and model flexible queries. The majority of these works are based in Fuzzy theory and consists in developing extensions to the relational data model to interrogate imprecise data. The essential idea in these works consists in extending the SQL language and adding an additional layer to the classic DBMS to evaluate fuzzy predicates. Among these works, we mention the one of Bosc Medina et Galindo [28]. Other recent works propose methodologies to introduce flexible queries into relational DBMS [9,10].

3.2 Existing Architectures

In order to model and treat flexible queries in the different types of DB (relational, object, etc.), some architectures are proposed to incorporate the different components used to built a system which provide answers to these queries. Generally, in these architectures, authors propose new languages, metabases, etc. specific to the DB nature. For lack of space, we propose, here, the known ones.

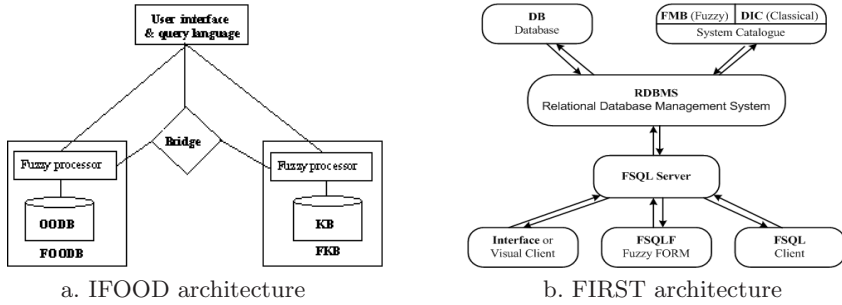


Fig. 2. Proposed architectures

First, we mention Yazici and Koyuncu [11] which proposed an Fuzzy Object-Oriented DB (FOODB) model as an extension to the object-oriented DB model, which permits the flexible modelling and querying of complex data and knowledge including uncertainty with powerful retrieval capability. The architecture of the proposed environment for intelligent fuzzy object oriented DB (IFOOD) modelling is given in Fig. 2.a. The architecture is based on coupling the FOODB with a Fuzzy Knowledge Base (FKB) and the development of a user interface and language. The communication between the DB and the FKB is performed by a bridge that provides interoperability between them to achieve the management of data and knowledge for answering various queries.

In their turn, Medina et al. proposed an architecture named Fuzzy Interface for Relational SysTems [12] (FIRST extended then with FIRST-2 [2]) in order to implement a system which represent and manipulate “imprecise” information. This architecture is built on RDBMS Client-Server architecture, in provided by Oracle. It reuses the existing structure and adds new components (Fuzzy Metaknowledge Base “FMB”, FSQL Server, etc.) to handle fuzzy information. The schema of the architecture is showed in Fig. 2.b.

Another architecture was proposed to deal with flexible queries for Fuzzy Relational Deductive DB named FREDDI (Fuzzy RELational Deductive DB Interface) [13]. In fact, FREDDI extends the Fuzzy Relational DBMS (FRDBMS) [14] architecture with deduction capabilities and preserves two of their approaches: the relational and the logical ones. It uses the first one to represent and manipulate imprecise information and the second one to obtain intensional information.

The queries are modelled by a specific fuzzy languages for every architecture. FSQL (Fuzzy SQL) for FIRST, IFOOD language for IFOOD and DFSQL (Deductive FSQL) for FREDDI. In all of these architectures, the fuzzy statements are translated into one or more SQL statements, which can be used by the host DBMS.

4 IDFQ: An Interface for Database Flexible Querying

Our proposed architecture of relational DB in the FCA environment is built by elements of different nature: Structures of data (tables, views, contexts, concepts, etc.), a grammar for the definition of data “DDL” (Data Definition

Language) and the handling of data “DML” (Data Manipulation Language), programs (in different languages), etc. These are the big lines, but in this work, we don’t discuss the extension of the SQL language, we only restricted to provide an architecture permitting to treat imprecise information in users queries, and consequently to formalize a method to do it. This architecture is implemented by a module called IDFQ (an Interface for Database Flexible Querying).

4.1 The Architecture

Figure 3 shows the general scheme of the proposed architecture for IDFQ. The architecture, built on Relational DBMS Client-Server approach, reuses the existing structure of relational DB and adds new components (MKB, CXB, CPB, Relaxable layer) to model with a new look (in the shape of context, concept lattice, etc.) the existing data and to make interoperability between them. This is, of course, on the aim of generating new look of flexible queries modelling.

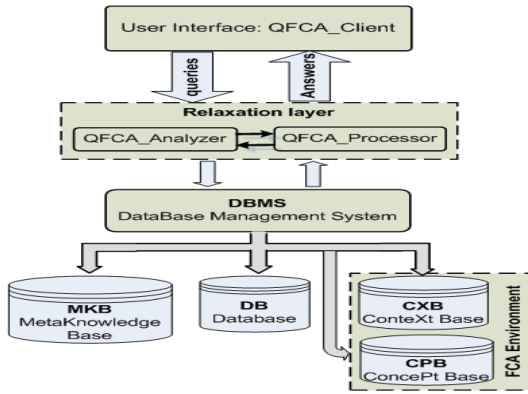


Fig. 3. IDFQ architecture

The main components of IDFQ are:

- **DB**: It stores all the permanent information following the relational model.
- **ConcePt Base (CPB)**, **ConteXt Base (CXB)** and **MetKnowledge Base (MKB)**: They extend the DB with the information about the relaxable attributes¹ and their mapping through the FCA representation. A presentation of these bases is described in the next section.
- **DBMS**: All the operations will be translated into requests for the host DBMS. These requests will be made using a language that the DBMS supports. Generally, all the requests for the system are solved by classical SQL sentences.

¹ A relaxable attribute is a column in the DB which can be described by linguistic terms in the query. For example the attribute *age* can be described by *Young*, *Adult* and *Old*.

- **QFCA₂Client**: This is the program that establishes an interface that provides a unified environment for both data and knowledge management and allows users to have the capability of query processing independent from the physical structure of the architecture.
- **Relaxation Layer**: In a global view, it is the part of the system that interconnects the user and the DBMS. In depth view, it interconnects the DB, MKB, CXB, CPB and the different queries treatment engines. It consists mainly of:
 - **QFCA_Processor**: Its objective is to extract the queries written in SQL language and translate them in concept queries (crisp-to-concept mapping). To do this translation, it will use the information stored in the MKB, DB, CXB and the CPB. It returns the results to the QFCA_Analyser.
 - **QFCA_Analyzer**: Its main function is the construction of the concept lattice of the user query and the navigation between their nodes. The returned nodes are analyzed, sorted and then rewritten in accordance with the original format (concept-to-crisp mapping). In Sect. 5.3, we explain that these lattices are, in fact, concept tables.

4.2 Data in IDFQ

As we described above, the basic idea in our approach is as follows: We firstly treat any relation of the DB as a many-valued context of FCA (Sect. 2.3). Then, we transform this relation to a formal context. After that, we generate the concept lattice of this context (Fig. 5 and 6). These are the main steps of pretreatment that precede the querying of a DB (Sect. 5.1), but at each transformation level, the data is stored in a specific base.

Context Base: The use of concept lattice in data mining, data base, data analysis, information retrieval suffer from its time building which depends strongly on its size (number of attributes and objects) and the algorithms used to built it. This problem led the researchers to study and use the lattice for academic DB of small size. To circumvent this problem, we propose a context base to store all the contexts generated from the tables containing relaxable attributes. These contexts, which are the startup point to use FCA are the result of a mapping table-to-context through these scaling methods:

1. **Discretization:** Discretization is a process that transforms quantitative data into qualitative data. A many to one mapping function is created so that each value of the original quantitative attribute is mapped onto a value of the new qualitative attribute. We can say that all discretization methods differ with respect to how they measure the quality of the partitioning. A good discretization algorithm has to balance the loss of information intrinsic to this kind of process and generating a reasonable number of cut points. In our work, we used the discretization method proposed by Fayyad et Irani [15]. A recent overview of various types of discretization algorithms can be found in [16,17].

² Query based on FCA.

2. **Clustering:** The clustering process attempts to discover significant groups present in a data set. It is unsupervised process. In our approach, the obtained clusters form the context attributes. In our work, we propose two clustering methods:
- Classical clustering: we generate the clusters through an extended version of Cluster Methods [18]. Each obtained cluster is represented by a trapezoidal membership function.
 - Fuzzy clustering: Fuzzy clustering techniques allow objects of a data set to belong to several clusters simultaneously, with different membership degrees. The data set is thus partitioned into a number of fuzzy partitions (clusters). We generate the fuzzy clusters through an extended version of Fuzzy C-Means Algorithm [19].

It is important to note that IDFQ architecture is not restricted to this scaling methods, the DB administrator can use the most appropriate one [7][20].

Concept Base: In the some way and to avoid the hard processing of lattices building, we generate, through specific algorithms [3][21], all the concepts of the different contexts and store them in a meta base named ConcePt Base (CPB). We propose two options (having similar schemas of tables) to define the CPB:

- to create similar tables, each one is prefixed in its name by the context name source,
- to create one table containing all the concepts of the CXB (used in this work). The scheme of this table (Concept_Table) is described in Table 1.

Table 1. Concept_Table

Context#	Level#	Node#	Intent#	Extent#	SuccessorList	PredecessorList	IntentSize	ExtentSize
...

The columns of Concept_Table and their meanings are as follows:

- **Context#:** The name of the context source.
- **Level#/Node#:** These two columns store the concept identifier in a Context. The first one represents the level of the concept in the lattice while the second one represents the sequence number of the concept at this level [22].
- **Intent/Extent:** These two columns store the intent (respectively extent) of each concept.
- **SuccessorList#/PredecessorList#:** These two columns store the identifier of the successor(s) (respectively the predecessor(s)) of a concept.
- **IntentSize/ExtentSize:** These two columns store the cardinality of a concept (respectively number of attributes and number of objects).

Since the data does not vary frequently (by assumption), the maintenance of CXB and CPB is not expensive. For this reason, we use periodically some algorithms to maintain the CXB and the CPB updating [21].

MetaKnowledge Base: The MetaKnowledge Base (MKB) is an extension of the host DBMS catalog whose mission is to provide IDFQ with information about all the relaxable attributes, data and definitions stored in the DB. All this information is organized and stored in accordance to the scheme of the host DBMS. Figure 4 shows the MKB relations, their attributes, their primary keys (underlined), and their foreign keys (with arrows). Due to the lack of space, we present briefly a description of their relations:

- `Relaxable_Attributes_List`: This table contains a list of the attributes that are defined in the columns of the DB and chosen for flexible queries treatment.
- `Contexts_List`: This table stores the list of contexts which are generated in the pretreatment phase.
- `Concepts_List`: This table stores the list of concepts for each context.
- `Attributes_Equivalence`: This table stores the equivalence of each DB attribute and its corresponding context attribute.
- `Compatible_Attributes`: This table contains a list of the attributes that are compatible between them. This table is used for comparison operations.
- `Labels_Definition`: This table contains the points that define the trapezoidal possibility distribution associated to the labels of the context attributes of type CA1 and CA3.
- `Labels_Similarity_Definition`: This table presents the measures of resemblance, similarity, or proximity between the different labels of the context attributes of type CA2.
- `Clusters_List`: This table contains the different clusters used in the CXB.

Attributes Types in IDFQ: As we saw in the previous section, and after the crisp-to-fuzzy mapping, we obtain two families of attributes: DB attributes and Context attributes. Since the flexible querying processing in IDFQ touches directly the context attributes, the DB attributes types depend strongly to the values of the context attributes. For example, context attributes which represent trapezoidal distribution must be generated from DB attributes interrogated by label values.

1. **DB attributes:** these are the attributes of the original DB which accept crisp data and interrogated by fuzzy values (labels, proximity values, trapezoidal distributions, etc.). We classify them in two types:
 - **Attributes Type 1 (T1):** these are attributes over an *ordered domain* (eg. Salary) which are interrogated by some labels defined by trapezoidal distributions over an underlying ordered dominion (fuzzy sets) (e.g. “low”, “medium”, “high”, etc.) or by crisp values. The correspondence between these labels and the context attributes is the result of fuzzy function determination and classification. The name and the definition (trapezoidal fuzzy set) of each label are stored in the MKB.
 - **Attributes Type 2 (T2):** these are attributes over *non ordered domain* (eg. Color). In these attributes, some labels are defined with a similarity relationship (or proximity). Example: colors are not ordered, but orange

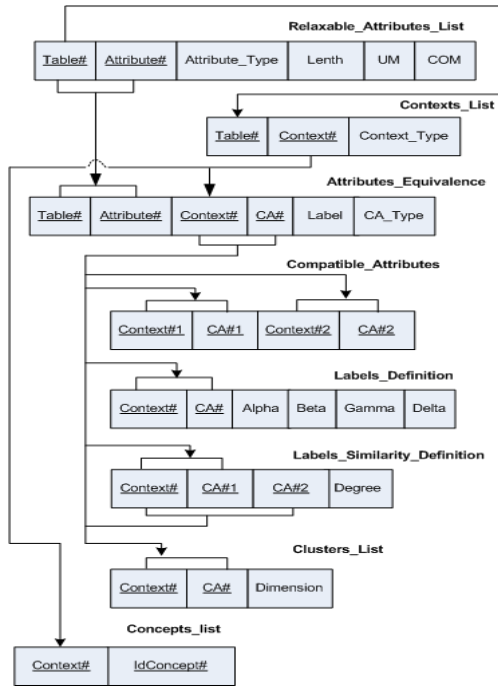


Fig. 4. MetaKnowledge Base scheme of IDFQ

is more similar to red than bleu. (Table 2 shows the similarity relations between the labels of the attribute color). Thus, this relation indicates to what extent each pair of labels is similar.

Table 2. similarity relations for the attribute Color

Similarity	Orange	Red	Blue
Orange	1	0.7	0.1
Red	0.7	1	0.05
Blue	0.1	0.05	1

- Context attributes (CA):** These attributes are the result of table-to-context mapping. This mapping returns intervals, clusters, trapezoidal distributions, etc. It is important to note that, before this mapping, these generated attributes were the values of the DB attributes. However, in FCA, these values are called attributes (proprieties). For these reason, to make the correspondence with FCA environment, and to avoid the overlapping between the notations we note these values “Context Attributes” (CA). We distinguish three types:

- CA1 (L1): these are labels over an *ordered domain* and defined by trapezoidal distributions (they can be fuzzy quantifiers) or by intervals. For example, we can represent the three labels *young*, *adult* and *old* of the employee *Age* by a trapezoidal distributions or by intervals (we consider these intervals as a particular case of trapezoidal possibility distribution with the values beta=alpha and gamma=sigma.). These attributes can represent the DB attributes of type 1. The parameters of these labels are stored in the MKB.
- CA2 (L2): these are labels over *non ordered domain* with similarity relations. These attributes represent the DB attributes of type 2.
- CA3 (L3): these are clusters of n dimension ($n > 1$) generated after a clustering process.

5 Processing of Flexible Queries

As indicated previously, if we combine a suitable representation of information with an adequate implementation through the data structure given by the host DBMS, it is possible to produce processes which translate classic queries (or queries written in FCA_SQL language) to concept queries (and vice versa) and execute them by the classical DBMS. In this section we are going to illustrate some aspects of this process.

5.1 Functioning Description

We suppose in our solution that the DB is steady. For this reason, we choose to share the treatment between two phases which can be summed up as follows (Fig. 5):

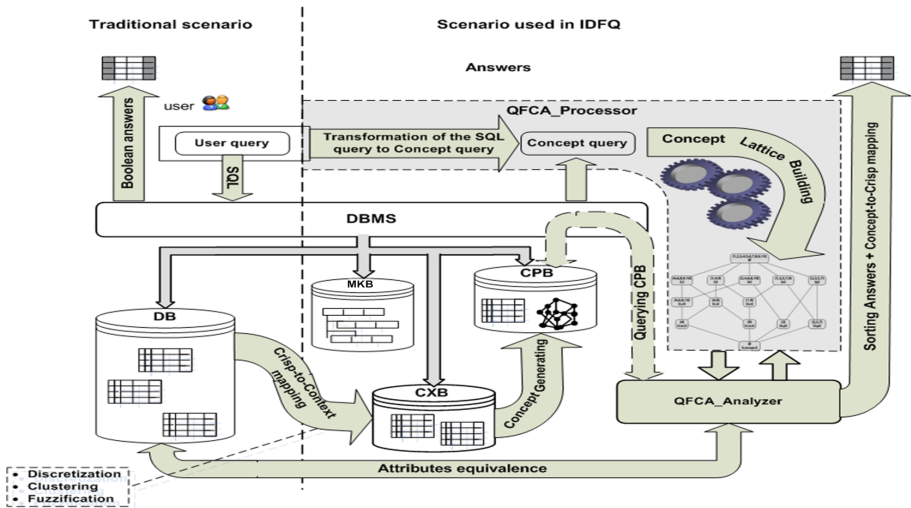


Fig. 5. Flexible queries processing

1. A pretreatment phase: it is executed before querying the DB and will be updated periodically depending on the DB variation and consists to:
 - store in the CXB all the contexts through a crisp-to-context mapping.
 - generate and store all the concepts related to these contexts in the CPB.
2. At the query time: it is launched by a user query and consists to search the answers in the concepts already stored in the CPB (concept_table). To do this, the query must be translated to the concept format (intent, extent).

5.2 Example

We summarize our approach by the following example:

Let us consider the relational table “Employee” (Table 3) described by the attributes: Ssn, Name, Salary and Productivity and the following query: “Retrieve the young employees with low salary and regular productivity”.

At the first look to the Employee table, we can not find any tuple which satisfy the user query. Table 4 shows the context table (binary) related to the Employee table and on the left of Fig 6 its related concepts.

Table 3. Extension of the table Employee

Ssn	Name	Age	Salary	Productivity
1	Ben Hassine	28	800	Regular
2	Grissa	26	1000	Regular
5	Guesmi	26	450	Bad
3	Hachani	31	600	Regular
4	Ounelli	47	1200	Good

Table 4. Binary Employee Context

	Young	Adult	Old	Low	Medium	High	Bad	Regular	Good
1	1	0	0	0	1	0	0	1	0
2	1	0	0	0	0	1	0	1	0
3	1	0	0	1	0	0	1	0	0
4	0	1	0	0	1	0	0	1	0
5	0	0	1	0	0	1	0	0	1

The user query is translated according to the table-to-context mapping as follow: concept query $CQ = (\{Query\}, \{Young, Low, regular\})$.

In this example, the insertion of the concept CQ leads to add a new concept since no object have the same intent. To make easy the understanding of the querying processing, Fig. 6 summarize this modification and gives an idea about the employees *near* to the query conditions. However, in our implementation, we consider the concept lattice as a table in the CPB, so we are not obliged to draw it.

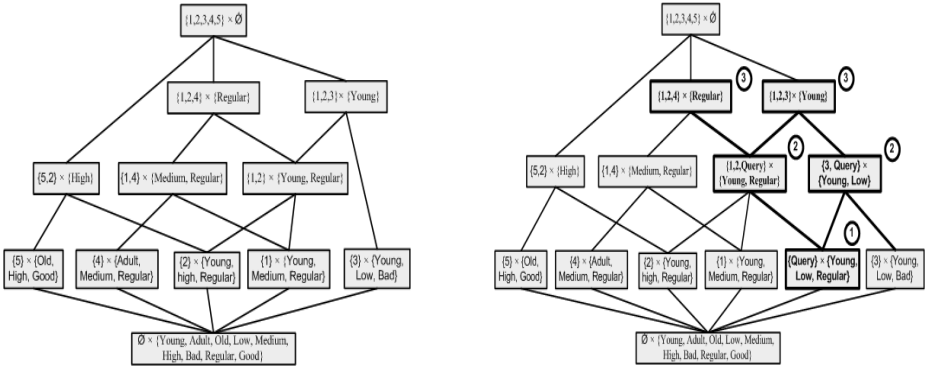


Fig. 6. Concept lattice of the Employee Context before and after the insertion of the query

The numbers shown near the concepts represent the steps (iterations) during which the corresponding concepts have been visited (considered) at the time of the construction of the result. It is in this order that the objects will appear in the final result. Thus, the result returned to the considered query is constituted of the employees of the set $R_0 = \{\emptyset\}$, $R_1 = \{1, 2, 3\}$ and $R_2 = \{4\}$. It is presented like follows:

- No object which shares with the query its three properties (R_0).
- 1 and 2 share with the query its two properties *Young* and *Regular* (R_1).
- 3 shares with the query its two properties *Young* and *Low* (R_1).
- 4 shares with the query its property *Regular* (R_2).

5.3 Discussion

Due to lack of space, the example described above shows a general querying processing. Of course, in our system we use some optimizations in this treatment. For example, we don't update for each query the corresponding lattice because it will increase the time processing and make the querying processing impossible for large DB (central memory saturation, etc.). Theoretically, we generate only the part of the the lattice related to the query, i. e. the part of the lattice marked in bold in Fig. 6. In our implementation, since we stored all the concept in the CPB and since each lattice constitutes a representation of a concept table (see table 1), we don't need to draw it; the method described in the example is translated then to a search in the CPB for the concepts having relation with the concept query. With this manner, on one hand, we reduce the time processing specially when the DB is large, a problem which constituted a handicap to interrogate DB by FCA theory. On the other hand, we get a set of the nearest (neighbours) answers (ordered by their satisfaction degree) to the query conditions especially in the case of empty answers.

6 Conclusion

This paper presents an interface for database flexible querying (IDFQ). At the level of data, IDFQ add a context base, a concept base and a Metaknowledge base containing information about relaxable attributes, context attributes with their definitions (intervals, labels, clusters, etc.). At the level of programs, IDFQ contains a relaxation layer permitting interconnection between data (relational and context) and the user interface. The comprehension level of query (that is the system flexibility) is higher that the Metaknowledge base is more completed. So that, through the user interface, the Metaknowledge base can be more continuous enriched by any authorized user, having the role of the knowledge engineer. We remark the general character of the system, which can be connected and used like an interface to query any database, but with condition the Metaknowledge base, the context base and the concept base must be prepared in advance for this task. The two big advantages of the system are: providing the nearest answers to the user query and using linguistic terms (for expression gradual properties, approximation or intensify properties). Of course, this last is still in work. This article presents only first steps towards an integration of Database Theory and Formal Concept Analysis by presenting an architecture grouping the main elements to establish their integration. Some of the topics to be approached next is certainly define an algebra to manage data in FCA environment and a language accepting linguistic terms and approximates attribute values in the user query (like the ones used in fuzzy DB).

References

1. Elmasri, R., Navathe, S.B.: *Fundamentals of Database Systems*, 4th edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
2. Galindo, J., Urrutia, A., Piattini, M.: *Fuzzy Databases: Modeling, Design, and Implementation*. IGI Publishing, Hershey (2006)
3. Ganter, B., Stumme, G., Wille, R.: *Formal Concept Analysis: Foundations and Applications*. Springer, Heidelberg (1999)
4. Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts. In: Rival, I. (ed.) *Ordered sets*, pp. 445–470. Reidel, Dordrecht (1982)
5. Stumme, G., Wille, R., Wille, U.: Conceptual knowledge discovery in databases using formal concept analysis methods. In: *Principles of Data Mining and Knowledge Discovery*, pp. 450–458 (1998)
6. Priss, U.: Establishing connections between formal concept analysis and relational databases. In: *13th International Conference on Conceptual Structures, ICCS 2005*, Kassel, Germany, July 2005, pp. 132–145 (2005)
7. Hereth, J.: Relational scaling and databases. In: Priss, U., Corbett, D., Angelova, G. (eds.) *ICCS 2002*. LNCS (LNAI), vol. 2393, pp. 62–76. Springer, Heidelberg (2002)
8. Bosc, P., Pivert, O.: Sqlf: a relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems* 3(1), 1–17 (1995)
9. Ben Hassine, M.A., Ounelli, H., Touzi, A.G., Galindo, J.: A migration approach from crisp databases to fuzzy databases. In: *Proc. IEEE International Fuzzy Systems Conference FUZZ-IEEE 2007*, London, July 23-26, 2007, pp. 1872–1879 (2007)

10. Ben Hassine, M.A., Grissa, A., Galindo, J., Ounelli, H.: How to achieve fuzzy relational databases managing fuzzy data and metadata. In: Galindo, J. (ed.) *Handbook on Fuzzy Information Processing in Databases*. Information Science Reference, vol. 2, pp. 348–372 (2008)
11. Koyuncu, M., Yazici, A.: Ifood: an intelligent fuzzy object-oriented database architecture 15(5), 1137–1154 (2003)
12. Medina, J.M., Pons, O., Vila, M.A.: First: A fuzzy interface for relational systems. In: *Proceedings of the Sixth International Fuzzy Systems, Association World Congress, Brazil*, vol. 2, pp. 409–412 (1995)
13. Medina, J.M., Pons, O., Cubero, J.C., Miranda, M.A.V.: Freddi: A fuzzy relational deductive database interface. *International Journal of Intelligent Systems* 12(8), 597–613 (1997)
14. Medina, J.M., Pons, O., Vila, M.A., Cubero, J.C.: Client/server architecture for fuzzy relational databases. *Mathware & soft computing* 3(3), 415–424 (1996)
15. Fayyad, U.M., Irani, K.B.: On the handling of continuous-valued attributes in decision tree generation. *Machine Learning* 8(1), 87–102 (1992)
16. Kotsiantis, S., Kanellopoulos, D.: Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering* 32(1), 47–58 (2006)
17. Liu, H., Hussain, F., Tan, C.L., Dash, M.: Discretization: An enabling technique. *Data Min. Knowl. Discov.* 6(4), 393–423 (2002)
18. Hachani, N., Ounelli, H.: Improving cluster method quality by validity indices. In: Wilson, D., Sutcliffe, G. (eds.) *FLAIRS Conference*, pp. 479–483. AAAI Press, Menlo Park (2007)
19. Sassi, M., Touzi, A.G., Ounelli, H.: Using gaussians functions to determine representative clustering prototypes. In: *DEXA 2006: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pp. 435–439. IEEE Computer Society, Washington (2006)
20. Stumme, G.: Local scaling in conceptual data systems. In: *ICCS 1996: Proceedings of the 4th International Conference on Conceptual Structures*, pp. 308–320. Springer, London (1996)
21. Godin, R., Missaoui, R., Alaoui, H.: Incremental concept formation algorithms based on galois (concept) lattices. *Computational Intelligence* 11, 246–267 (1995)
22. Gammoudi, M.M.: *Décomposition conceptuelle des relations binaires et ses applications*. Habilitation en Informatique, Faculté des Sciences de Tunis (June 2005)

Autonomic Databases: Detection of Workload Shifts with n-Gram-Models

Marc Holze and Norbert Ritter

University of Hamburg, Department of Informatics,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{holze,ritter}@informatik.uni-hamburg.de
<http://www.informatik.uni-hamburg.de/>

Abstract. Autonomic databases are intended to reduce the total cost of ownership for a database system by providing self-management functionality. The self-management decisions heavily depend on the database workload, as the workload influences both the physical design and the DBMS configuration. In particular, a database reconfiguration is required whenever there is a significant change, i.e. shift, in the workload.

In this paper we present an approach for continuous, light-weight workload monitoring in autonomic databases. Our concept is based on a workload model, which describes the typical workload of a particular DBS using n-Gram-Models. We show how this model can be used to detect significant workload changes. Additionally, a processing model for the instrumentation of the workload is proposed. We evaluate our approach using several workload shift scenarios.

1 Introduction

Vendors of commercial relational database management systems (DBMS) have continuously added new features to their products. While on the one hand this featurism makes DBMSs applicable for many different purposes, it on the other hand requires a subtle configuration for a particular environment. Commercial DBMS today offer hundreds of configuration parameters and physical design options that must be correctly chosen for a reasonable performance. Hence, it has been noticed ([1], [2]) that today the total cost of ownership for database systems (DBS) is not dominated by hardware- or software-costs anymore, but by “people cost” for database operation and maintenance. The ability of DBMS to perform self-management is considered as a way out of the high DBS maintenance costs. The DBMS is expected to autonomically perform maintenance operations and re-configurations when they are required, thus providing self-configuration, self-optimization, self-healing and self-protection [3].

The workload of a DBS has major influence on all DBMS self-optimization and self-configuration decisions, because it reflects the way the DBMS is used in its particular environment. For example, all physical design decisions heavily depend on the database workload, because the access paths should optimally support the typical query structure. Many configuration parameters also should

be set according to the workload, e.g. the buffer- and sort-pool sizes. As the workload of a database typically evolves over time, it is required to regularly check whether the DBS is still optimally configured for the workload. Thus, in order to realize an autonomic database system, it is essential to perform continuous monitoring and analysis of the DBS workload.

Recently, commercial DBMS vendors have started to integrate *autonomic features* into their products, which help the DBA to adapt the DBS to the workload: *Advisors* are designed as offline tools that support DBAs with recommendations on specific administration tasks, e.g. physical design advisors ([1], [4], [5]). Although these advisors produce good results, their analysis is far too heavy-weight to be run continuously. In contrast, *Feedback Control Loops* constantly monitor and reconfigure a specific managed resource. To keep the monitoring and analysis overhead small, their view is limited to a specific domain, e.g. autonomic memory management ([6], [7]). For this reason, feedback control loops focus on the workload of only the component that they manage (e.g. buffer pools) and cannot determine how their observations relate to the workload in other components or the entire DBS.

The work presented in this paper focuses on the continuous, light-weight monitoring and analysis of the overall database workload. For this purpose, we follow the idea of a two-staged workload analysis that we have described in [8]. This idea is based on the observation that in real-world DBS the workload is determined by the applications using the database. Typically, these applications work with a fixed set of static SQL statements or statement-templates, leading to specific usage patterns. In times where the workload is stable, i.e. there are only minor fluctuations in the way the database is used, there is no need to perform extensive reconfiguration analysis. Our idea of two-staged workload-analysis (see Fig. 1), exploits this observation: In the first stage, the self-management logic continuously monitors information about the current workload of the database. This stage only performs a light-weight comparison of the observed workload against a workload model, which describes the typical usage of the DBS. Only if a significant change (*workload shift*) from the typical usage is detected, the second, heavy-weight reconfiguration analysis stage is started.

The contribution of this paper is an approach based on n-Gram-Models for the detection of workload shifts in the first stage of workload analysis (the second stage is not in the focus of this paper). We illustrate how to automatically learn a workload model, how to assess an observed workload based on a n-Gram-Model, and how to adapt the model to a changed DBS usage. For this purpose we give an algorithm for the management of the workload model lifecycle. Furthermore, we propose a flexible approach for the instrumentation of the database workload.

The paper is organized as follows: The problem description in Section 2 discusses the requirements of a workload model. Section 3 then first describes a flexible approach for monitoring database workload, before Section 4 presents the contents and management of the workload model for workload shift detection. We evaluate our approach in Section 5 and discuss related work in Section 6. We conclude with a summary and outlook on future work in Section 7.

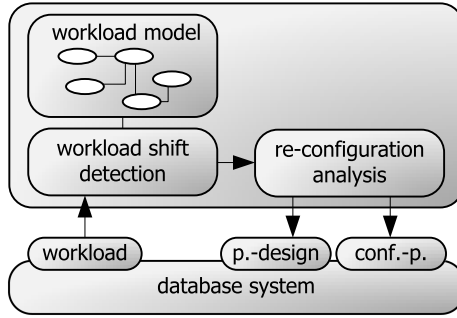


Fig. 1. Two-Stage Workload Analysis

2 Problem Description

The purpose of workload shift detection is the identification of situations when the usage starts to significantly change. The usage changes that must be detectable are all those scenarios, which usually require a DBA to re-analyse the configuration of a DBS. In particular, we consider the following scenarios as relevant shift situations:

- *New Applications*: When new database applications are deployed, they will cause additional workload on the DBS. To meet the performance requirements of all applications it may be necessary to perform DBS reconfigurations.
- *Obsolete Applications*: Workload that does no longer occur may also require DBS reconfigurations. If for example one database application is undeployed, some physical access structures or bufferpools may become obsolete.
- *Modified Applications*: With the installation of new releases, the functionality of enterprise applications usually evolves over time. The resulting workload may change significantly in every release.
- *Application Usage Changes*: Even with a fixed number and type of database applications, the DBS workload may change due to the user behaviour. If for example a reporting tool is used by twice the number of end users, the composition of the overall DBS workload may shift significantly.

In the above scenarios, the change in the workload is *permanent*, i.e. the changes will persist in the future. In addition to permanent shifts, databases often also face periodic workload shifts. Periodic shifts occur when a certain type of database usage occurs in regular intervals. For the workload shift detection, it is important to distinguish between two types of periodic shifts:

- *Short-Term Patterns*: These patterns refer to periodic workload shifts where the interval between the usage changes is small, e.g. batch updates in an 15 minutes interval or hourly reports. In this case, the periodic reconfiguration analysis and execution effort is likely to exceed the resulting benefit. Their actual length is installation-specific and must be configurable. Short-term patterns must *not* be detected as a workload shift.

- *Long-Term Patterns*: In cases of long-term patterns the period length justifies the reconfiguration analysis effort. An example is a DWH-scenario, where the DBS is loaded at night, and queried at day-time. These two distinct workload types require completely different DBS configurations and should therefore be detected as workload shifts. The workload model should furthermore provide the possibility to predict these periodic patterns, as this would make it possible to perform the reconfigurations proactively.

The workload shift detection must detect the *detection of the usage change scenarios* described above. This requirement implies that the detection mechanism may identify all usage changes which *might* require a DBS reconfiguration. The decision whether or not this usage change actually leads to a reconfiguration does not have to be covered by the shift detection. To provide a reliable reconfiguration indicator, the workload shift detection must furthermore ensure that *only* the described usage changes are detected. Minor workload fluctuations must be identifiable as such (*resilience to noise*).

The goal of continuous workload monitoring is the autonomic reconfiguration of the DBS, in order to lower the total cost of ownership for a DBS. Hence, the workload model must be *learnable and adaptable* by the DBMS without causing any effort for the DBA. For a light-weight shift detection, the learned workload must be represented in a *compact model*, i.e. the model should abstract from unnecessary details. One of the lessons learned from the first autonomic features in commercial DBMSs was that users do not trust in autonomic functionality [9]. For this reason the workload model should be *comprehensible*.

From these workload shift detection requirements, the following key challenges can be identified: In order to realize a DBS usage change detection, the ability for *workload instrumentation* is an essential prerequisite. It has to be decided

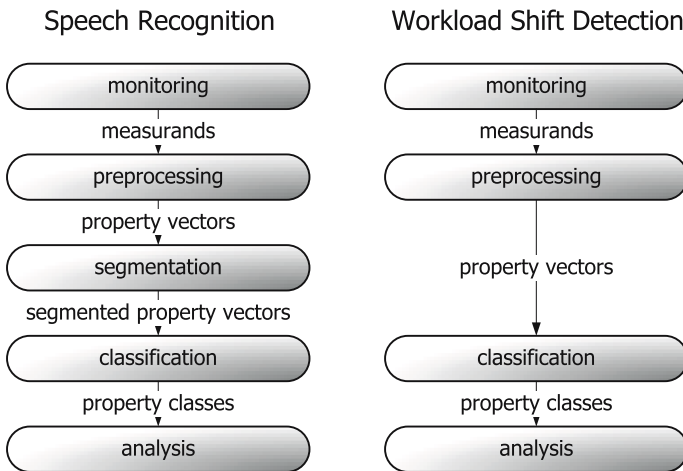


Fig. 2. Processing Models in Speech Recognition and DBS Workload Analysis

which metrics of the operating system or DBMS are monitored for this purpose, while imposing as little overhead as possible. The measured workload information must be represented in a compact workload model, i.e. the information in the model must abstract from minor differences in the measurements. Hence, the workload shift detection must perform a *workload classification* of similar workload situations. As the model must be automatically learned, an appropriate modelling technique has to be chosen that supports *model learning*. Afterwards, the workload shift detection has to perform *workload assessment*, i.e. the DBS workload must be continuously compared against the learned model. Metrics must be found that can be used to describe how well the observed workload matches the model. These metrics have to be assessed in a way that on the one hand reliably detects the usage change scenarios, and on the other hand assures resilience to noise. Whenever a significant workload shift has been detected, an alarm should be raised which triggers the DBS reconfiguration. In this case, a *model adaptation* must be performed automatically in order to represent the changed workload in the model.

3 Processing Model

An analysis of the key challenges described in Section 2 has shown that they are similar to the challenges in pattern recognition, especially speech recognition. This field of computer science also requires the assessment of measured data with the help of (language) models. As a well-established and tested processing model has already been developed in this discipline, we adapted this model for the purpose of workload shift detection. Fig. 2 provides an overview of the pattern recognition processing model as described in [10] and its adaptations for DBS workload analysis. The remainder of this section describes those processing stages of the processing model in detail.

The first stage in the processing model is *monitoring*. In principle, there are many possibilities for the monitoring of DBS workload. Some existing autonomic features like the IBM DB2 Utility Throttling [11] use the CPU usage, memory usage or I/O activity operating system metrics. Others monitor database-internal metrics of specific sub-components, like the buffer pool hit ratios or optimizer cost estimations. In contrast to these approaches, we consider monitoring the DBS workload at the SQL API (e.g. via a proxy or SQL statement monitor) as appropriate for workload shift detection. The reason is that the load of the DBMS-components directly depends on the set of observed SQL statements. Compared to monitoring the workload at all DBMS components independently, this centralized monitoring will impose less overhead. Furthermore, monitoring this standardized API makes the approach applicable for all relational DBS.

In the pattern recognition processing model, the monitored information (measurands) is preprocessed, e.g. filtered and described in terms of multi-dimensional property vectors. *Preprocessing* the monitored DBS workload is also reasonable for workload shift detection in order to filter certain types of SQL which should

not be considered in the model, e.g. DDL statements. The conversion of the monitored workload into multi-dimensional property vectors is future work.

On the stream of preprocessed property vectors the speech recognition processing model next performs *segmentation* into meaningful sections, i.e. splitting at the word boundaries. As the DBS workload is already segmented into the individual statements, this stage of the processing model may be skipped.

Classification refers to placing individual observations into groups according to certain characteristics. For speech, this stage maps the property vector segments to a symbolic representation of the spoken words. Only some characteristics of the recorded speech are evaluated for this purpose (thus allowing the same word to be pronounced/spoken in different ways). Classification is also required for DBS workload in order to fulfil the requirement of a compact workload model. Otherwise every monitored SQL statement with a distinct statement text would result in a separate class in the model. For a database with one single table of 100,000 customers that are accessed by their ID, the workload model would comprise 100,000 different SQL statements. From a DBS-load perspective, all statements that cause a similar load on the DBS should be classified into common classes, e.g. by comparing access plans (*DBS-load classification*). But the workload-shift-scenarios that we defined in Section 2 suggest a more semantic view on the classification. From this perspective, similar actions in the application programs should be classified into common classes (*semantic classification*). Both classification approaches have major drawbacks: The semantic classification would require the manual definition and maintenance of classification rules. This manual effort thwarts the goals of autonomic databases. The DBS-load classification would lead to considerable effort for the comparison of access plans, and it would not be easily comprehensible. Hence, we have chosen a simple *statement type classification* mechanism that is inexpensive and can be performed without DBA configuration effort: The SQL statements are classified by replacing all concrete parameter values in the statement text with a wildcard-character. If a class that represents the resulting un-parametrized string already exist, then the SQL statement is assigned to this class. Otherwise, a new class is dynamically created. The statement type classification introduces inaccuracy into the model by not considering effort differences due to data skew or locality of transactions. Also the number of classes cannot be known or limited in advance, and identical queries written in syntactically different ways will be assigned to different classes. Still, this straight-forward classification has produced good results (see Section 5) while imposing little overhead. More sophisticated classification algorithms can be evaluated and – due to the strict separation of processing stages – easily integrated in the future.

The last stage of the processing model is the *analysis* of the classified information. In the context of speech recognition this typically is language understanding. For workload shift detection, it comprises the comparison of the classified DBS workload with the workload model. The workload modelling technique and model management concepts that have been developed for this purpose are described in the following Section 4.

4 Workload Analysis

After the monitored information has been classified according to Section 3, it is compared to a workload model in order to detect workload shifts. This section first presents the modelling technique chosen for the representation of DBS workload and afterwards describes the management of the workload model.

4.1 DBS Workload Modelling

Our approach for modelling the DBS workload is based on the idea of creating a model of the typical database application *behaviour*. This behaviour describes the interaction of the applications with the database, i.e. the SQL statements they submit and the context in which these appear. From another perspective, this can be seen as modelling the “language” the applications speak to the DBS.

Considering the goal of workload shift detection, it is not required to recognize a single new statement in the workload. Instead, significant changes in the overall composition of statements must efficiently be recognized. So statistical methods are applicable for this task. In our concept we have decided to use *n-Gram-Models*, which have already been successfully applied in other problem domains (e.g. bioinformatics and natural language processing). These models are learnable, offer a strong formal background with existing algorithms, and can be represented in a comprehensible, graphical way. n-Gram-Models are based on markov chains, which model the behaviour of an event source behaving according to a stochastic process. This stochastic process generates a sequence of events $\mathbf{X} = (X_1, X_2, \dots, X_t : t \in T)$, where each event X_t takes a value of $S = (s_1, s_2, \dots, s_k : k \in \mathbb{N})$. To be a markov chain, the probability of events must adhere to the markov property, i.e. it must not depend on the entire history of previous events, but only on a limited history of length m (*order*). From DBS perspective, database applications can be considered as a stochastic process generating a workload $\mathbf{W} = (W_1, W_2, \dots, W_t)$ as a sequence of SQL statements. To represent this workload in a markov chain model, each statement W_t must be mapped to an event X_t taking the value s_i . State s_i can be seen as the internal state of the application when generating the SQL statement at time t . We have investigated two alternative modelling approaches for this purpose:

- *Inference of Transactions*: All statements issued in the same transaction type are mapped to one model state. This approach results in a compact workload model with a small number of states. But as there usually is no information about the type of transaction in the workload \mathbf{W} , it is necessary to infer the transaction types from the observed statements. This is a complex task for procedurally controlled transactions [12], where the number and type of SQL statements in a transaction varies depending on loop/branch-conditions.
- *Statement Types*: In this approach we directly map each statement type to a markov model state. The transition probabilities are computed according to the ordering in \mathbf{W} . On the one hand this approach results in model with more states, while on the other hand we avoid the problem of transaction inference.

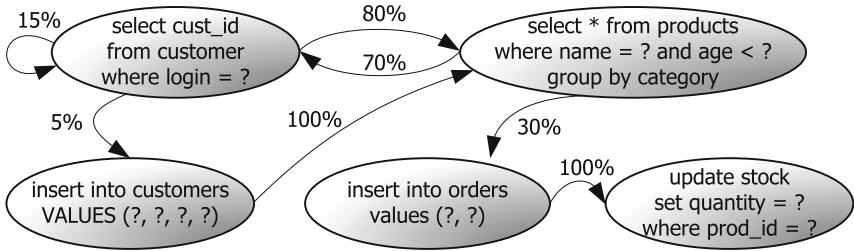


Fig. 3. Example for DBS-Workload modelled as Markov Chain of Order 1

For our concept of workload shift detection we have chosen the statement type mapping, because it produces less overhead than the inference of transactions (cf. [12]). In the processing model presented in Section 3 we have furthermore proposed a classification approach that already performs this task. In the following we therefore assume that the workload \mathbf{W} already is given in terms of statement types. So \mathbf{W} is equal to \mathbf{X} in our case and takes the values of the model states S . Fig. 3 shows an illustration of a markov-chain workload model using the statement type mapping. Each node of this graph represents a statement type. The probability of forthcoming events in this example depends only on the current state (markov chain of order 1). These *transitions* between states are illustrated as attributed edges.

As described above, the prerequisite for creating a markov-chain model is that the event source satisfies the markov property. For this reason, creating a markov-chain model requires thorough knowledge about the stochastic process and a manual modelling of its behaviour. In order to avoid this requirement, n-Gram-Models consider the markov property only as an approximation. Hence, the order of the markov chains determines how well the model approximates the underlying process. With this approximation, n-Gram-Models can automatically be learned from training data by counting the occurrences of events:

$$P(W_{t_i} | W_{t_{i-n}}, \dots, W_{t_{i-1}}) = \frac{\text{count}(W_{t_{i-n}}, \dots, W_{t_{i-1}}, W_{t_i})}{\text{count}(W_{t_{i-n}}, \dots, W_{t_{i-1}})}. \quad (1)$$

So for the creation and analysis of n-Gram-Models, only sub-sequences with the length n of the events \mathbf{W} are considered (n-Gram-Models result in markov-chains of order $n - 1$).

As n-Gram-Models are only an approximation of the real world process, all analysis on the model faces the problem of *unseen events*. Unseen events are events that have not been observed, either because the event W_t has not occurred yet at all, or because it has not been seen with a particular history $(W_{t-n}, \dots, W_{t-1})$ yet. To prevent these unseen events from being evaluated to a probability of 0, there are a number of techniques in literature ([13], [14]). For our evaluation described in Section 5 we have chosen a combination of *absolute discounting* and *backing off* for this purpose.

4.2 Workload Shift Detection

Having introduced the concepts of DBS workload modelling, we will now describe how workload shift detection is realized on this basis. For this purpose it is essential to assess how well the observed DBS workload is described by the workload model. In statistics, the metric commonly used to assess the quality of a probability model is the *perplexity*. Informally, the perplexity describes how “surprised” the model is by the observed workload. For an observed workload \mathbf{W} , the perplexity PP of an DBS workload n-Gram-Model is computed as

$$PP(\mathbf{W}) = \left(\prod_{t=1}^T P(W_t | W_{t-n} \dots W_{t-1}) \right)^{-\frac{1}{T}}. \quad (2)$$

For the computation of the perplexity, the probabilities $P(W_t)$ of the individual SQL statements are taken from the workload model. Using the perplexity PP as a metric, the decision of whether or not the model sufficiently well describes the actual DBS workload results in choosing a threshold for the perplexity. The appropriate value for the perplexity threshold depends on the installation-specific workload characteristics. From Equation 2 it is obvious that the expected perplexity for a completely random workload is equal to the number of statement classes. Procedurally controlled workloads result in lower perplexities. In our evaluation (Section 5) a perplexity value of $1,5 * |S|$ has produced good results for a mixed workload. It is important to note that the perplexity definition only implicitly considers obsolete workload: A workload model will determine low perplexity values for a workload even if there are many obsolete states in the model.

In order to realize workload shift detection for an autonomic database, the workload model must be maintained without any user interaction. Fig. 4 illustrates the lifecycle that is used to automatically manage the workload model in our approach. As the goal of autonomic databases is the reduction of the maintenance costs, creating an initial workload model from a set of manually provided training data is not suitable. Instead, the workload model must be automatically learned. The shift detection logic therefore always creates a new workload model in state “learning”. This state indicates that the model must still be trained. After the model has learned the typical DBS workload, it is switched to state “stable”. It remains in this state as long as the fluctuations in the DBS workload are small. When a significant change is detected the workload model switches to state “adapting”, where it must again be trained for the changed workload before returning to state “stable” again.

Considering the power of the selected n-Gram-Modelling approach and the above lifecycle, there remain two main challenges that must be addressed in order to realize workload shift detection: The decision on the end of the learning/adapting phase and the detection of outdated states and transitions in the model.

The decision on the end of the “learning” and “adapting” phases for a workload model can be made by monitoring the perplexity values. If the perplexity has not exceeded the perplexity threshold for a certain period of time, it can be assumed that the model is stable. The length of this period is determined

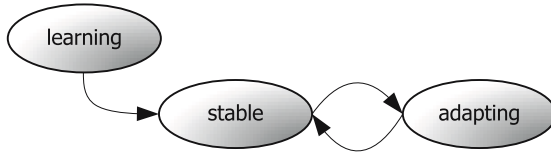


Fig. 4. Workload Model Lifecycle for DBS Workload Shift Detection

by the configurable length of short-term patterns (cf. Section 2). As according to our requirement definition these patterns should not be detected as workload shifts, they must be represented in the model. Requiring the perplexity to be strictly below the threshold for the entire pattern length may lead to long learning intervals, because a single probe that exceeded this threshold would restart the counter. Thus, we consider it sufficient if a certain percentage (e.g. 80%) of perplexity values in the past short-term pattern period length has had values below the threshold (stability factor).

Whereas the perplexity metric provides a good indicator for new events or a different event composition, it is not meaningful in case of obsolete events. Hence, we have included the following ageing mechanism in the workload model management: Every transition in the model is attributed with a timestamp, which indicates the most recent observation of the transition. When the age of a transition has exceeded a certain age, their information is removed from the model. For a consistent assessment of deviations from the workload changes, we again utilizes the perplexity metric. We generate an artificial sample probe from the workload model before the removal of model elements, and then compute the perplexity for this sample probe with the new, adapted model. If the perplexity value exceeds the threshold, a workload shift due to obsolete workload has occurred.

The overall workload shift detection logic is illustrated as pseudo code in Algorithm 4.1. In a first step, the shift detection logic converts the observed workload \mathbf{W} to n-Grams (1). Every n-Gram consists of the observed event (statement type) and the history of $n-1$ preceding events, where $n-1$ is given as a predefined value `CHAIN_LENGTH` (1). Based on these n-Grams, it computes the perplexity for the current model according to Equation 2 (2). The subsequent actions depend on the state of the model: If the model is in state “learning” or “adapting”, then the transition probabilities in the model are recalculated according to Equation 1 for all probes that exceed the perplexity threshold `THRESH` (17). In addition, new states are added to the model for previously unknown statements in this step. Afterwards the workload shift detection logic checks whether the required percentage `STAB` of probes in the past short-term pattern period (of length `SHORT`) has taken perplexity values below the threshold (18). If this is the case, then the model is switched to state “stable” (21). A workload shift alarm is raised (19-20) now that the workload has reached a stability level again.

¹ Typical values for the chain lengths in n-Gram-Models are 2 and 3.

Algorithm 4.1. Model Management for Workload Shift Detection**Algorithm:** detectShift**Input:** an observed workload \mathbf{W} , a workload model $model$

```

1 nGrams  $\leftarrow$  convertToNGrams( $\mathbf{W}$ , CHAIN_LENGTH);
2 pp  $\leftarrow$  model.computePerplexity(nGrams);
3 hist  $\leftarrow$  hist.add(pp > THRESH);
4 if model.state = "stable" then
5     if pp > THRESH then
6         if hist.cntStable(SHORT) < (STAB * hist.countAll(SHORT)) then
7             model.reset();
8             model.state  $\leftarrow$  "adapting";
9     else
10        if model.hasOutdatedStatesAndTransitions(2 * SHORT) then
11            probeNGrams  $\leftarrow$  model.generateProbe();
12            model.removeOutdatedStatesAndTransitions(SHORT);
13            if model.computePerplexity(probeNGrams) > THRESH then
14                raiseAlert("DBS Workload Shift. Reconfiguration required.");
15 if model.state = "learning" or model.state = "adapting" then
16     if pp > THRESH then
17         model.learnFrom(nGrams);
18     else if hist.cntStable(SHORT) > (STAB * hist.cntAll(SHORT)) then
19         if model.state = "adapting" then
20             raiseAlert("DBS Workload Shift. Reconfiguration required.");
21             model.state  $\leftarrow$  "stable";
22         if model.state = "adapting" then
23             model.removeOutdatedStatesAndTransitions(SHORT);

```

In contrast, while the model is in state “stable”, the model is not adapted at all. That is, even if there are differences between the observed workload and the model, the transition probabilities and states are not updated. The workload shift detection logic solely checks whether the percentage STAB of perplexity values has exceeded the threshold, and changes the model state to “adapting” in this case (5-8). Afterwards, the ageing mechanism described above is applied to the model (10-14).

5 Evaluation

For the evaluation of our concepts we have created the testbed illustrated in Fig. 5. Two sample applications (DVD-Shop and Reporting) are used to generate workload on a DBS. The DVD-Shop is a simple web shop based on a sample application by Dell Labs [15], whereas the Reporting generates custom reports from the shop data. As a LoadGenerator for the applications we have used Apache JMeter. The workload information is polled from the DBS by a prototypical implementation of the workload shift detection (WSD Prototype) in regular intervals. In this prototype we have realized the processing model as described in Section 3. All source-specific preprocessing steps (i.e. monitoring and filtering)

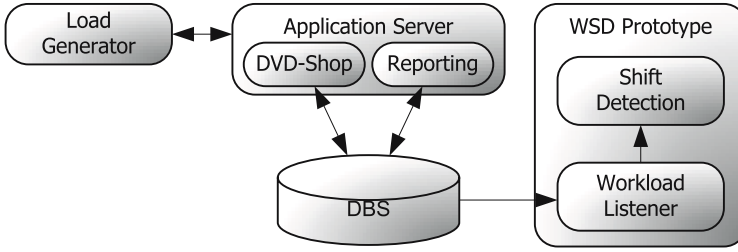


Fig. 5. Testbed for Evaluation of DBS Workload Shift Detection Prototype

of the workload information are encapsulated in a `WorkloadListener` component. The observed workload is then passed to the `ShiftDetection` component, where it is classified (according to the statement types) and analysed.

In order to achieve an efficient implementation of the n-Gram-Model in the `ShiftDetection` component, we have followed the approach proposed in [10]: To store the transition probabilities, we use a suffix tree which only stores the events and histories actually seen. Compared to using a transition matrix, this approach leads to a very compact model. To prevent the probability from taking a value of 0 for previously unseen events we have implemented a combination of the techniques *absolute discounting* and *backing off* [13].

The evaluation of our workload shift detection approach is based on the requirements defined in Section 2. Hence we have first defined the workload listed in Table 1a. The workloads `Shop1` and `Shop2` represent different user behaviours on the web shop application, whereas `Rep1` and `Rep2` simulate different releases of the reporting application (where the second release has an increased functionality, i.e. it offers twice the number of reports). From these workloads we have then composed the test cases TC1-TC9, which are described in Table 1b. The test cases cover the shift scenarios as defined in Section 2 (New Applications, Obsolete Applications, Modified Applications, Application Usage Changes, Long-Term-Patterns) and the requirements of resilience to noise and automatic learning/adapting.

We have executed the test cases TC1-TC9 in the testbed described above. For each of the test cases, the perplexity-values are plotted in Fig. 6. These values have been computed with a probe size of 100 statements on a 3-Gram-Model. In our tests, the maximum length for short-term patterns `SHORT` was defined as 180 seconds, the stability factor `STAB` as 0.8. The perplexity threshold `THRESH` was set to 30, which was 50% above the number of statement types in the test. For each test case Fig. 6 also denotes the changes in the model state for this parametrization (S denotes a change to state “stable”, A a change to state “adapting”, D for a deletion of outdated model states).

The results in Fig. 6 prove that the model is reliably learned from the observed workload (TC1), and that it automatically adapts quickly in case of usage changes (TC3). Minor fluctuations or short-term patterns in the workload (caused by the probabilistic workload generation by JMeter and explicitly

Table 1. Definition of Test Scenarios

(a) Workloads					(b) Test Cases			
Workl.	Action	Rep.	TT	P.	Scenario	Workl.	Users	Interv.
Shop1	1. Login	1	5-10s	1.0	TC1: Learning	Shop1	10	[0;400]
	2. Search	1	0s	1.0	TC2: R. to Noise	Shop1	10	[0;400]
	3. AddToCart	1	0s	1.0		Shop2	1	[200;300]
	4. Checkout	1	0s	1.0	TC3: Adaptation	Shop1	10	[0;200]
	5. Confirm	1	0s	1.0		Shop2	10	[200;400]
Shop2	1. Login	1	5-10s	0.5	TC4: New App.	Shop1	10	[0;400]
		1	0-5s	0.5		Rep1	10	[200;400]
	2. Search	10-20	0s	1.0	TC5: Obsol. App.	Shop1	10	[0;800]
	3. AddToCart	1	0s	1.0		Rep1	10	[0;200]
	4. Checkout	1	0s	1.0	TC6: Mod. App.	Rep1	10	[0;200]
5. Confirm	1	0s	1.0	Rep2		10	[200;400]	
Rep1	1. Report 1	1	0-5s	0.2	TC7: Usage Ch.	Shop1	10	[0;400]
	⋮	⋮	⋮	⋮		Rep1	2	[0;200]
Report 5	1	0-5s	0.2	Rep1		10	[200;400]	
Rep2	1. Report 1	1	0-5s	0.1	TC8: Long Patt.	Shop1	10	[0;400]
	⋮	⋮	⋮	⋮		Rep1	10	[400;800]
	⋮	⋮	⋮	⋮		Shop1	10	[800;1200]
	Report 10	1	0-5s	0.1		Rep1	10	[1200;1600]
Rep1	⋮	⋮	⋮	⋮	TC9: Short Patt.	Shop1	10	[0;100]
						Rep1	10	[100;200]
						Shop1	10	[200;300]
	⋮	⋮	⋮					
	Rep1	10	[1500;1600]					

induced in TC3 and TC9), do not cause a model instability. In contrast, all shift scenarios (TC4, TC5, TC6, TC7, TC8) are reliably detected as workload shifts.

The computation times for the perplexity of a probe and for the adaptation of a probe depend on the probe size, the number of statement classes and the markov chain length. Fig. 7 illustrates the computation times on a desktop workstation (clock cycle: 3 GHz/ memory:1 GB) for different chain lengths m . As the probe size can be considered as a scaling constant (cf. Equation 2), it has been set to a fixed value of 100. The results show that the proposed workload shift detection is very efficient while the model is stable, because even with a long chain length $m = 10$ the computation of the perplexity is always less than $50ms$. The adaptation of a model in the “learning” or “adapting” states requires more overhead, but these adaptations will be necessary only very rarely.

6 Related Work

All major commercial DBMSs today offer advisors, which recommend a good physical design for a given workload ([1], [4], [5]). Due to their heavy-weight analysis algorithms (e.g. knapsack) they are designed as offline-tools, and must be

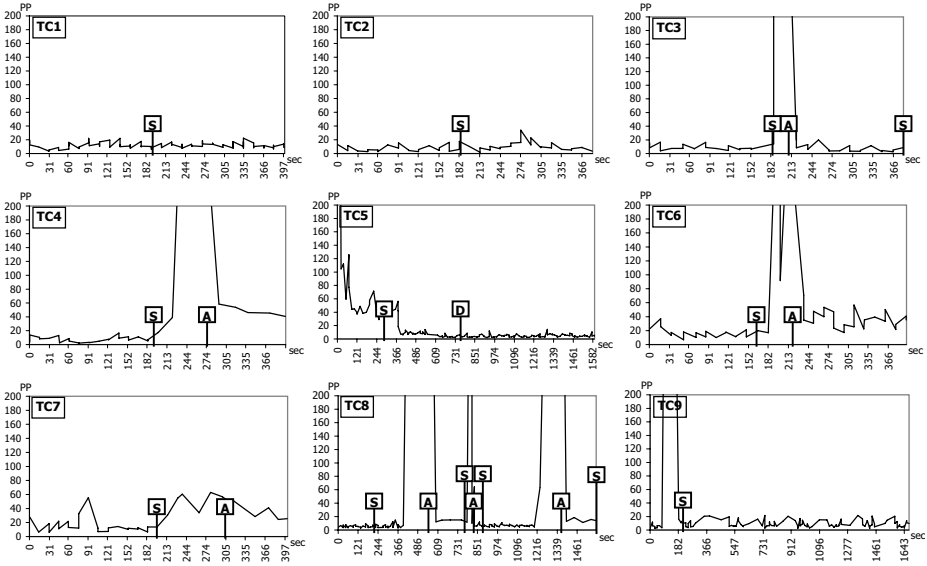


Fig. 6. Test Results for Test Cases TC1-TC9

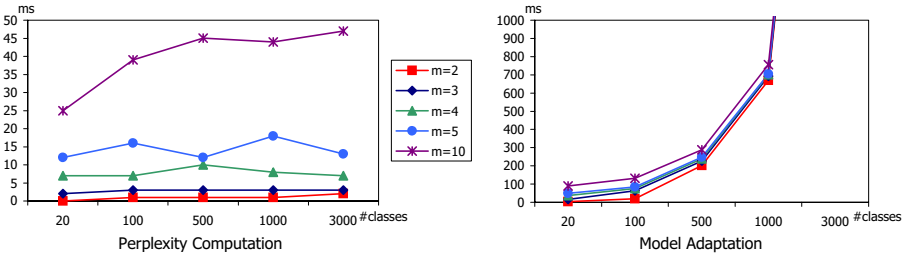


Fig. 7. Perplexity Computation Times for different Model Configurations

explicitly started by the DBA when he suspects possible improvements. Hence, these advisors are intended for the reconfiguration analysis rather than workload shift detection.

In contrast to the advisors, COLT [16] performs online workload monitoring and analysis. By using the database optimizer in a “what-if-mode”, COLT continuously determines the possible benefit of hypothetical additional indexes. In contrast to our approach, COLT focuses on the online adaptation of the physical design. It does not provide a general-purpose workload shift detection, but is intended as an online-tool for autonomic index management.

The online tuning approach [17] by Bruno and Chaudhuri gathers execution plan information from the optimizer during normal query processing. From this information, it can directly derive the execution costs for alternative physical designs without having to issue additional calls to the optimizer. So like COLT,

the approach in [17] focuses on the efficient online-adaptation of the physical design. Other changes to the configuration of the DBS, which may also be caused by workload shifts, are not considered.

Elnaffar et al. [18] propose the use of a workload model for the automatic classification of database workload in order to distinguish between OLTP and DSS workload. However, the workload model in their approach does not actually represent the database workload. It is instead a decision tree that is used for classification. Hence, their approach is limited on the detection of shifts between OLTP and DSS workloads. In a later work [19], the authors have also identified the general need for exploratory models (models of the monitored workload), but an approach for building them is not given.

N-Gram-Models have also been used in [12] for creating a statistical model of database workload. This work is directed at the inference of user sessions from SQL statement traces. Heavy-weight analysis algorithms are applied on these models in order to identify and cluster procedurally controlled sessions. The information about the user sessions is intended to be used for the prediction of queries based on the queries already submitted, e.g. to optimize the cache replacement strategies.

7 Conclusion

The workload has major influence on the configuration and optimization of a DBS. Hence, for a self-managing database systems, it is an essential prerequisite to continuously monitor and analyse the database workload. In this paper we have presented an approach based on n-Gram-Models, which detects shifts in the typical workload of a database. Its results provide a strong indicator for when a detailed reconfiguration analysis should be performed. Our evaluation has shown that the approach is very light-weight and can be used to perform continuous monitoring. In addition, it reliably detects significant changes and is resilient to noise.

In the future, we are going to extend our concept in two directions: On the one hand, long-term patterns should not only be detected as workload shifts, but should be reflected in the workload model. This extension will support the prediction of periodic changes in the workload and the proactive adaptation of the DBSs configuration. On the other hand, we are working on a more sophisticated classification algorithm than statement type identification. Our goal is a classification that allows the definition of a fixed (upper) limit of classes, thus providing an upper bound for the workload shift detection effort.

References

1. Agrawal, S., et al.: Database tuning advisor for Microsoft SQL Server 2005. In: Proceedings of the 30th International Conference on Very Large Data Bases, pp. 1110–1121. Morgan Kaufmann, San Francisco (2004)
2. Dageville, B., Dias, K.: Oracle's self-tuning architecture and solutions. IEEE Data Engineering Bulletin 29(3) (2006)

3. Ganik, Corbi.: The dawning of the autonomic computing era. *Ibm Systems Journal* 42(1), 5–18 (2003)
4. Dageville, B., et al.: Automatic SQL tuning in Oracle 10g. In: *Proceedings of the 30th International Conference on Very Large Data Bases*, pp. 1098–1109. Morgan Kaufmann, San Francisco (2004)
5. Zilio, D.C., et al.: Recommending materialized views and indexes with IBM DB2 design advisor. In: *Proceedings of the International Conference on Autonomic Computing*, pp. 180–188. IEEE Computer Society, Los Alamitos (2004)
6. Lahiri, T., et al.: The self-managing database: Automatic SGA memory management. White paper, Oracle Corporation (2003)
7. Storm, A.J., et al.: Adaptive self-tuning memory in DB2. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*, pp. 1081–1092. ACM, New York (2006)
8. Holze, M., Ritter, N.: Towards workload shift detection and prediction for autonomic databases. In: *Proceedings of the ACM first Ph.D. workshop in CIKM*, pp. 109–116. ACM Press, New York (2007)
9. Lightstone, S., et al.: Making DB2 products self-managing: Strategies and experiences. *IEEE Data Engineering Bulletin* 29(3) (2006)
10. Fink, G.: *Markov Models for Pattern Recognition*, 1st edn. Springer, Heidelberg (2008)
11. Parekh, S., et al.: Throttling utilities in the IBM DB2 universal database server. In: *Proceedings of the American Control Conference*, pp. 1986–1991. IEEE Computer Society, Los Alamitos (2004)
12. Yao, Q., et al.: Finding and analyzing database user sessions. In: Zhou, L., Ooi, B.C., Meng, X. (eds.) *DASFAA 2005. LNCS*, vol. 3453, pp. 851–862. Springer, Heidelberg (2005)
13. Katz, S.: Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35(3), 400–401 (1987)
14. Jelinek, F., Mercer, R.L.: Interpolated estimation of markov source parameters from sparse data. In: *Pattern Recognition in Practice*, pp. 381–397. North-Holland, Amsterdam (1980)
15. Dell Laboratories: Dell DVD store database test suite (2008), <http://linux.dell.com/dvdstore/>
16. Schnaitter, K., et al.: On-line index selection for shifting workloads. In: *Proceedings of the 23rd International Conference on Data Engineering Workshops*. IEEE Computer Society Press, Los Alamitos (2007)
17. Bruno, N., Chaudhuri, S.: An online approach to physical design tuning. In: *Proceedings of the 23rd International Conference on Data Engineering*, pp. 826–835. IEEE Computer Society Press, Los Alamitos (2007)
18. Elnaffar, S., Martin, P., Horman, R.: Automatically classifying database workloads. In: *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management*. ACM Press, New York (2002)
19. Martin, P., et al.: Workload models for autonomic database management systems. In: *Proceedings of the International Conference on Autonomic and Autonomous Systems*, p. 10. IEEE Computer Society, Los Alamitos (2006)

Large Datasets Visualization with Neural Network Using Clustered Training Data

Sergėjus Ivanikovas^{1,2}, Gintautas Dzemyda^{1,2}, and Viktor Medvedev^{1,2}

¹ Institute of Mathematics and Informatics,
Akademijos str. 4, LT-08663 Vilnius, Lithuania

² Vilnius Pedagogical University,
Studentų str. 39, LT-08106 Vilnius, Lithuania
Ivanikovas@gmail.com, Dzemyda@ktl.mii.lt, Viktor.m@ktl.mii.lt

Abstract. This paper presents the visualization of large datasets with SAMANN algorithm using clustering methods for initial dataset reduction for the network training. The visualization of multidimensional data is highly important in data mining because recent applications produce large amount of data that need specific means for the knowledge discovery. One of the ways to visualize multidimensional dataset is to project it onto a plane. This paper analyzes the visualization of multidimensional data using feed-forward neural network. We investigate an unsupervised backpropagation algorithm to train a multilayer feed-forward neural network (SAMANN) to perform the Sammon's nonlinear projection. The SAMANN network offers the generalization ability of projecting new data. Previous investigations showed that it is possible to train SAMANN using only a part of analyzed dataset without the loss of accuracy. It is very important to select proper vector subset for the neural network training. One of the ways to construct relevant training subset is to use clustering. This allows to speed up the visualization of large datasets.

1 Introduction

The research area of this work is the analysis of large multidimensional datasets. Visualization techniques are especially relevant to multidimensional data, the analysis of which is limited by human perception abilities. Visualization is a useful tool for data analysis, especially when the data is unknown. However, when the dimension is huge, to produce robust visualization is difficult. Therefore, the dimensional reduction technique is needed. The goal of the projection method is to represent the input data items in a lower-dimensional space so that certain properties of the structure of the data set were preserved as faithfully as possible. The projection can be used to visualize the data set if a sufficiently small output dimensionality is chosen. A deep review of the dimensionality reduction methods is performed in [4].

This paper focuses on dimensionality reduction methods as tool for the visualization of large multidimensional datasets.

Several approaches have been proposed for reproducing nonlinear higher-dimensional structures on a lower-dimensional display. The most common methods allocate a representation of each data point in a lower-dimensional space and try to optimize these representations so that the distances between them are as similar as possible to the original distances of the corresponding data items. The methods differ in that how the different distances are weighted and how the representations are optimized. Multidimensional scaling (MDS) refers to a group of methods that is widely used [3]. Nowadays multidimensional scaling means any method searching for a low (in particular two) dimensional representation of multidimensional data sets. The starting point of MDS is a matrix consisting of pairwise dissimilarities of the data vectors. The goal of projection in the metric MDS is to optimize the representations so that the distances between the items in the lower-dimensional space would be as close to the original distances as possible. Denote the distance between the vectors X_i and X_j in the feature space R^n by d_{ij}^* , and the distance between the same vectors in the projected space R^d by d_{ij} . In our case, the initial dimensionality is n , and the resulting one (denote it by d) is 2. The metric MDS tries to approximate d_{ij} by d_{ij}^* . Usually the Euclidean distances are used for d_{ij} and d_{ij}^* . A particular case of the metric MDS is Sammon's mapping [14]. It tries to optimize a cost function that describes how well the pairwise distances in a data set are preserved:

$$E_S = \frac{1}{\sum_{i,j=1;i < j}^m d_{ij}^*} \sum_{\substack{i,j=1 \\ i < j}}^m \frac{(d_{ij}^* - d_{ij})^2}{d_{ij}^*}.$$

E is commonly mentioned as Sammon's stress (projection error). It is a measure of how well the interpattern distances are preserved when the vectors are projected from a high-dimensional space to a low-dimensional one. Sammon used a gradient steepest descent procedure (diagonal Newton method) to find a configuration of m vectors in the n -dimensional space that minimizes E . A disadvantage of the original Sammon mapping is that when a new data point has to be mapped, the whole mapping procedure has to be repeated. The application of Sammon mapping becomes impractical for large m .

Nonlinear visualization methods are computationally very intensive for large data sets. The original MDS algorithms are not appropriate for large scale applications because they require an entire $m \times m$ distance matrix to be stored in memory and may have $O(m^3)$ complexity. Thus, the MDS method is unsuitable for large datasets, it takes much computing time or there is not enough computing memory. However, in the last 10 years, several scalable MDS algorithms have been proposed. For example, Faloutsos and Lin [5] proposed FastMap, which is an MDS method that determines one coordinate at a time by examining a constant number of rows of the distance matrix. Wang, et. al [16] proposed an improvement on FastMap, called MetricMap, which attempts to do the entire projection at once. de Silva and Tennenbaum [15] proposed Landmark

MDS (LMDS) as another attempt at scalable MDS. An original scheme providing very accurate layouts of large datasets is introduced in [12].

The combination and integrated use of data visualization methods of a different nature are under a rapid development. The combination of different methods can be applied to make data analysis, while minimizing the shortcomings of individual methods. Artificial neural networks (ANN) may be used for reducing dimensionality and data visualization. A feed-forward neural network is utilized to effect a topographic, structure-preserving, dimension-reducing transformation of the data, with an additional facility to incorporate different degrees of associated subjective information. The MDS got some attention from neural network researchers [11], [9], [17].

In this paper, we focus on SAMANN, as tool for the visualization of large multidimensional datasets.

2 Multidimensional Data Visualization Using Feed-Forward Neural Network

There is no mechanism to project one or more new data points without expensively regenerating the entire configuration from the augmented data set in Sammon's algorithm. To project new data, one has to run the program again on all the data (old data and new data). To solve this problem, some methods (triangulation algorithm [8], [2], standard feed-forward neural network [13]) have been proposed.

Mao and Jain [11] have suggested a neural network implementation of Sammon's mapping. A specific back propagation-like learning rule has been developed to allow a normal feed-forward ANN to learn Sammon's mapping in an unsupervised way. In Mao and Jain's implementation the network (SAMANN) is able to project new vectors after training – a property Sammon's mapping does not have. A drawback of using SAMANN is that it is rather difficult to train and the training process is extremely slow. It has been concluded in [11] that the SAMANN network preserves the data structure, cluster shape, and interpattern distances better than a number of other feature extraction or data projection methods.

The architecture of the SAMANN network is a multilayer perceptron where the number of input vectors is set to be the input space dimension, n , and the number of output vectors is specified as the projected space dimension, d .

Mao and Jain have derived a weight updating rule for the multilayer perceptron [11] that minimizes Sammon's error (projection error E_S), based on the gradient descent method. The general updating rule for all the hidden layers, $l = 1, \dots, L - 1$ and for the output layer ($l = L$) is:

$$\Delta\omega_{jk}^{(l)} = -\eta \frac{\partial E_S(\mu, \nu)}{\partial \omega_{jk}^{(l)}} = -\eta(\Delta_{jk}^{(l)}(\mu)y_j^{(l-1)}(\mu) - \Delta_{jk}^{(l)}(\nu)y_j^{(l-1)}(\nu)), \quad (1)$$

where ω_{jk} is the weight between the unit j in the layer $l - 1$ and the unit k in the layer l , η is the learning rate, $y_j^{(l)}$ is the output of the j th unit in the layer l , and μ

and ν are two vectors from the analysed data set $\{X_1, X_2, \dots, X_m\}$. The $\Delta_{jk}^{(l)}$ are the errors accumulated in each layer and backpropagated to a preceding layer, similarly to the standard backpropagation. The sigmoid activation function with the range $(0, 1)$ is used for each unit.

The network takes a pair of input vectors each time in the training. The outputs of each neuron are stored for both points. The distance between the neural network output vectors can be calculated and an error measure can be defined in terms of this distance and the distance between the points in the input space. The weights of neural network are updated according to the update rule (II) using the error measure. After training, the network is able to project previously unseen data generalising the mapping rule.

3 Clustering Methods for Preparing the Training Dataset

It has been noticed that the SAMANN network training depends on different parameters. The results of the experiments showed [6] that it is possible to find such a subset of the analyzed dataset that while training the SAMANN network with this subset lower projection errors are obtained faster than by training with all the points of the set.

Proper way to create the training subset is clustering of the analyzed dataset. Clustering is a common technique used in understanding and manipulating large datasets. Generally, clustering can be defined as follows: given m data points in a n -dimensional metric space, partition the data points into k clusters such that the data points within a cluster are more similar to each other than data points in different clusters. Clustering algorithms should concern about the following issues: 1) the definition of similarity of data items; 2) the characteristics of clusters, including size, shape and statistical properties; 3) the computation cost and error rate of the result.

Using clustering for preparing the training subset, the neural network training is divided into the two stages: clustering of the analyzed dataset and the neural network training using only clustered data. Sometimes before applying any clustering or whatever data mining technique, it is useful and also necessary to preprocess the data. The preprocessing consists of the data normalization. Two different clustering methods have been used for the analyzed dataset reduction: k-means and SOM.

K-means [10] seeks to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new

centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. Finally, this algorithm aims at minimizing such objective function $J = \sum_{j=1}^k \sum_{i=1}^m \|X_i^{(j)} - C_j\|^2$, where $\|X_i^{(j)} - C_j\|^2$ is a chosen distance measure between a vector $X_i^{(j)}$ and the cluster centre C_j , is an indicator of the distance of the m data points from their respective cluster centres.

Self-Organizing maps (SOM), first introduced by Kohonen, are used to organize unstructured data much like the k-means clustering approach. SOM-based algorithms can generate clusters from raw data. They can also produce lower dimensional projections of high dimensional data.

The Self-Organizing Map (SOM) is an ANN model consisting of a regular grid of processing units, “neurons” [7]. A model of some multidimensional observation, eventually a vector consisting of features, is associated with each unit. An observation vector of the same dimensionality as that of the models is compared with all the model vectors. The map attempts to represent all the available observations with optimal accuracy using a restricted set of models. In unsupervised learning, the models become ordered inside the grid so that similar models are close to each other and dissimilar models lie far from each other.

Using clustering for the analyzed dataset reduction, the obtained training subset for SAMANN network consists of new vectors. Using k-means method, training subset is constrained by clustering the initial dataset into k clusters. Cluster centroids are used to train the network and projection error is calculated using initial dataset. Using SOM for the training subset construction the size of the map is defined and SOM is trained using initial dataset. Then we use SOM vectors-winners for SAMANN network training. Projection error is calculated using initial dataset (i.e., all analyzed vectors) as previously.

In this work real datasets were used to investigate the ability to visualize large dataset using SAMANN. The SAMANN network was trained by a subset of the analyzed dataset to speed up the training process and to improve the precision.

Three real datasets have been used in the experiments:

Dataset1. Page blocks classification. This dataset is taken from UCI Machine Learning Repository [1]. The dataset consists of 5473 10-dimensional vectors from five classes. The task is to classify all the blocks of the page layout of a document that has been detected by a segmentation process.

Dataset2. Pen-Based recognition of handwritten digits dataset [1]. The dataset consists of 10992 16-dimensional vectors from ten classes. The classes are pen-based handwritten digits with ten digits 0 to 9.

Dataset3. MAGIC Gamma telescope dataset [1]. The dataset consists of 19020 10-dimensional vectors from two classes. The data set was generated by a Monte Carlo program. The data are Monte Carlo generated to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique.

In the analysis of the neural network training using clustered data, a particular case of the SAMANN network was considered: a feed-forward neural network

with one hidden layer and two outputs ($d=2$). During all the experiments, the same number of neurons of the hidden layer was.

The projection errors were calculated and the calculation time was measured using reduced training subset and whole analyzed dataset for the SAMANN network training.

4 Experiments on Real Large Datasets

Several experiments have been performed for all analyzed datasets using k-means and SOM clustering methods. The results of clustering have been used to create the reduced training dataset. Reduced dataset consists of centroids in case of k-means clustering and vectors-winner in case of SOM.

Working with each dataset three experiments have been drawn out. The neural network has been trained by the initial dataset (all vectors) and two reduced training datasets. In all experiments the projection error has been calculated using the initial dataset. The first reduced dataset has been created using k-means clustering. The second reduced training dataset has been created using SOM. The size of all the reduced training datasets was 10 times smaller than the size of the original dataset. Running all the experiments the run time of the program has been limited to approximately 20 hours.

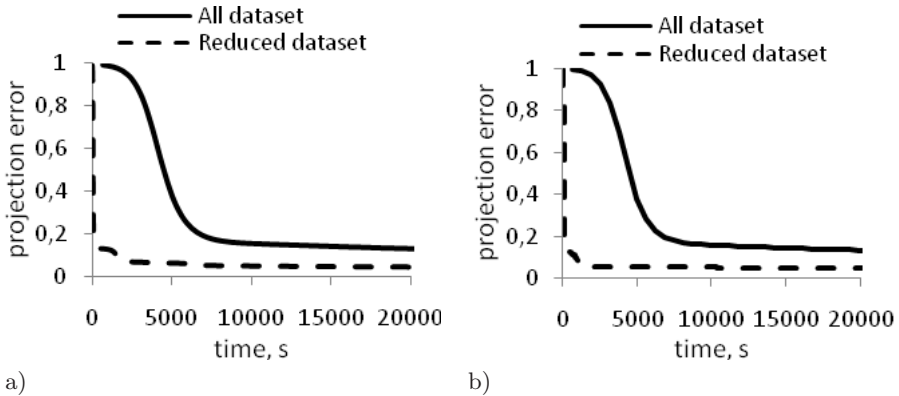


Fig. 1. The dependence of the projection error on the computation time for the *Dataset1*, using (a) SOM method and (b) k-means method

Working with *Dataset1* the training process using full dataset took 20.04 hours. 116 training iterations were accomplished. The projection error was 0.1183. Using k-means clustering for creating reduced training subset (approx. 10% of initial dataset) 7437 training iterations were accomplished during 20 hours. The projection error was 0.0365. Using SOM for creating reduced training subset 24x24 SOM net was created. 559 vectors-winners were obtained from this net. The reduced training subset was constructed from these vectors. 7860

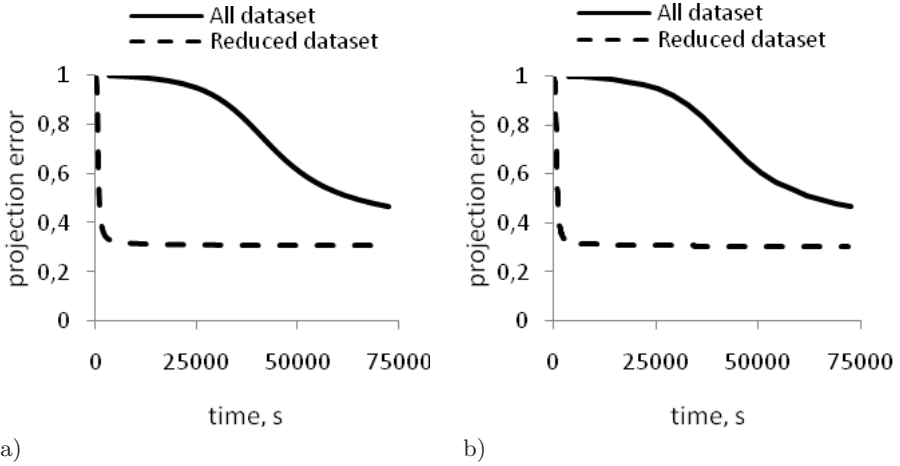


Fig. 2. The dependence of the projection error on the computation time for the *Dataset2*, using (a) SOM method and (b) k-means method

training iterations were accomplished during 20 hours working with this reduced training set. The projection error was 0.036. The results of the experiments working with *Dataset1* are presented in Fig. 1. Using both clustering methods for training dataset reduction better results were obtained faster than while working with all the dataset for SAMANN training.

Working with *Dataset2* the training process using full dataset took 20.18 hours. Only 20 training iterations were accomplished. The projection error was 0.466. Using k-means clustering for creating reduced training subset (approx. 10% of initial dataset) 1418 training iterations were accomplished during 20 hours. The projection error was 0.303. 34x34 SOM net was created reducing the

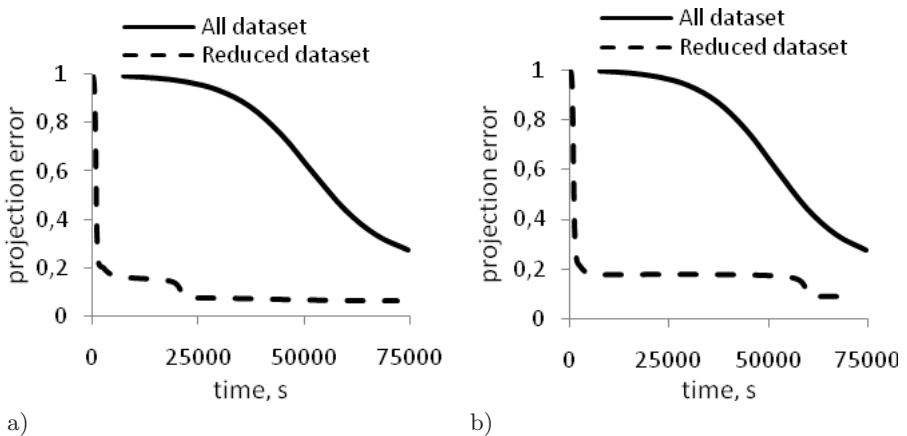


Fig. 3. The dependence of the projection error on the computation time for the *Dataset3*, using (a) SOM method and (b) k-means method

training subset with SOM. 1068 vectors-winners were obtained from this net. 1482 training iterations were accomplished during 20 hours using this training set. The projection error was 0.308. The results of the experiments working with *Dataset2* are presented in Fig. 2.

Working with *Dataset3* the training process using full dataset took 20.65 hours. Only 10 training iterations were accomplished. The projection error was 0.2758. Using k-means clustering for creating reduced training subset (approx. 10% of initial dataset) 623 training iterations were accomplished during 20 hours. The projection error was 0.086. Using SOM for creating reduced training subset 45x45 SOM net was created. 2015 vectors-winners were obtained from this net. 618 training iterations were accomplished during 20 hours while training the SAMANN with this training set. The projection error was 0.063. The results of the experiments working with *Dataset3* are presented in Fig. 3.

The results of using training subsets reduced with k-means and SOM methods are almost the same. The advantage of SOM is that this method is faster than k-means clustering method. Sometimes the results of training the SAMANN using training set reduced with k-means gives better projection error.

The comparison of the results obtained while using full dataset for SAMANN network training and while working with reduced datasets are presented in

Table 1. The comparison of the experimental results

Datasets	SAMANN (all dataset)	SAMANN (reduced dataset)	
	projection error	proj. error (k-means)	proj. error (SOM)
<i>Dataset1</i>	0.1559	0.0525	0.0476
<i>Dataset2</i>	0.9898	0.3111	0.3141
<i>Dataset3</i>	0.9862	0.1755	0.1570

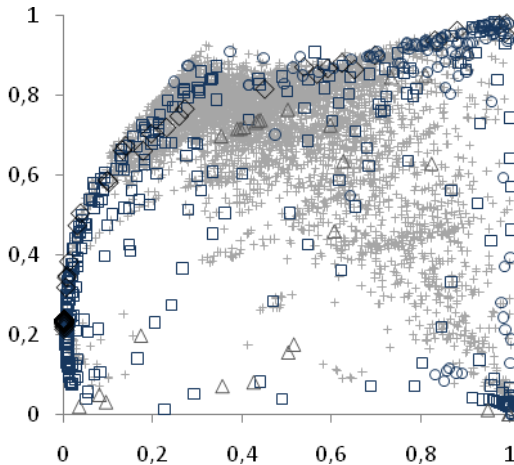


Fig. 4. Mapping results of the *Dataset1* using SOM method for reducing training dataset. Five classes are presented by different shapes.

Table 1. The value of the projection error was taken after a fixed time (approx. 3 hours) of program execution. The visualization results of *Dataset1* are presented in Fig.4.

5 Conclusions

It has been noticed that it is possible to speed-up the SAMANN network training process using a part of analyzed dataset. The results of the experiments proved that it is possible to find such a subset of the analyzed dataset that while training the SAMANN network with this subset lower projection errors are obtained faster (more than 10 times) than by training with all the vectors of the set.

The usage of the reduced dataset for SAMANN training enabled us to process large datasets and to get good enough results within reasonable time.

Using clustering for a constructing of the training subset (i.e., for the initial dataset reducing) it is important to choose proper size of the subset. The experiments indicated that using approximately 10% of the analyzed dataset as amount of vectors for training subset a better projection error is obtained and neural network training process is speeded-up comparing with initial dataset. The usage of smaller training subsets gives different result depending on a dataset.

The experiments showed that rather similar results are obtained using different clustering methods. Further investigation should be performed in this area.

Acknowledgements

The research is partially supported by the Lithuanian State Science and Studies Foundation project “Information technology tools of clinical decision support and citizens wellness for e.Health system” (No. B-07019).

References

1. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine (2007), <http://www.ics.uci.edu/~mllearn/MLRepository.html>
2. Biswas, G., Jain, A.K., Dubes, R.C.: Evaluation of Projection Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3(6), 701–708 (1981)
3. Borg, I., Groenen, P.: *Modern Multidimensional Scaling: Theory and Applications*. Springer, Heidelberg (1997)
4. Dzemyda, G., Kurasova, O., Medvedev, V.: Dimension Reduction and Data Visualization Using Neural Networks. In: *Emerging Artificial Intelligence Applications in Computer Engineering – Real World AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. *Frontiers in Artificial Intelligence and Applications*, vol. 160, pp. 25–49. IOS Press, Amsterdam (2007)
5. Faloutsos, C., Lin, K.: FastMap: a fast algorithm for indexing, datamining, and visualization. In: *Proc. ACM SIGMOD*, pp. 163–174 (1995)

6. Ivanikovas, S., Medvedev, V., Dzemyda, G.: Parallel Realizations of the SAMANN Algorithm. In: Beliczynski, B., Dzielinski, A., Iwanowski, M., Ribeiro, B. (eds.) ICANNGA 2007. LNCS, vol. 4432, pp. 179–188. Springer, Heidelberg (2007)
7. Kohonen, T., Oja, E.: Visual Feature Analysis by the Self-Organising Maps. *Neural Computing & Applications* 7(3), 273–286 (1998)
8. Lee, R.C.T., Slagle, J.R., Blum, H.: A Triangulation Method for Sequential Mapping of Points from n-Space to Two-Space. *IEEE Transactions on Computers* 27, 288–299 (1977)
9. Lowe, D., Tipping, M.E.: Feed-forward Neural Networks and Topographic Mappings for Exploratory Data Analysis. *Neural Computing and Applications* 4, 83–95 (1996)
10. MacQueen, J.B.: Some Methods for classification and Analysis of Multivariate Observations. In: Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297. University of California Press, Berkeley (1967)
11. Mao, J., Jain, A.K.: Artificial Neural Networks for Feature Extraction and Multivariate Data Projection. *IEEE Trans. Neural Networks* 6, 296–317 (1995)
12. Naud, A.: An Accurate MDS-Based Algorithm for the Visualization of Large Multidimensional Datasets. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 643–652. Springer, Heidelberg (2006)
13. de Ridder, D., Duin, R.P.W.: Sammon’s Mapping Using Neural Networks: A comparison. *Pattern Recognition Letters* 18, 1307–1316 (1997)
14. Sammon, J.W.: A Nonlinear Mapping for Data Structure Analysis. *IEEE Transactions on Computers* 18, 401–409 (1969)
15. de Silva, V., Tenenbaum, J.B.: Global versus local methods in nonlinear dimensionality reduction. In: Becker, S., Thrun, S., Obermayer, K. (eds.) Proc. NIPS, vol. 15, pp. 721–728 (2003)
16. Wang, J.T.-L., Wang, X., Lin, K.-I., Shasha, D., Shapiro, B.A., Zhang, K.: Evaluating a class of distance-mapping algorithms for data mining and clustering. In: Proc. ACM KDD, pp. 307–311 (1999)
17. van Wezel, M.C., Kusters, W.A.: Nonmetric Multidimensional Scaling: Neural networks Versus Traditional Techniques. *Intelligent Data Analysis* 8(6), 601–613 (2004)

Efficient Execution of Small (Single-Tuple) Transactions in Main-Memory Databases

Heine Kolltveit and Svein-Olaf Hvasshovd

Department of Computer and Information Science
Norwegian University of Science and Technology

Abstract. Various applications impose different transactional loads on databases. For example for telecommunication systems, online games, sensor networks, and trading systems, most of the database load consists of single-tuple reads and single-tuple writes. In this paper, approaches to handle these single-tuple transactions in main-memory systems are presented. The protocols are evaluated by simulation and verified by statistical analysis. The results show that 70 - 150% more transactions can be executed while keeping response times low using the new approach, compared to a state-of-the-art protocol.

1 Introduction

In recent years, there has been a gigantic increase in subscribers of mobile services. From late 2005 to November 2007, there was an increase from 2.14 billion to 3.3 billion subscriptions worldwide [1,2]. This is a staggering 50% increase in less than two years. In addition, prices for usage keeps dropping, increasing the usage by every subscriber [3]. Telecommunication (telecom) systems must facilitate the increasing load. For instance, the *Home Location Registry* is the central database in any mobile network. Its most important task is to manage subscriber mobility. Each time a mobile subscriber is called, the location of the subscriber is read, and each time a subscriber moves from one *Location Area* to another, the location of the subscriber is updated. Consequently, most of the load are single-record writes and single-record reads.

The most important requirements for telecom databases are [4]:

Reliability: No more than 30 seconds downtime per year, corresponding to a 10^{-6} probability of being out of service. This means no planned system downtime. Also, disaster failures such as earthquakes or fires must be masked.

Performance: Up to 10-20,000 transactions per second. Most of these are either read or write a single record.

Delays: Both write and read operations must have delays ranging from 5 to 15 milliseconds. It is also generally required that a certain percentage (e.g. 95%) of transactions finish within the time limit [5].

As a consequence of the delay requirement, and because disk accesses are slow, transactions cannot read from or write to disk. In contrast, fast main-memory accesses [6] should be used instead. However, since main memory is *volatile*, the reliability requirement cannot be met unless data is *replicated* and kept at multiple nodes.

Normal SQL queries have multiple interactions with the client. For instance, a user making an online reservation will first issue a search, then reserve one or more items, before finally completing the transaction. In contrast, the common single-tuple accessing transactions in telecom system can be implemented as canned transaction or Persistent Stored Modules (PSM) [7]. A PSM is a precompiled transaction which is stored in the database management system, and the client simply invokes it with input parameters and the result is returned. PSM makes it easy to distinguish the transaction type. Since the simple transactions are the predominant transactional load on the telecom database, their execution should be optimized.

This work uses the fact that the type and size of transactions may be known at execution time for PSMs. Therefore, the semantic differences between read-only transactions (queries) and write-only transactions (updates) can be used to reduce the delay of these transactions and improve the performance of the system, without affecting its overall reliability.

The motivation of this work is the high load on transactional systems with real-time requirements as the ones presented for telecom systems above. The main contributions of this work are main-memory transaction execution and commit protocols where the differences between single-record updates and single-record queries are exploited to increase the throughput and reduce the delay. The results of this work can be favorable used in telecom and other systems with similar loads and requirements for reliability, performance, and delays. Examples include online games, trading systems and sensor networks.

The rest of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 presents the properties of the two types of transactions, and how to handle them, while Section 4 proves the correctness. Section 5 outlines the simulation model and parameters used for the simulations in Section 6. The simulations are compared to a statistical analysis in Section 7 while Section 8 gives some concluding remarks.

2 Related Work

This section presents important related work. First, approaches using the transaction type are presented. Then, the most relevant main-memory commit processing approaches are introduced.

2.1 Transaction Type

The idea of using the type of transaction to optimize transaction processing was proposed several decades ago. It uses the fact that the system can treat various transactions types in different ways. For instance, not all transactions require synchronization as strong as global locking. In [8], four synchronization levels are defined and protocols are proposed.

Multi-versioning is suggested to be used for long read-only transactions [9,10]. A read-only transaction can then use a consistent versioned copy of the database by reading old snapshot data. In this way, the interference between long read-only transactions and shorter and more urgent update transactions is minimized. However, it is not

suitable for all applications since data may be outdated. A special case of read-only transactions are *free queries* [11]. Free queries have no consistency or concurrency requirements, and can thus be executed without locking. Assuming that update transactions do not write to the database until commit time, an example of a transaction which is always a free query is one which reads only one record. Such a transaction will always be consistent [11].

In [12], an approach is presented where multi-versioning is used to execute write-only operations logically in the future. This is the opposite of the versioned read-only optimization. Standard write transactions read the value, adjust it, and then write it back. These are called *read-write* transactions. *Write-only* transactions [13][12], however, do not read the value before updating it. By distinguishing between these types of transactions and using version control, the interference between read-write and write-only transaction are minimized.

The read-only commit optimization [14] is based on the observation that read-only operations have no work to do during the first phase of the two-phase commit protocol. Hence, it can be skipped. The second phase must still be executed since the locks must be released. This optimization has little effect on one-phased protocols, since the first phase is included in the transaction execution. Ramamritham [15] divides the transaction processing load for real-time systems into three types of transactions:

1. Read-only transactions, which read data from a database and do not make any updates.
2. Write-only transactions, which write external data into the database.
3. Read-write transactions, which read data from a database, update it, and write it back.

The RODAIN database architecture [16][17] is designed to be used in telecommunication systems. It is a real-time object-oriented architecture of a database management system. In [16], it is argued that a telecommunication system should distinguish between three transaction types: (1) Short and frequent simple-read transactions, (2) long and frequent update transactions and (3) long and rare read and write transactions. These correspond to the three transaction types by Ramamritham presented above. However, the large volume of short simple-update transactions generated by the mobility of the subscribers seem to be overlooked, and although distribution of data across a cluster is mentioned, the discussion is only for a single primary-backup pair. How to do global commit is not discussed. Also, the approach relies on the disk at the backup node. In addition, if the primary asks for the acknowledgement again (i.e. the message has disappeared), the backup may have deleted the transaction from the transaction table.

2.2 Main-Memory Commit Processing

This section introduces some existing approaches from previous work which are used in this paper. The first is the *Circular One-Phase Commit protocol* (C1PC) [18]. Figure 1(a) shows the failure-free execution of C1PC. When receiving a transaction request, the coordinator piggybacks a *Prepare* message on every *DoWork* to each of the participants. This instructs the participants to prepare the transaction and vote. Using C1PC, the primary participant sends the necessary log records to the backup

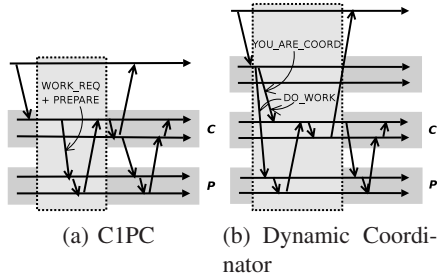


Fig. 1. Existing commit processing approaches and optimizations for main memory

participant, which sends its vote to the primary coordinator. The primary coordinator collects the vote and decides the outcome of the transaction. The outcome is sent to the backup coordinator which gives an *Early Answer* [5] to the client and notifies the primary participants. The decision is then forwarded to the backup participants. When the backup coordinator has received acknowledgements from all participants, it notifies the primary coordinator, and the transaction is completed.

Typically, the node where the client request arrives becomes the transaction coordinator. As each transaction arrives at a random node in the system, there is no guarantee that the coordinator is placed at a node with data accessed by the transaction. The dynamic coordinator approach was presented in [19]. In this approach, the transactional workload is split by the original coordinator, and sent to each participant (node holding data to be accessed). One of the participants are chosen as the new coordinator and a *YouAreCoordinator* flag is added to the message. The message to the other participants contains a *CoordinatorIs* variable which identifies the new coordinator. In this way, the number of messages and processing required during commit processing are reduced, since the number of participants are reduced by one.

Piggybacking and prioritizing can also be used. In [20], an approach where messages and tasks are prioritized depending on whether they are urgent or not is presented. A task is urgent if it is in the critical path before the early answer is sent to the client. Non-urgent tasks are only executed when there are no urgent tasks to do, and non-urgent messages are not sent until there are no urgent messages. To avoid starvation, however, messages and tasks which have waited longer than a given time limit are executed. Also, when a message is sent to another node, both the outgoing urgent message and non-urgent message queues are checked to see if there are any other messages going to the same destination. If there are, they are *piggybacked* on the first message. Because of the reduced overhead of sending fewer messages, bandwidth and time is saved. The prioritizing reduce the delay observed by the client, since urgent tasks and messages do not have to wait for the non-urgent, while the piggybacking improves the throughput. Both of these have more effect if the utilization is high (70% and more) [20].

3 Using the Semantics of Read and Write Operations

This work takes advantage of the different semantics of read and write operations. For the purpose of clarifying the semantic differences, transactions are classified into three groups in this paper:

Simple update. The transaction updates a single record.

Simple query. The transaction reads a single record.

General transaction. The transaction reads and/or writes two or more records.

The general transaction is efficiently handled using C1PC with Early Answer, Dynamic Coordinator, and prioritizing and piggybacking optimizations presented in Section 2. This leaves the two simple transaction types to be presented in this section.

3.1 Simple Update

A simple update transaction can be treated differently than the general transaction. In particular, it does not need to execute a distributed commit protocol. For disk-based databases, it is enough to set a write lock, update the record, force the log to disk, and return an answer to the client. Thus, a distributed commit protocol is not required. In contrast, main-memory databases persistently store the log by sending it to the backup. An answer can be returned to the client after both primary and backup know the outcome of the transaction.

When a simple update transaction arrives at a node in a main-memory system, the data tables are checked to see if the record being updated resides at the current node. If not, the transaction is forwarded to the proper node. This is the same as the dynamic coordinator optimization presented in Section 2 with the number of participants set to one.

At the proper node, the transaction must acquire a write lock on the record and perform the write. Then the log is sent to the backup. Piggybacked on the log are *Prepare* and *YouAreCoordinator*. Thus, it is a version of the *transfer of commit* optimization [14]. The backup becomes the coordinator and, since it knows that the primary is already prepared, it can decide to COMMIT or ABORT the transaction depending on its own vote.

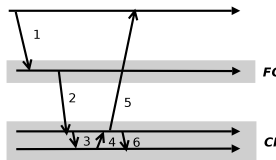


Fig. 2. The protocol for single-update

To guarantee that the system is *recoverable*, the COMMIT log record must be persistently stored, before a reply can be given to the client. ABORTs does not need this since a missing decision log record is presumed to be an ABORT [14]. The COMMIT is stored persistently by sending it to the primary. Since both the primary and backup now know the outcome, the primary can send an *Early Answer* to the client and an acknowledgement (Committed) to the backup. Then, the backup can forget about the transaction, i.e. remove it from the Active Transaction Table. After receiving an acknowledgement, the primary can also forget about the transaction.

Listings 1.1 and 1.2 give the algorithms for the primary and backup, respectively.

```

simple update: 1
if (data at another node) forward txn;
else {
  acquire write lock on record; 4
  write record;
  if (successful) {
    send log + prepare + root to backup; 7
    wait for outcome from backup;
    decide outcome;
    return early answer to client; 10
    release lock;
    send acknowledgement to backup;
  } else { 13
    decide abort;
    release lock;
    return error code to client; 16
  }
  forget transaction;
} 19
    
```

Listing 1.1. Primary coordinator

```

simple update: 20
save log;
if (ok) outcome = COMMIT;
else outcome = ABORT; 23
decide outcome;
send outcome to primary;
wait for acknowledgement; 26
forget transaction;
    
```

Listing 1.2. Backup coordinator

3.2 Simple Query

Simple query transactions require even less work than the simple update transactions. A read does not need to be logged and, since only a single record is read, no commit coordination is required. However, to avoid reading uncommitted data (a record written by an uncommitted transaction), a read lock must be acquired before the record is read.

```

simple read:
if (data at another node) forward txn;
else { 29
  acquire read lock;
  read record;
  release lock; 32
  return answer to client;
  forget transaction;
} 35
    
```

Listing 1.3. Primary coordinator

The algorithm is given in Listing 1.3. As for simple updates, it makes use of the dynamic coordinator optimization, forwarding the transaction to the proper node. It waits for the read lock to be acquired, reads the records, and releases the lock. Then, an answer can be sent to the client and the transaction can be forgotten.

4 Correctness

This section proves the correctness of the Simple Update (SU) protocol. Each of the properties given in Section 4.1 is proved in Section 4.2 in this order: NB-AC2, NB-AC3, NB-AC4, NB-AC1 and NB-AC5. The Simple Query (SR) protocol does not need a distributed commit protocol and is therefore trivial to prove.

4.1 The Non-blocking Atomic Commitment Problem

An atomic commitment protocol ensures that the participants in a transaction agree on the outcome, i.e. ABORT or COMMIT. Each participant votes, YES or NO, on whether it can guarantee the local ACID properties of the transaction. All participants have the right to *veto* the transaction, thus causing it to abort. The *Non-Blocking Atomic Commitment* problem, NB-AC, has these properties [21][22]:

NB-AC1 *<uniform agreement>*. All processes that decide reach the same decision.

NB-AC2 *<integrity>*. A process cannot reverse its decision after it has reached one.

NB-AC3 *<uniform validity>*. COMMIT can only be reached if *all* processes voted YES.

NB-AC4 *<non-triviality>*. If there are no failures and no processes voted NO, then the decision will be to COMMIT.

NB-AC5 *<termination>*. Every correct process eventually decides.

4.2 Proof

Lemma 1 NB-AC2: *A process cannot reverse its decision after it has reached one.*

Proof. The algorithm for the backup use an if-else statement to decide only once. The primary accepts the outcome decided by the backup.

Lemma 2 NB-AC3: *The COMMIT decision can only be reached if all processes voted YES.*

Proof. The backup decides COMMIT if the primary is successful (voted YES) and itself is successful. The primary can only decide commit if the backup has decided commit. Thus, all processes have voted YES for the COMMIT decision to be reached.

Lemma 3 NB-AC4: *If no process failed and no process voted NO, then the decision will be COMMIT.*

Proof. If no process failed, and no process voted NO, then, since the communication system is reliable, the backup receives YES from the primary and itself. Thus, COMMIT is reached (line 22).

Lemma 4 NB-AC1: *All processes that decide reach the same decision.*

Proof. The primary can decide ABORT if it is not successful in writing the record. The backup does not need to be informed of the decision, since it does not even know about the transaction. Aside from that, the primary can only decide ABORT if the backup decided ABORT first. Similarly, the primary can only decide COMMIT if the backup decided COMMIT first. As proven in Lemma 2, COMMIT can be decided (line 22) only if the primary voted YES. A vote YES from the primary implies that it was successful at updating the record and cannot decide ABORT. Hence, the primary and backup cannot reach diverging decisions.

Lemma 5 NB-AC5: *Every correct process eventually decides.*

Proof. To enable a process to decide in the presence of failures, all failure scenarios as well as the scenario with no failures must be handled. These scenarios can occur:

- 1: The primary fails before sending the vote to the backup or an error code to the client.
- 2: The primary fails after sending the reply to the client, but before sending an `ACKMSG` to the backup.
- 3: The primary fails after sending the `ACKMSG` to the backup.
- 4: The backup fails before sending the decision to the primary.
- 5: The backup fails after sending the decision to the primary.
- 6: No node fails.

Scenario (1): The backup does not know of the transaction and is consistent. The client can later resend the request.

Scenario (2): When the backup does not receive an acknowledgement from the primary, it can complete the transaction with the decided outcome. If the primary successfully sent a reply to the backup, the client may receive duplicate replies which must be filtered.

Scenario (3): This does not affect the processing of the current transaction.

Scenario (4): If the backup node has failed, the primary can safely abort the transaction.

Scenario (5): The transaction is completed by the primary.

Scenario (6): This is proven similarly to Lemma 3. Since no process failed and the communication system is reliable, either the primary decides `ABORT` or the backup receives a `YES` vote. If the backup receives a `YES`, it decides either `COMMIT` in line 22 or `ABORT` in line 23. If the primary decides `ABORT`, the backup will not decide. However, since an abort leaves the system in the same state as before the transaction arrived, the state of the backup will be identical to an aborted state.

All scenarios are handled, thus, all correct processes eventually decide.

Theorem 1. *SU is a valid non-blocking atomic commitment protocol.*

Proof. Since SU satisfies properties **NB-AC1** - **NB-AC5** it solves NB-AC.

5 The Simulation Model

To evaluate the performance of various main-memory commit protocols, a simulator of a realistic distributed transaction processing system has been developed. It has been implemented using Desmo-J, a framework for discrete-event modelling and simulation [23].

Measurements to provide the input values for the simulations are performed on an AMD Athlon(TM) 64 bits 3800+ processor, running a Linux 2.6 kernel. The length of the transaction operations are derived from measurements performed on a Java main-memory database prototype developed by Løland and Hvasshovd [24].

The distributed transaction processing TP system considered in this paper is composed of N nodes connected through a communication network, as shown in Fig. 3(a). Each node has a *transaction manager* (TM) which is responsible for orchestrating the correct *local* execution of transactions. The correct *global* execution of transactions is ensured by coordination between TMs.

Each transaction is associated with a *coordinator*, which is responsible for execution and termination. Typically, the coordinator is the *transaction manager* (TM) at the

node where the transaction was *initiated*. After the transaction execution is finished it is terminated using an atomic commitment protocol between the coordinator and participant TMs.

The distribution of processing and data is transparent to the transaction clients. The coordinator is responsible for issuing subtransactions to the appropriate nodes. Nodes receiving subtransactions are called *participants*. Generally, subtransactions can create further subtransactions, causing a multi-level transaction execution tree [25]. However, this paper discusses only trees that are one level deep. Transactions are assumed to be precompiled, and all nodes know where data are located. Thus, the coordinator can forward subtransactions directly to the correct participant.

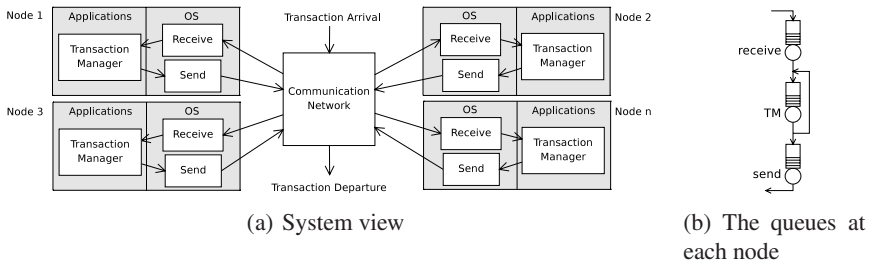


Fig. 3. Logical model of the system

We model each node to consist of a *transaction manager* (TM) executing on top of the operating system. The *operating system* (OS) is responsible for receiving and sending messages, while the transaction manager processes the requests. Other applications are assumed to be invoked from the TM. For each node, there are three First-In-First-Out queues: One for incoming messages, one for outgoing messages, and one for tasks to be processed by the TM. This is shown in Fig. 3(b). A processed task can result in a new local task, which is inserted at the end of the TM-queue, or a remote task, which is inserted into a message and sent by the operating system.

Process *context switches* between the two processes of the system, OS and TM, are modelled. The OS alternates between sending and receiving messages. Measurements show that these are performed in just $3.5 \mu\text{s}$. The *timeslice* is the maximum time given to each process to execute requests before another process is given time at the CPU. The default time slice for Linux kernel 2.6 systems is 100 ms and the minimum is 5 ms. To avoid delaying operations too long for this application, a time slice of 5 ms is chosen.

An *open* simulation model is used. The utilization of the system was varied from 1% - 97.5% to see the impact on the system performance. To be able to set the utilization for each protocol and optimization, a single transaction was run for each combination, and the total service demand of it was found. Then, since there are only a few context switches per transaction at high utilization, the costs of nearly all context switches was subtracted from the total. Then, the altered service demand was used as a divisor to find the maximum throughput for each of the protocols per node.

The transaction load is composed of simple reads, simple updates and standard transactions. The standard transaction is chosen to include three participants, and the

Table 1. The input parameters of the simulation

<i>Parameter</i>	<i>Value</i>	<i>Parameter</i>	<i>Value</i>
<i>Simulation Model</i>	Open	<i>Send Message</i>	$(26+0.055*B) \mu s$ if $B < 1500$,
<i>Context Switch</i>	$3.5 \mu s$		$(73+0.030*B) \mu s$ else
<i>Timeslice</i>	5 ms	<i>Receive Message</i>	$(52+0.110*B) \mu s$ if $B < 1500$,
<i>Simulation Time</i>	200 s		$(146+0.060*B) \mu s$ else
<i>Capture Time</i>	160 s [40 - 200]	<i>Long Operations</i>	700 μs DoWork,
<i>Simulation runs</i>	10		DoWorkAndPrepare
<i>Subtransactions</i>	Variable (1 or 3)	<i>Medium Operations</i>	150 μs BeginTxn, Prepare,
<i>Transactions</i>	Single-tuple update or read		Abort, Commit
<i># of simulated nodes</i>	20	<i>Short Operations</i>	50 μs WorkDone, Vote,
<i>CPU Utilization</i>	Variable		Aborted, Committed

operations can be both read and write. The ratio between the transactions are 20 simple reads and 20 simple writes for every standard transaction. The coordinator and participants are chosen randomly and uniquely for each transaction. Since none of the 3 records chosen is assumed to be at the same node, each of the three subtransactions of a transaction has a unique destination node.

In line with update transactions, the messages sent over the network are assumed to be small, i.e., 500 bytes for the first message to reach each participant for each transaction and 50 bytes for the others. Test experiments on a 100 Mbit/s Ethernet network show that no queuing effects for the network are likely to happen for the load of a distributed main-memory database. Also, small messages have a very short transmit time. Thus, the time to send a message from one node to another can be modelled as CPU execution time at the receiver and sender. The formulas for the response time are found by taking the average of 100.000 round-trip times for UDP datagrams with 10 bytes intervals. Measurements of CPU usages indicate that it takes twice as long to receive, as to send a message.

Operations are divided into three groups: Long, medium and short. These are made to reflect the various service demands needed by an *average* operation of that kind. Long operations (DoWork and DoWorkAndPrepare) take 700 μs , medium operations (BeginTxn, Prepare, Abort and Commit) take 150 μs and short operations (WorkDone, Vote, Aborted and Committed) takes 50 μs .

As the primary focus of this paper is the performance of commit protocols, the database internals are not modelled. A very large number of records is assumed, giving negligible data contention, hence, lock waiting effects have been ignored.

Each simulation lasted for 200 simulated seconds. The first 40 seconds are disregarded, as we are interested in the *steady-state* system. For each result presented, 10 simulation runs with random seed generators were made.

We assume the typical transaction load of 80% read-only transactions and 20% write-only transactions [26], and vice versa for comparison.

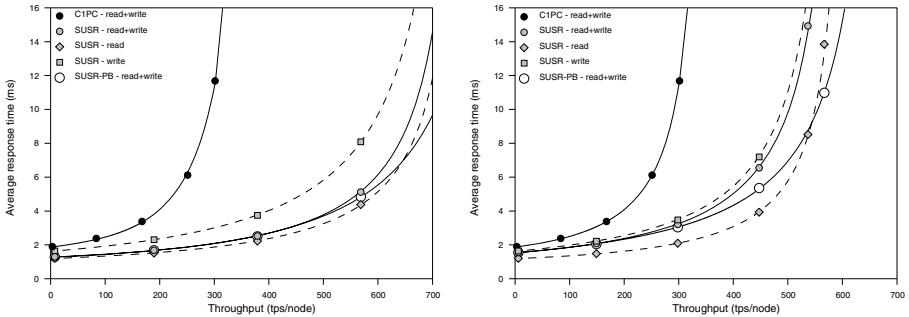
Table 1 summarizes the simulation parameters.

6 Simulation Experiments

The simulation results were gathered using the system model presented in Section 5. The algorithms presented in Section 3 (here referred to as Single Update - Single Read, SUSR) were compared with an efficient commit processing protocol (C1PC) and not using the dynamic coordinator optimization. In this section, the simulation results are presented. First, graphs of the average response time versus throughputs are presented, then the graphs for the upper bounds on the maximum response time for 95 % of the transactions [5] are explored.

6.1 Average Response Time

Simulations were performed for C1PC, SUSR, and SUSR with piggybacking (SUSR-PB) while varying the load on the system. The average response time versus throughput is shown in Figure 4. C1PC is shown in black, SUSR in grey, and SUSR-PB in white. For SUSR, the average response time for read, write, and combined read and write operations are shown as diamonds, squares, and circles, respectively.



(a) 80% simple reads and 20% simple updates (b) 20% simple reads and 80% simple updates

Fig. 4. The average response time

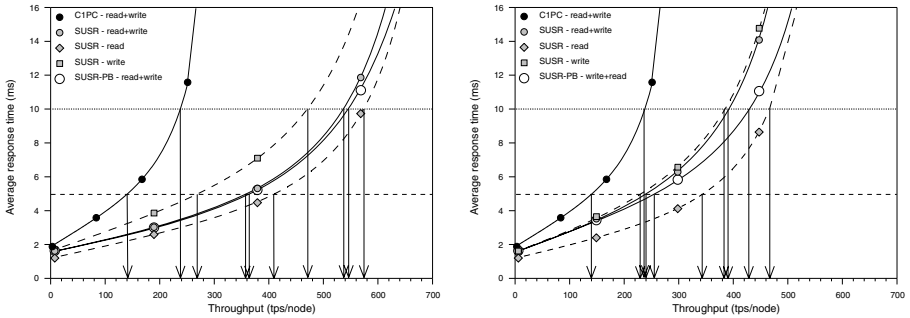
Figure 4(a) shows the results where 80% of the load consist of simple reads and 20% simple updates. C1PC has the longest average response time and the least throughput. For low utilization, SUSR and SUSR-PB have on average two-thirds of the response time of C1PC. Since SUSR and SUSR-PB support higher throughput than C1PC, a higher utilization makes the relative improvement higher. Up to approximately 500 transactions per second per node, SUSR and SUSR-PB has the same response time. For higher utilization, piggybacking and priority of operations reduce the response time. For SUSR, the read-only transaction part of the load has an average response time of 25% less than the write transactions at low utilization. At 80% utilization, the difference has risen to approximately 50%. Since the load mostly consists of read transactions, the average for both read and write transactions are closer to the read average than the write average. At very high throughput, the average response time for SUSR-PB gets shorter

than the same for the read transaction part of SUSR, because of the piggybacking and prioritizing optimizations.

Comparing the results in Figure 4(b) to the previously discussed results presented in Figure 4(a), the results for C1PC are, as expected, equal. This is because C1PC treats reads and writes in the same way. For SUSR and SUSR-PB the results are similar. The two major differences are that the maximum possible throughput is lower for the former, and the combined read and write transaction results for SUSR are closer to the average write response time. These are both caused by a larger part of the transactional load consisting of slower write transactions. For 80% update transactions, the piggybacking and priority improvements show a difference between SUSR and SUSR-PB already at 250 transactions per second per node. This is because there are more messages to be piggybacked for update transactions than for write transactions.

6.2 Response Time Demands

To examine the typical delay requirement for a certain percentage (i.e. 90%, 95%, 97%, 98%, or 99%) of transactions to finish within a given timelimit, Figure 5 plots the maximum response time for the 95% shortest transaction response times. The 5- and 10-milliseconds time limit is shown in dashed and dotted lines, respectively. The arrows points at the maximum throughput for each of the protocols at each time limit.



(a) 80% simple reads and 20% simple updates (b) 20% simple reads and 80% simple updates

Fig. 5. The maximum response time for the 95% quickest transactions

Figure 5(a) shows the results from executing the protocols with a load of 80% read transactions and 20% write transactions. For a time limit of 5 milliseconds, C1PC supports 140 tps/node and SUSR and SUSR-PB 360 tps/node. For a time limit of 10 milliseconds, C1PC tolerates 240 tps/node, while SUSR and SUSR-PB support 540 and 550 tps/node, respectively. For SUSR, data from the read-only and write-only transactions are plotted separately (grey diamonds and grey squares, respectively). If, for instance, the requirement for read-only transactions is that 95% complete in 5 milliseconds, while write-only can tolerate delays of up to 10 milliseconds, the maximum possible throughput is 410 tps/node (the minimum of the 5 millisecond limit for read-only and the 10 millisecond limit for write-only).

The results from executing the protocols with 20% read-only transactions and 80% write-only transactions are plotted in Figure 5(b). These show that C1PC tolerates 140 and 240 tps/node for the 5 and 10 milliseconds limits, respectively. SUSR and SUSR-PB support up to 240 and 260 tps/node, respectively, while keeping 95% of transaction response times below 5 milliseconds. If the limit is 10 milliseconds, the numbers are 390 and 430 tps/node for SUSR and SUSR-PB, respectively. As for the read-heavy load, the SUSR read-only and write-only transactions are plotted separately, and if 95% of the read-only transactions and write-only transactions must complete in 5 and 10 milliseconds, respectively, the maximum throughput is 340 tps/node.

Looking at SUSR versus C1PC for read-heavy load in Figure 5(a), the maximum throughput more than doubles while satisfying the same response time demands. For the write-heavy load in Figure 5(b), the maximum throughput is increased by 60 – 80% for both time limits.

7 Analysis

To verify the results from the simulations, the analytical model in [27] is used. The model consists of a multiple-class, open queuing network. The inter-arrival times are exponentially distributed. Single-server queuing theory can be used by assuming uniformity for all nodes [28]. The steady-state solutions are derived using results from $M/G/1$ queues [28].

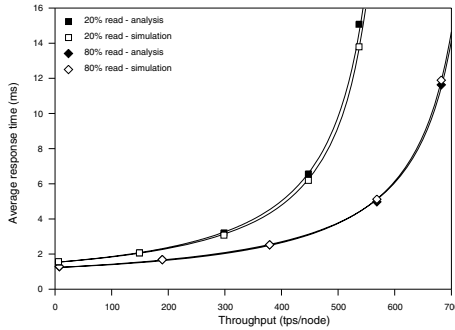


Fig. 6. Comparison between analyzes and simulations of SUSR

Figure 6 shows a comparison of simulation and analysis results for the SUSR protocol. The analyzes are shown in black and the simulations in white. The read-intensive results are plotted as diamonds, while the write-intensive are plotted as squares. The figure shows that the shape of the plots seem to verify the simulations. The largest difference for response time is less than 10%, which strongly indicates that the results are accurate.

8 Conclusion and Further Work

In this paper, approaches for efficient execution of simple read-only and write-only transactions are presented. The context of the paper is main-memory databases where

replication ensures durability and high availability. Both simulations and statistical analyzes were performed. The results show that, when applicable, the SUSR protocols significantly reduce the delay and increase the throughput. Compared to C1PC, SUSR tolerates 150% more transactions during read-heavy load while keeping the response time below 5 ms for 95% of the transactions. For write heavy load, the improvement is around 70%. The results indicate that the protocols can be favorably applied in systems which require high availability and short real-time responses and a large part of the load consists of transactions which only read or write a single record, such as telecommunications, sensor networks and online games.

Further work includes implementing SUSR in a real-world system and adopting the approach to multiple backups.

References

1. MobileTracker (2005), <http://www.mobiletracker.net/archives/2005/05/18/mobile-subscribers-worldwide>
2. Reuters (2007), <http://investing.reuters.co.uk/news/articleinvesting.aspx?type=media&storyID=nL29172095>
3. Ronström, M.: Database requirement analysis for a third generation mobile telecom system. In: Proceedings of the International Workshop on Databases in Telecommunications, pp. 90–105. Springer, London (2000)
4. Ronström, M.: Design and Modelling of a Parallel Data Server for Telecom Applications. PhD thesis, Uppsala University (1998)
5. Hvasshovd, S.O., Torbjørnsen, Ø., Bratsberg, S.E., Holager, P.: The ClustRa telecom database: High availability, high throughput, and real-time response. In: Proceedings of the 21th International Conference on Very Large Data Bases, pp. 469–477 (1995)
6. Garcia-Molina, H., Salem, K.: Main memory database systems: An overview. *IEEE Transactions on Knowledge and Data Engineering* 04, 509–516 (1992)
7. ISO: Information Technology - Database Language SQL - part 4: Persistent Stored Modules (SQL/PSM). ISO/IEC 9075-4 (2003)
8. Bernstein, P.A., Rothnie, J.B., Goodman, N., Papadimitriou, C.A.: The concurrency control mechanism of SDD-1: A system for distributed databases (the fully redundant case). *IEEE Transactions on Software Engineering* 4, 154–168 (1978)
9. Chan, A., Gray, R.: Implementing distributed read-only transactions. *IEEE Transactions on Software Engineering* 11, 205–212 (1985)
10. Lu, B., Zou, Q., Perrizo, W.: A dual copy method for transaction separation with multiversion control for read-only transactions. In: SAC 2001: Proceedings of the 2001 ACM symposium on Applied computing, pp. 290–294. ACM, New York (2001)
11. Garcia-Molina, H., Wiederhold, G.: Read-only transactions in a distributed database. *ACM Transactions on Database Systems* 7, 209–234 (1982)
12. Agrawal, D., Krishnaswamy, V.: Using multiversion data for non-interfering execution of write-only transactions. In: SIGMOD 1991: Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, pp. 98–107. ACM, New York (1991)
13. Heddaya, A.A.: Managing event-based replication for abstract data types in distributed systems. PhD thesis, Harvard University, Cambridge, MA, USA (1988)
14. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco (1993)

15. Ramamritham, K.: Real-time databases. *Distributed and Parallel Databases* 1, 199–226 (1993)
16. Niklander, T., Kiviniemi, J., Raatikainen, K.: A real-time database for future telecommunication services. In: Găiti, D. (ed.) *Intelligent Networks and Intelligence in Networks*. Chapman & Hall, Boca Raton (1997)
17. Lindström, J., Niklander, T., Porkka, P., Raatikainen, K.E.E.: A distributed real-time main-memory database for telecommunication. In: *Proceedings of the International Workshop on Databases in Telecommunications*, pp. 158–173. Springer, London (2000)
18. Kolltveit, H., Hvasshovd, S.O.: The Circular Two-Phase Commit Protocol. In: Kotagiri, R., Krishna, P.R., Mohania, M., Nantajeewarawat, E. (eds.) *DASFAA 2007*. LNCS, vol. 4443, pp. 249–261. Springer, Heidelberg (2007)
19. Kolltveit, H., Hvasshovd, S.O.: Efficient High Availability Commit Processing. *ARES* (2008)
20. Kolltveit, H., Hvasshovd, S.O.: Main memory commit processing: The impact of priorities. In: Haritsa, J.R., Kotagiri, R., Pudi, V. (eds.) *DASFAA 2008*. LNCS, vol. 4947, pp. 470–477. Springer, Heidelberg (2008)
21. Bernstein, P.A., Hadzilacos, V., Goodman, N.: *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publ. Co., Inc, Amsterdam (1987)
22. Guerraoui, R.: Revisiting the relationship between non-blocking atomic commitment and consensus. In: Helary, J.-M., Raynal, M. (eds.) *WDAG 1995*. LNCS, vol. 972, pp. 87–100. Springer, Heidelberg (1995)
23. Page, B., Kreutzer, W.: *The Java Simulation Handbook. Simulating Discrete Event Systems with UML and Java*. Shaker Verlag (2005)
24. Løland, J., Hvasshovd, S.-O.: Online, Non-blocking Relational Schema Changes. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) *EDBT 2006*. LNCS, vol. 3896, pp. 405–422. Springer, Heidelberg (2006)
25. Mohan, C., Lindsay, B., Obermarck, R.: Transaction management in the R* distributed database management system. *ACM Transactions on Database Systems* 11, 378–396 (1986)
26. Lindström, J., Niklander, T.: Benchmark for real-time database systems for telecommunications. In: *DBTel 2001: Proceedings of the VLDB 2001 International Workshop on Databases in Telecommunications II*, pp. 88–101. Springer, London (2001)
27. Kolltveit, H., Hvasshovd, S.O.: Performance of Main Memory Commit Protocols. Technical Report 06/2007, NTNU, IDI (2007)
28. Jain, R.: *The Art of Computer Systems Performance Analysis*. Wiley & sons, Chichester (1991)

Analysis and Interpretation of Visual Hierarchical Heavy Hitters of Binary Relations

Arturas Mazeika^{1,2}, Michael H. Böhlen², and Daniel Trivellato³

¹ Databases and Information Systems, Max Plank Institute for Informatics,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany

`amazeika@mpi-inf.mpg.de`

² Department of Computer Science, Free University of Bozen-Bolzano
Dominikanerplatz-3, I-39100, Bozen, Italy

`boehlen@inf.unibz.it`

³ Department of Mathematics and Computer Science, Technische Universiteit
Eindhoven, Den Dolech 2, 5612 AZ, Eindhoven, The Netherlands

`d.trivellato@tue.nl`

Abstract. The emerging field of visual analytics changes the way we model, gather, and analyze data. Current data analysis approaches suggest to gather as much data as possible and then focus on goal and process oriented data analysis techniques. Visual analytics changes this approach and the methodology to interpret the results becomes the key issue.

This paper contributes with a method to interpret visual hierarchical heavy hitters (VHHHs). We show how to analyze data on the general level and how to examine specific areas of the data. We identify five common patterns that build the interpretation alphabet of VHHHs. We demonstrate our method on three different real world datasets and show the effectiveness of our approach.

1 Introduction

Visual analytics changes the way we model, gather, and analyze data. In the past, the gathering and analysis of data was process and goal specific. Essentially, as much information as possible was collected and a number goal and process specific tasks were developed to analyze the data, including hypothesis testing, clustering, classification, and trend analysis. The role of the methods and technologies is passive: the data is processed, enriched with statistical information, and presented to the analyst. Only the analyst looks at the data and actively interprets the results. Visual analytics shifts the focus from how to model and process data to how and when to use data mining methods and how to interpret the results.

In this paper we develop a method to analyze and interpret data with the help of visual hierarchical heavy hitters (VHHHs) [15]. VHHHs allow to visually analyze two dimensional categorical data with hierarchical attributes. The method computes the hierarchical heavy hitters (HHHs) of a dataset and visualizes the HHH information in three dimensional space.

VHHHs support the analysis of binary relations captured by two-dimensional datasets. Binary relations abound and examples include datasets with “Source” and “Destination” attributes, such as Internet router traffic data, firewall data, airline databases (passengers travel from an arrival to a destination airport), etc. Other application areas are trend and pattern analysis to discover the relationships between car manufacturers and broken parts in cars, geographical information systems, etc.

The paper makes the following contributions:

- We formalize the VHHH method. The VHHH method uses *filtering*, *grouping*, and *count balancing* to compute two dimensional HHHs (2DHHHs). This yields a *lattice* structure that captures frequencies and hierarchical information. The VHHH method *orders* the HHHs and visualizes HHHs as rectangles in three dimensional space.
- We show how to analyze and interpret data with the VHHH method. The analysis starts with an overview of the HHHs and then zooms in and focuses on specific characteristics of the data. We identify the five most frequent patterns that form the interpretation alphabet of the VHHH method.
- We give an experimental evaluation of VHHHs on real world datasets. Our analysis confirms that the VHHH method is effective for a variety of tasks, including getting an overview of the domain, anomaly detection, and analysis of the distribution of the data.

The paper is organized as follows. Related work is reviewed in Section 2. In Section 3 we formalize the VHHH method. In Section 4 we develop techniques to interpret the data with the VHHH method. We draw conclusions and offer future work in Section 5.

2 Related Work

Research on hierarchical heavy hitters focused on the efficient computations in terms of time and space [8,17,19], and on incremental techniques suitable for streaming data [5,6]. Hershberger et al. [8] propose a method to visualize the lattice of HHHs, but they focus on the computation of HHHs information, and the data analysis potential of HHHs is not considered.

VHHHs method were introduced by Trivellato et al. [15]. The VHHH method computes two dimensional HHHs and visualizes the HHHs in three dimensional space. The potential of VHHHs as a visual data analysis tool is illustrated by real world datasets. In this paper we use the VHHHs method for visual analytics. We identify the five most frequent patterns and illustrate the patterns by real world datasets.

Various grouping techniques that present concise summaries of data have been proposed in the past. For example, data warehousing and OLAP techniques [12] group data according to a lattice into a high dimensional data cube, and use this cube for fast query answering. Simple visualization methods [11,16,13] offer the possibility to roll up and drill down selected groupings and to visualize the

groupings at different granularities. Similarly, association rules [1] identify large itemsets with the help of a lattice. Different visualizations have been proposed for association rules [3,18,10]. These works provide a general framework to interact and visualize data from OLAP tools.

There are many visualization techniques that have been proposed for categorical data [7,2,4,9]. The techniques focus on information illustration and a comprehensive presentation of all properties of the data, including bar-charts, pie-charts, parallel sets, histograms, tree-maps, pair-maps, mosaic plots, pixel bar charts, etc. The potential and the benefits of the tools are illustrated by selected examples. Often, these examples communicate deep knowledge of domain experts and valuable insights of experts of the respective tools. Further steps towards a comprehensive methodology for how to analyze data are not worked out as contributions.

3 The VHHs Method

In this section we present the Hierarchical Heavy Hitters method for mining large datasets. We show how two-dimensional hierarchical heavy hitters (2DHHs) are computed (Sections 3.1 and 3.2) and visualized with the VHHs tool (Section 3.3). We illustrate the concepts on a subset (cf. Figure 1(a)) of the `immigration` dataset (cf. Section 4).

3.1 The Idea of Two-Dimensional Hierarchical Heavy Hitters

2DHHs generalize equi-height histograms to categorical domains with hierarchies. An equi-height histogram for continuous numeric data allocates variable bins so the height of the bins approximately reaches the given threshold τ . The construction of equi-height histograms is typically done in the following way. First, the database is sorted and the process is initiated with the left most data item. This is the left boundary of the first bin. Then the bin is enlarged until the number of points in the bin reaches the given threshold τ .

2DHHs use a similar idea to build a summary structure: it allocates the smallest bin so that its count exceeds the given threshold τ . Due to the absence of natural order in the categorical domain, we cannot use the equi-histogram strategy to flexibly enlarge the bin until the threshold is reached. Instead, the construction of 2DHHs groups points into bins based on the hierarchical information of the dataset. This process is illustrated in Figure 1. The construction starts with the individual categories (cf. the node at the bottom of Figure 1). A category is a HHH if the count of the category exceeds the threshold (cf. HHHs (Europe/France, 20–30/25–29) and (Europe/Germany, 20–30/25–29)). These tuples are filtered out from further processing. Next, we group the points along the first hierarchy (cf. node *A* in Figure 1(b)) and along the second hierarchy (cf. node *B* in Figure 1(b)), and the filtering step is applied again. The process is continued until the top grouping is reached or no data is left. The grouping along two dimensions yields a lattice structure (cf. Figure 1(b)).

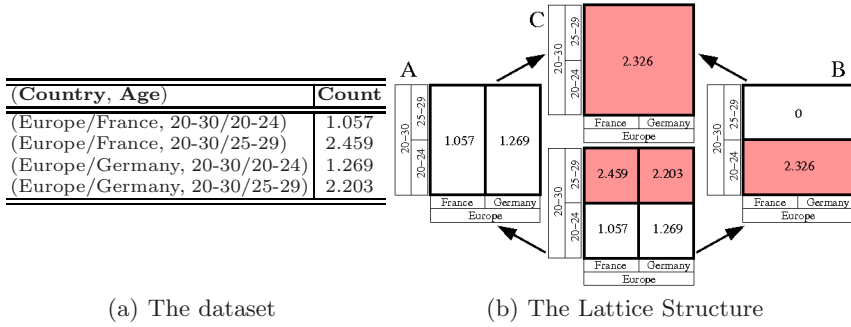


Fig. 1. The Idea of the 2DHHs Method

The computation of 2DHHs is based on three concepts: (i) *filtering*, (ii) *grouping*, and (iii) *count balancing*. The filtering outputs the groups that exceed threshold τ as HHHs and removes them from further processing. The grouping step accumulates the counts of non-HHHs along each dimension. The count balancing ensures that counts of HHHs are not counted twice. This can happen since grouping and filtering is done independently along each dimension and the combination of dimensions can lead to double counts. For example, (Europe, 20–30) in node *C* has a count of 2.326 (computed by grouping node *A*). However, this count was already used for the declaration of (Europe, 20–30/20–24) as a HHH in node *B*, and therefore the counts are not supposed to be propagated upwards. We discuss the concepts in detail in Section 3.2.

3.2 Foundations and Computation of 2DHHs

This section defines the filtering, grouping, and count balancing steps of the 2DHH method. Furthermore, we formalize the concept of the lattice structure obtained by the iterative grouping of the dataset. Let D be a three-dimensional dataset of size n with tuples $t = (t^x, t^y, t^c)$, where t^x and t^y are hierarchical attributes, and t^c is the count attribute. Hierarchical attributes are modeled with the help of trees $T^d = (V^d, E^d)$, where V^d are the nodes in the tree, E^d are the edges between the nodes, and $d \in \{x, y\}$.

Example 1 (Dataset and Hierarchies). Consider the following three dimensional hierarchical immigration dataset:

$$\{(Europe/France, 20-30/20-24, 1.057), (Europe/France, 20-30/25-29, 2.459), (Europe/Germany, 20-30/20-24, 1.269), (Europe/Germany, 20-30/25-29, 2.203)\}.$$

Then the hierarchies $T^x = (V^x, E^x)$ and $T^y = (V^y, E^y)$ are defined as follows

$$\begin{aligned} V^x &= \{Europe, Europe/France, Europe/Germany\} \\ E^x &= \{(Europe, Europe/France), (Europe, Europe/Germany)\}; \\ V^y &= \{20-30, 20-30/20-24, 20-30/25-29\} \\ E^y &= \{(20-30, 20-30/20-24), (20-30, 20-30/25-29)\} \end{aligned}$$

Filtering. The filtering step determines groups with counts that are above threshold τ as HHHs, and removes the corresponding counts from further processing.

Definition 1 (Filtering). Let D be a dataset of size n with hierarchies T^x and T^y . Let $A \subset T^x \times T^y \times \mathbb{R}^*$ be a dataset with categorical values from the hierarchies and positive counts. Let $\tau \in [0, 1]$ be a threshold parameter. Then the filtering step yields all tuples that are below the threshold:

$$F(A, \tau) = \{(t^x, t^y, t^c) \in A : t^c < \tau \cdot n\}$$

The HHHs are the tuples that exceed the threshold:

$$HHH(A, \tau) = \{(t^x, t^y, t^c) \in A : t^c \geq \tau \cdot n\}.$$

Example 2 (Filtering). We continue Example 1 with the immigration dataset and threshold $\tau = 0.2$ (the count for a HHH is $\tau \cdot n = 0.2 \cdot 6.988 \approx 1.398$). Then

$$HHH = HHH(\text{immigration}, 0.2) = \{(\text{Europe/France}, 20\text{--}30/25\text{--}29, 2.459), \\ (\text{Europe/Germany}, 20\text{--}30/25\text{--}29, 2.203)\}$$

is the set of HHHs. The tuples that remain for further processing are

$$D = F(\text{immigration}, 0.2) = \{(\text{Europe/France}, 20\text{--}30/20\text{--}24, 1.057), \\ (\text{Europe/Germany}, 20\text{--}30/20\text{--}24, 1.269)\}$$

Grouping. Grouping aggregates the counts of non-HHHs. Conceptually, we enlarge the bin to increase the counts of the group to reach threshold τ . On the technical level, the step groups tuples (Europe/France, 20–30/20–24, 1.057) and (Europe/Germany, 20–30/20–24, 1.269) into (Europe/France, 20–30, 1.057) and (Europe/Germany, 20–30, 1.269) (grouping along the Age hierarchy, cf. node A in Figure 1(b)), and (Europe, 20–30/20–24, 2.326) (grouping along the Country hierarchy, cf. node B in the Figure). Below we formalize the grouping step. Let t^d be a value in the d hierarchy. Then $children(t^d) = \{t'^d \in V^d : (t^d, t'^d) \in E^d\}$, where $d \in \{x, y\}$. For example, $children(\text{Europe}^{\text{Country}}) = \{\text{France}, \text{Germany}\}$.

Definition 2 (Grouping). Let D be a filtered dataset. Let $t_g^x \in V^x$ and $A = \{(t^x, t^y, t^c) \in D : t^x \in children(t_g^x)\}$. Then $t_g = (t_g^x, t^y, t_g^c)$ is the grouping of A according to hierarchy x iff $t_g^c = \sum_{\substack{(t^x, t^y, t^c) \in D \\ t^x \in children(t_g^x)}} t^c$. We say that A is grouped into t_g according to x and write $t_g = G^x(A)$.

Similarly, let $t_g^y \in V^y$ and set $B = \{(t^x, t^y, t^c) \in D : t^y \in children(t_g^y)\}$. Then $t_g = (t^x, t_g^y, t_g^c)$ is the grouping of B according to hierarchy y iff $t_g^c = \sum_{\substack{(t^x, t^y, t^c) \in D \\ t^y \in children(t_g^y)}} t^c$. We say that B is grouped into t_g according to y and write $t_g = G^y(B)$.

Set $G(D)$ is the grouping of dataset D (D is grouped into $G(D)$) iff it contains all tuples that are groupings according to hierarchies x and y . That is, let $t_g^x \in T^x$ and $t_g^y \in T^y$ such that the following sets are not empty

$$A_{t_g^x} = \{(t^x, t^y, t^c) \in D : t^x \in \text{children}(t_g^x)\}$$

$$A_{t_g^y} = \{(t^x, t^y, t^c) \in D : t^y \in \text{children}(t_g^y)\}$$

where $A_{t_g^x}$ and $A_{t_g^y}$ are the sets of children of t_g^x and t_g^y respectively. Then $G(D)$ is the union over all such sets:

$$G(D) = \bigcup_{t_g^d \in T^d, d \in \{x, y\} : A_{t_g^d} \neq \emptyset} G^d(A_{t_g^d}).$$

Example 3 (Grouping). We continue Example 2. The grouping is applied to the filtered dataset $F(\text{immigration}, 0.2)$. Then

$$G^x(\{(\text{Europe}/\text{Germany}, 20\text{-}30/20\text{-}24, 1.269), (\text{Europe}/\text{France}, 20\text{-}30/20\text{-}24, 1.057)\}) = (\text{Europe}, 20\text{-}30/20\text{-}24, 2.326),$$

$$G^y(\{(\text{Europe}/\text{France}, 20\text{-}30/20\text{-}24, 1.057)\}) = (\text{Europe}/\text{France}, 20\text{-}30, 1.057),$$

$$G(\text{immigration}) = \{(\text{Europe}/\text{France}, 20\text{-}30, 1.057), (\text{Europe}/\text{Germany}, 20\text{-}30, 1.269), (\text{Europe}, 20\text{-}30/20\text{-}24, 2.326)\}.$$

Grouping is done iteratively until no data is left to group or the root is reached along both hierarchies.

Count Balancing. Count balancing (CB) ensures that grouping achieves the same result independently of the order of the grouping.

CB starts with the independently aggregated and identified HHHs (cf. node A and B in Figure 1(b)). Next it subtracts the counts of the HHHs from the children level (cf. zeros in the child node in Figure 2). This way the children level obtains the information about the independently computed HHHs. Finally, it regroups the counts and proceeds with the grouping to the next level. This ensures that all counts that form a HHH at a given level are not propagated upwards in the lattice.

Below we formalize the CB step. We first define the set count difference and use the concept to define count balancing.

Definition 3 (Set Count Difference). Let A and B be datasets. The set count difference between sets A and B is set $A \setminus_c B$, containing all tuples from A with counts subtracted from the corresponding counts in B :

$$A \setminus_c B = \{(t^x, t^y, \max\{t_A^c - t_B^c, 0\}) : (t^x, t^y, t_A^c) \in A, (t^x, t^y, t_B^c) \in B\} \cup \{(t^x, t^y, t_A^c) : (t^x, t^y, t_A^c) \in A, (t^x, t^y, t_B^c) \notin B, t_B^c \in \mathbb{R}\}.$$

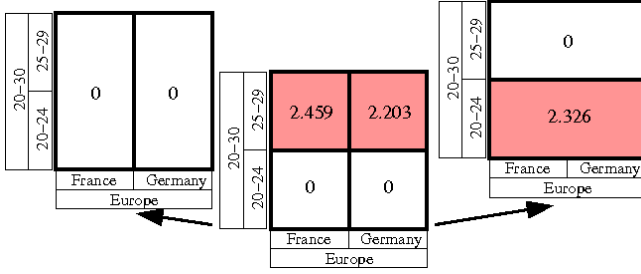


Fig. 2. The Counts After the Balancing Step. Dark Tuples are HHHs.

Example 4 (Set Count Difference). Let datasets A and B be defined in the following way:

$$A = \{(\text{Europe}/\text{France}, 20-30, 1.057), (\text{Europe}/\text{Germany}, 20-30, 1.269), \\ (\text{Europe}, 20-30/20-24, 2.326)\},$$

and

$$B = (\text{Europe}/\text{France}, 20-30, 1.000), (\text{Europe}/\text{Germany}, 20-30, 1.500)\}.$$

Then

$$A \setminus_c B = \{(\text{Europe}/\text{France}, 20-30, 57), (\text{Europe}/\text{Germany}, 20-30, 0), \\ (\text{Europe}, 20-30/20-24, 2.326)\}.$$

Definition 4 (Count Balanced Grouping). Let D be a dataset of size n with hierarchies T^x and T^y . Let $A \subset T^x \times T^y \times \mathbb{R}$. Let τ be the threshold parameter. $CBG(A)$ is a count balanced grouping of A iff

$$CBG(A) = G(A) \setminus_c G(A_{HHH_G}),$$

where A_{HHH_G} are the tuples that contributed for the HHHs and $HHH_G = HHH(G(A), \tau)$:

$$A_{HHH_G} = \left\{ (t^x, t^y, t^c) \in A : \text{either } (\exists t_g^x : (t_g^x, t^y, t_g^c) \in HHH_G \text{ and } t^x \in \text{children}(t_g^x)) \right. \\ \left. \text{or } (\exists t_g^y : (t^x, t_g^y, t_g^c) \in HHH_G \text{ and } t^y \in \text{children}(t_g^y)) \right\}.$$

Example 5 (Balanced Grouping). We continue Example 3. Then

$$HHH_G = \{(\text{Europe}, 20-30/20-24, 2.326)\}.$$

The children that contribute to HHH_G are the following:

$$\text{immigration}_{HHH_G} = \{(\text{Europe}/\text{France}, 20-30/20-24, 1.057), \\ (\text{Europe}/\text{Germany}, 20-30/20-24, 1.269)\}.$$

The grouping of the children is

$$G(\text{immigration}_{HHG}) = \{(\text{Europe}/\text{France}, 20-30, 1.057), \\ (\text{Europe}/\text{Germany}, 20-30, 1.269), (\text{Europe}, 20-30/20-24, 2.326)\}.$$

Therefore the count balanced grouping is

$$CBG(\text{immigration}) = \{(\text{Europe}/\text{France}, 20-30, 0), \\ (\text{Europe}/\text{Germany}, 20-30, 0), (\text{Europe}, 20-30/20-24, 0)\}.$$

The Lattice Structure. The iterative grouping of the filtered and balanced data along the two dimensions yields a lattice structure. The lattice consists of all possible combinations of attribute values together with the counts, and is organized into nodes. Strictly, each tuple represents an independent node. In practice, we group the individual nodes into nodes defined by grouping. Let $CBG^d(D) = G^d(D) \setminus_c G^d(D_{HHG})$ be the grouping of a dataset D according to hierarchy $d \in \{x, y\}$. Then the groupings $CBG^x(D)$ and $CBG^y(D)$ represent two distinct nodes. Two nodes are connected by an edge if there is a grouping relationship between the two.

Definition 5 (Lattice). Let $T^x = (V^x, E^x)$ and $T^y = (V^y, E^y)$ be the hierarchies of the attributes. $L = (V, E)$ is the lattice generated by grouping iff the following is true. Nodes V of the lattice are the Cartesian product of the nodes of the hierarchies and the count domain: $V \subseteq V^x \times V^y \times \mathbb{R}$. Edges E in the lattice are defined by the relations between tuple t and its grouping $G(t)$: $(t_1, t_2) \in E \iff t_2 = CBG(t_1), t_1, t_2 \in V$.

Example 6 (Lattice). Let's continue Example 3. $L = (V, E)$ is the lattice generated by the iterative process of the computation of the HHHs with $V = \{t_1, t_2, \dots, t_9\}$, where

$$\begin{aligned} t_1 &= (\text{Europe}/\text{France}, 20-30/20-24, 0), & t_6 &= (\text{Europe}/\text{Germany}, 20-30, 0), \\ t_2 &= (\text{Europe}/\text{France}, 20-30/25-29, 2.459), & t_7 &= (\text{Europe}, 20-30/20-24, 2.326), \\ t_3 &= (\text{Europe}/\text{Germany}, 20-30/20-24, 0), & t_8 &= (\text{Europe}, 20-30/25-29, 0), \\ t_4 &= (\text{Europe}/\text{Germany}, 20-30/25-29, 2.203), & t_9 &= (\text{Europe}, 20-30, 0) \\ t_5 &= (\text{Europe}/\text{France}, 20-30, 0), \end{aligned}$$

and $E = \{(t_1, t_5), (t_1, t_7), (t_2, t_5), (t_2, t_8), (t_3, t_6), (t_3, t_7), (t_4, t_6), (t_4, t_8), (t_5, t_9), (t_6, t_9), (t_7, t_9), (t_8, t_9)\}$. The lattice is illustrated in Figure 3.

In order to determine HHHs we need to scan all nodes in the lattice and filter out all nodes that are above the given threshold.

Definition 6 (Hierarchical Heavy Hitters). Let $L = (V, E)$ be the lattice obtained by filtering, balancing, and grouping dataset D with threshold τ . Then the set of HHHs of D is the set

$$HHH(D, \tau) = HHH(V, \tau).$$

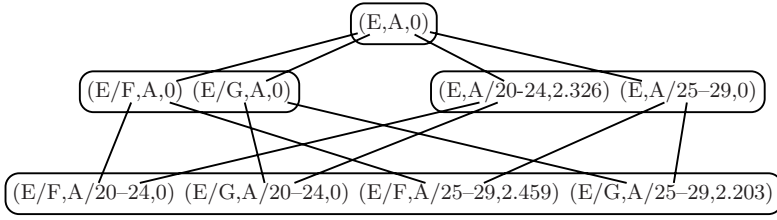


Fig. 3. The Lattice of the `immigration` Dataset (cf. Figure 1(b)). Europe is abbreviated as E, France as F, Germany as G, 20–30 as A.

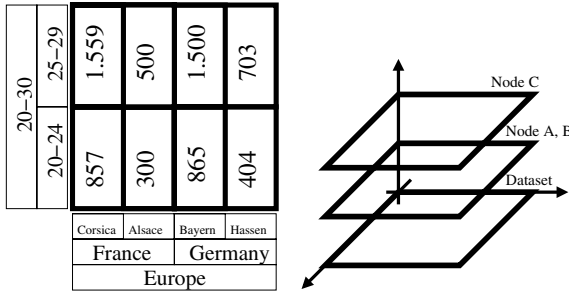
Example 7 (Hierarchical Heavy Hitters). Let’s continue Example 6. The set of HHHs for dataset `immigration` and threshold $\tau = 0.2$ is the following:

$$HHH(\text{immigration}) = HHH(V, 0.2) = \{(\text{Europe/France}, 20\text{--}30/25\text{--}29, 2.459), (\text{Europe/Germany}, 20\text{--}30/25\text{--}29, 2.203), (\text{Europe}, 20\text{--}30/20\text{--}24, 2.326)\}.$$

3.3 Visualization of 2DHHHs

Two issues must be solved in order to visualize 2DHHHs: (i) ordering of categorical data and (ii) representation of the lattice on the Cartesian coordinates system. We discuss these issues in turn.

Ordering of Categorical Data. All visualizations that use a Cartesian coordinates system require an ordering of the data. For numerical data this is trivial, since numerical values enjoy an ordering. For some categorical data a natural ordering exists (for example Saturday precedes Sunday, June precedes July), but in the general case an ordering needs to be established for categorical data for the mapping to the axes.



(a) Ordering of Categorical Data (b) Visualization Planes

Fig. 4. VHHHs: Planes and HHH Nodes

We order categorical data based on the count information of the tuples. We compute the counts of the tuples and position the values of highest counts towards the beginning of the axes. This is illustrated in Figure 4(a). First, we identify the tuple with the highest count (Europe/France/Corsica, 20–30/25–29, 1.559). This value is placed left most according to the Country and the top part according to the Age coordinate (since a natural ordering exists for the Age coordinate). The placement of Europe/France/Corsica value at the beginning of the axes restricts the ordering of the other values: the values from France will be allocated in the first part of the axes, while the values from Germany will be allocated in the second part of the axes. Then we look for the second largest count in the database; Europe/France/Alsace is placed second after Corsica value. Then we continue with the second largest count (Europe/Germany/Bayern, 20–30/25–29, 1500). This places Bayern value first after the France group and completes the ordering of the data.

Representation of the Lattice with VHHHs. The nodes of the lattice are mapped into planes in the Cartesian system based on the height of the node (or the level of grouping of the data in the node) from the leaf (cf. Figure 4(b)). This way the leaf node is mapped to level 0, nodes *A* and *B* are mapped to level 1, node *C* is mapped to level 2. The HHHs on each visualization plane are visualized as red rectangles.

4 Interpretation of VHHHs

In this section we develop a method how to analyze data with the help of VHHHs. First, we give an overview of the VHHHs method and discuss the choice of the threshold parameter τ in Sections 4.2 and 4.3. Then we zoom in, and identify typical patterns of the visualizations along with interpretations in Section 4.4.

The method is illustrated on three real world datasets: Internet traffic log (`traffic`, more than 2,000 records), Olympic games gold medals (`medals`, more than 800 tuples), and Italian immigration data (`immigration`, more than 1,300,000 records) (cf. Figure 5). The `traffic` dataset records information about the URLs that are accessed by the programs of a personal computer (cf. Figure 5(a)). The `medals` dataset records the information about the gold medals won in terms of (country, sport discipline) (cf. Figure 5(b)). The `immigration` dataset records the country and the age of the immigrants to Italy (cf. Figure 5(c)).

4.1 Getting Started with the VHHHs Method

We implemented the VHHH method as a module of the XVDM system [14]. The `traffic`, `medals`, and `immigration` datasets are distributed in the comma separated values (CSV) format together with the XVDM system and available for the interested reader to test and experiment.

The VHHH module allows to select the following input parameters: the two-dimensional hierarchical dataset, threshold value, layer of the visualization to

(URL,Program)	Count	(Type of Sport,Country)	Count
(www.unibz.it, C:/WINDOWS/system32/ntoskrnl.exe)	96	(World/Asia/Eastern/China, Olympic/Summer/Outdoor/Shooting)	7
(hp.msn.com, C/Programmi/Intercom/iexplore.exe)	50	(World/Australia/Australia, Olympic/Summer/Outdoor/Archery)	1
(owa.unibz.it, C/Programmi/Msoffice/OUTLOOK.exe)	74	(World/Europe/Eastern/Romania, Olympic/Summer/Gymnastics/AGymnastics)	7
...

(a) traffic Dataset

(Country,Age)	Count
(World/Europe/EU/Austria, Age/<20/<5)	64
(World/Europe/EU/Germany, Age/20-40/20-24)	1269
(World/Europe/EU/Latvia, Age/>60/>75)	7
...	...

(b) medals Dataset

(c) immigration Dataset

Fig. 5. Real World Datasets Used in the Analysis

display (show all HHHs or just a single level), switch on/off labels, and transparency level of the HHHs. Changing the threshold value allows the analyst to focus on different granularities of the data (cf. Section 4.1). The other parameters only impact the visualization of the HHHs and are used to fine tune the visualization (switch off the labels to see the general picture or switch on the labels to identify the individual HHHs). For clarity of the printed version of the figures we do not show the labels of HHHs.

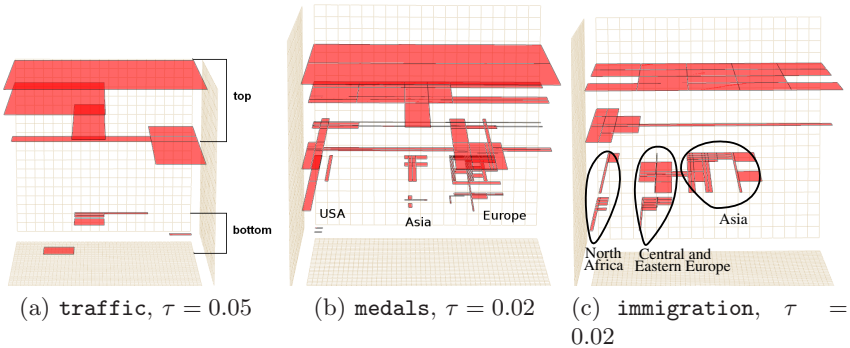


Fig. 6. Getting Started with the Data

Figure 6 shows screen-shots of the VHHH module of the system. The HHHs are shown in red, and organized in layers (cf. Section 3.3). The red rectangles at the bottom indicate the HHHs that result from the dataset immediately, or with a few iterations of grouping. The heavy hitters at the top layers indicate that a lot of grouping had to be performed to reach the threshold.

4.2 General Strategy to Analyze Data with the VHHHs Method

The analysis of a dataset starts with an overview visualization. The overview allows to investigate the general distribution of the data: the analyst should

look for accumulations of HHHs, empty regions of space, as well as transitions of HHHs from one region to the other. Typically this is done by visualizing all layers simultaneously, looking at the data from different angles, and analyzing the change of HHHs as the threshold parameter varies (cf. Section 4.3).

Once the general knowledge and feeling about the data is established, the analyst focuses on more specific areas of the visualization combined with specific values of the threshold parameter. Typical patterns along with interpretations of the patterns (cf. Section 4.4) are identified in the visualizations. In this stage, individual layers of the lattice and individual HHHs within the level planes are being investigated. This allows to identify specific HHHs as well as ranges of HHHs.

The VHHH method combined with domain knowledge is a powerful visual data analysis tool. For example, the gap between the top and bottom layers in the `traffic` data (cf. Figure 6(a)) clearly separates potential spyware from the rest of the programs; the accumulations of HHHs in Figure 6(c) denote the countries that most contribute to the immigration to Italy, including North Africa (with one leading country), Central/Eastern Europe (one leading country), and Asia (a number of leading countries); there is also a smooth transition of number of immigrants between Europe and Asia; the accumulations in the `medals` dataset identify the main players in sports: USA, Europe, and Asia in their respective fields. Furthermore, one can see that Europe has many HHHs both spread horizontally and vertically, indicating the specialization of individual countries in certain areas of sports (horizontal spread), as well as groups of countries excelling in a specific discipline (vertical spread).

4.3 Choosing the Threshold Parameter

The choice of the threshold parameter τ depends on the purpose of the analysis, the distribution of the data, and the available domain knowledge.

If the purpose of the analysis is well specified and domain knowledge is available (for example, identify all programs that are responsible of 95% of the traffic, or find all regions that handle at least 10% of the overall phone calls) then the choice of the threshold parameter is straightforward.

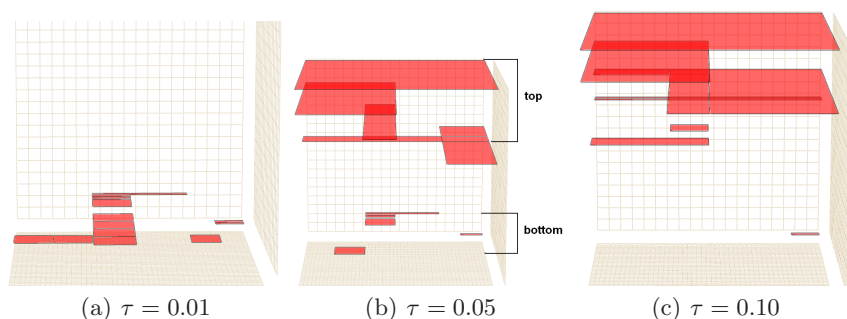


Fig. 7. The `traffic` Dataset

If the purpose of the analysis is specific but no domain knowledge is given (for example, identify possible groups of spyware, retrieve the list of regions that receive most of the phone calls) then typically the threshold parameter is first identified by trial and error, and then the selected threshold value is used to analyze the data.

If the purpose of the analysis is vaguely defined or left unspecified then no best threshold parameter exists. In this case, the range of the threshold parameter is identified with the lowest value resulting in all HHHs to be at the bottom of the visualization (cf. Figure 7(a)), and the highest value resulting in all HHHs to be at the top (cf. Figure 7(c)). Then animated visualizations for threshold values covering the range are shown to the analyst.

4.4 Visualization Patterns of the VHHHs Method

In this section we identify five typical visualizations patterns (flag, chess board, cross, triangular, and sandwich) of the VHHH method and provide interpretations for each of them. We conceptualize and illustrate the patterns by real world examples.

Flag Pattern. The flag pattern is defined by HHH stripes followed by non-HHH stripes (cf. Figures 8(a)–8(b)). This pattern shows a periodicity for one attribute, and a steady behavior for the other one. Flag patterns are probably the most common patterns, and an example is very clearly seen in the `immigration` dataset (cf. Figure 8(b)). Here, the three stripes correspond to the three main continents of immigrants to Italy (Europe, Africa, and Asia). Within each continent there is one or two leading countries that dominate the other ones.

Chess Board Pattern. In the chess board pattern one or several HHH nodes is followed by one or several non-HHH nodes. The conceptual idea is shown in Figures 8(c)–8(d). Such very regular patterns rarely occur in real world data but the alternation of HHHs with non-HHHs can be identified easily (cf. Figure 8(d)).

The chess board pattern in Figure 8(d) shows that in general Europe is strong in winter sports with individual countries being strong in individual sport disciplines.

Cross Pattern. The cross pattern consists of two HHHs crossing each other on the same layer (cf. Figure 9(a)–9(b)). This indicates a very scattered dataset consisting of a wide variety of tuples with very small values. Grouping must be applied many times (often up to the root level along one of the dimensions) until the accumulated counts reach the threshold.

The cross pattern is prominently seen in the `medals` dataset (cf. Figure 9(b)). Here, most of the data is reported as HHHs at the lower levels of the visualization and a lot of scattered individual counts should be accumulated up to the top in order to form a HHH.

Triangular Pattern. In the triangular pattern layers of HHHs form a triangle or multiple triangles (cf. Figures 9(c) and 9(d)). The triangle distribution of

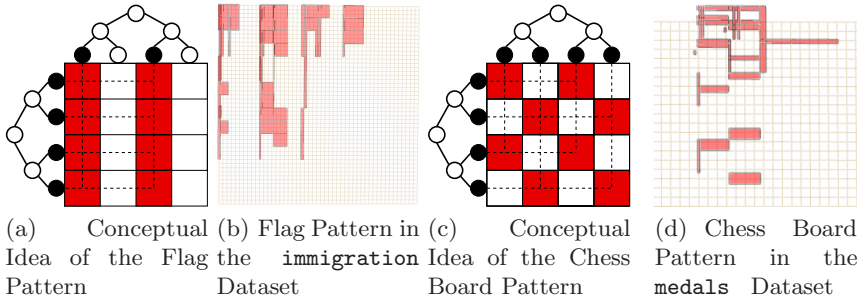


Fig. 8. The Flag and the Chess Board Patterns

layers of HHHs indicates a skew (zipfian distribution) in the data. Typically, in such datasets there is a dominant tuple (lowest level of grouping), a dominant group (1st level of grouping), a dominant group of groups (2nd level of grouping), etc. Since the VHHH method orders the tuples so that the highest counts are allocated towards the origin of the space, the result is a triangle pattern.

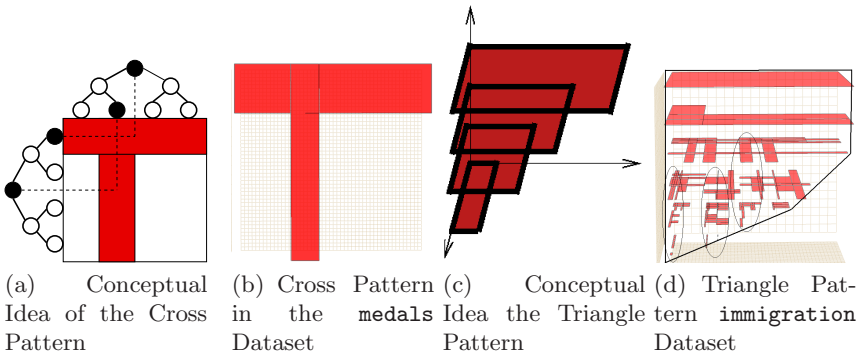


Fig. 9. The Cross and the Triangle Pattern

A triangle pattern is clearly visible in the immigration dataset (cf. Figure 9(d)). The dataset is very skewed: there are two countries (one from Africa and one from Eastern Europe) with a very high immigration level, followed by a group of countries with large counts from Asia and finishing with a small set of countries with small counts from America.

Sandwich Pattern. The presence of the most detailed and the most aggregated layers and the absence of middle layers of HHHs forms the sandwich pattern (cf. Figures 10(a) and 10(b)). Similar to the triangle pattern, this denotes the skew in the dataset: there are a number of large HHHs at the tuple level, and the rest of the counts are scattered among a large number of different tuples.

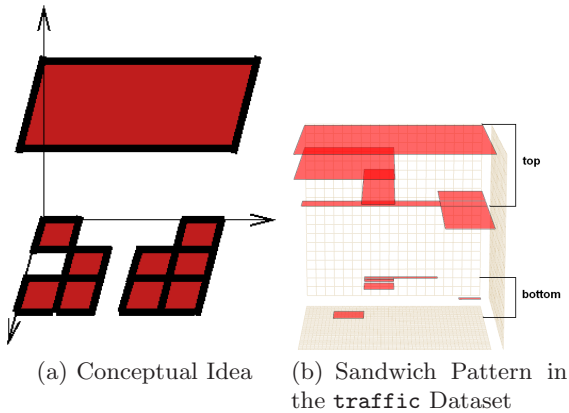


Fig. 10. The Sandwich Pattern

Example of the sandwich pattern is clearly seen in the `traffic` dataset (cf. Figure 10(b)). Here the top layers are clearly separated from the bottom layers of HHHs distinctively, isolating possible spyware from the rest of the programs.

5 Conclusions

This paper formalizes and shows how to analyze the data and interpret the results with the help of the VHHH method. First, 2DHHS are computed with the help of filtering, grouping, and count balancing. This creates a lattice representation of HHHs. Second, an ordering for the categorical data is introduced, the lattice is mapped into a three-dimensional space, and HHHs are shown as rectangles in the three-dimensional space. The analysis of VHHH starts with an overview of the general distribution of the data, and continues with detail analysis of selected areas of the visualization. We identify five frequent patterns of VHHHs. The patterns build the interpretation alphabet of the VHHH method and help to understand the data.

References

1. Adamo, J.-M.: Data mining for association rules and sequential patterns: sequential and parallel algorithms. Springer, Heidelberg (2001)
2. Bendix, F., Kosara, R., Hauser, H.: Parallel sets: Visual analysis of categorical data. In: INFOVIS 2005, p. 18. IEEE Computer Society, Los Alamitos (2005)
3. Chakravarthy, S., Zhang, H.: Visualization of association rules over relational dbms. In: SAC 2003: Proceedings of the 2003 ACM symposium on Applied computing, pp. 922–926 (2003)
4. Chintalapani, G., Plaisant, C., Shneiderman, B.: Extending the utility of treemaps with flexible hierarchy. In: IV2004, pp. 335–344 (2004)
5. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Finding hierarchical heavy hitters in data streams. In: VLDB, pp. 464–475 (2003)

6. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In: SIGMOD 2004, pp. 155–166 (2004)
7. Friendly, M.: Visualizing categorical data: Data, stories, and pictures. In: SAS Users Group International 25th Annual Conference (2000)
8. Hershberger, J., Shrivastava, N., Suri, S., Tòth, C.D.: Space complexity of hierarchical heavy hitters in multi-dimensional data streams. In: PODS2005, pp. 338–347 (2005)
9. Keim, D.A., Hao, M.C., Dayal, U.: Hierarchical pixel bar charts. *IEEE Transactions on Visualization and Computer Graphics* 8(3), 255–269 (2002)
10. Liu, Y., Salvendy, G.: Design and Evaluation of Visualization Support to Facilitate Association Rules Modeling. *International Journal of Human-Computer Interaction* 21(1), 15–38 (2006)
11. Mansmann, S., Scholl, M.H.: Exploring olap aggregates with hierarchical visualization techniques. In: SAC 2007, pp. 1067–1073 (2007)
12. Marakas, G.M.: *Modern Data Warehousing, Mining, and Visualization: Core Concepts*. Pearson Education, London (2002)
13. Stolte, C., Tang, D., Hanrahan, P.: Query, analysis, and visualization of hierarchically structured data using polaris. In: SIGKDD, pp. 112–122 (2002)
14. XVDM System (2007) [Online; accessed 02-October-2007], <http://www.inf.unibz.it/dis/wiki/doku.php?id=xvdm:xvdm/>
15. Trivellato, D., Mazeika, A., Böhlen, M.H.: Using 2D hierarchical heavy hitters to investigate binary relationships. In: Simoff, S.J., Böhlen, M.H., Mazeika, A. (eds.) *Visual Data Mining*. LNCS(LNAI), vol. 4404, pp. 215–236. Springer, Heidelberg (2008)
16. Vinnik, S., Mansmann, F.: From analysis to interactive exploration: Building visual hierarchies from olap cubes. In: EDBT, pp. 496–514 (2006)
17. Wang, J., Miller, D.J., Kesidis, G.: Efficient mining of the multidimensional traffic cluster hierarchy for digesting, visualization, and anomaly identification. Technical Report NAS-TR-0023-2005, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park (August 2005)
18. Wong, P.C., Whitney, P., Thomas, J.: Visualizing association rules for text mining. In: INFOVIS 1999, p. 120 (1999)
19. Zhang, Y., Singh, S., Sen, S., Duffield, N., Lund, C.: Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In: IMC 2004, pp. 101–114 (2004)

QUESTO: A Query Language for Uncertain and Exact Spatio-temporal Objects

Hoda Mokhtar¹ and Jianwen Su²

¹ Information Systems Dept.
Faculty of Computers and Information
Cairo University

hmokhtar@gmail.com

² Department of Computer Science
University of California at Santa Barbara
su@cs.ucsb.edu

Abstract. Querying moving objects is a crucial ingredient in many applications involving moving objects. Being able to query the location of an object is important to achieve services like E-911, m-commerce, fleet management, etc. However, moving objects have inherent uncertainty in their location information. Obtaining the exact location of an object at every time instant is infeasible. In this paper we aim towards designing an algebraic query language that can query and reason about trajectory properties of moving objects both with precise and uncertain trajectories.

1 Introduction

The current rapid advances in wireless, mobile, positioning, and sensor technologies, together with the continuous miniaturization of computing devices are pushing the migration from desktop to hand-held computing devices as PDAs (Personal Digital Assistant), and powerful cell phones. Today, location based services (LBS) are invading the market with a wide range of applications including: automatic vehicle location (AVL), fleet management, transportation and traffic management, air traffic control, location-aware content delivery (e.g. m-commerce, marketing, and nearest interest location), and digital battlefield.

The mobility characteristics of moving objects have led to several challenges. Querying moving objects is one of the problems that was studied in several earlier works. Precise trajectories were the core in most of those efforts including [10,14,19,6,13,18,21,8,5]. Modeling and querying uncertain trajectories was also presented in some earlier work including [16,21,15,20,17]. However, designing query languages for uncertain moving objects trajectories was not investigated. In this paper we outline an algebra for querying both precise and uncertain trajectories. Our algebra is different from the algebra developed in [10]. In [10] the authors propose abstract data types for moving objects. They present an SQL based query language for precise trajectories. Although one can express trajectory properties such as “on-border”, “touches”, etc., their language essentially depends on using the abstract data types that encapsulate many important representation

and evaluation issues. Temporal properties are expressed through lifting spatial properties to include a temporal flavor.

Another relevant earlier work is the one proposed in [9], in this paper the authors propose a constraint query algebra for linear constraint databases. Although the work is not related to moving objects, the algebra provides a fundamental step in manipulating linear constraints using operators extended from the relational algebra. Specifically, their algebra, similar to Codd’s algebra for finite relations [3], has operators that apply to finite representations of possibly infinite sets. Our algebra presents an extension from the algebra in [9], this extension comes from our data model for trajectories which separates time and space.

The rest of the paper is organized as follows: Section 2 discusses the syntax of the proposed algebra for both precise and uncertain trajectories. Section 3 presents the semantics of the algebra and evaluation techniques for precise trajectories. This section also discusses the complexity of query evaluation over precise trajectories. Section 4 presents the semantics of the algebra and evaluation techniques for uncertain trajectories. In addition complexity of algebra operators is also presented. Section 5 summarizes the paper.

2 Syntax of Algebra

We start our technical presentation by introducing the data types and schema that we will use along with the syntax of the algebra expressions and formulas. Moving object queries are mainly composed of three components namely, *regions*, *time*, and *trajectories*. In this paper we focus on static regions to ease the technical presentation.

2.1 Data Types and Schema

In this section we present some necessary concepts that will be used throughout the paper. First, we define moving objects as spatio-temporal objects whose location and/or extent change over time. In our discussion we only consider moving points. Moving points are suitable to model planes, cars, buses, trains, people, etc. whose locations and movements are important. Moving points are therefore points whose location changes over time but have no extent. Our data model uses “linear constraints” that are logical formulas to represent spatial and spatio-temporal information, in the spirit of [11].

As presented in [13], an *atomic linear constraint* over variables x_1, \dots, x_n is an expression of the following form:

$$c_1x_1 + c_2x_2 + \dots + c_nx_n \theta c_0,$$

where c_0, c_1, \dots, c_n are real numbers in \mathbb{R} and θ is a predicate in $\{=, <, \leq, >, \geq\}$. Constraints are interpreted over the real numbers in the natural manner. A *linear constraint* over variables x_1, \dots, x_n is a Boolean combination of atomic linear constraints over variables x_1, \dots, x_n in disjunctive normal form (DNF).

(Linear) constraints are characterized by their ability to *finitely* represent *infinite* sets of points (i.e., regions). Therefore, an *n-dimensional region* where $n > 0$ is a natural

number, can be represented as a boolean combinations of atomic linear constraint in disjunctive normal form over n variables as defined in [13].

For the rest of this discussion we fix n to denote the dimensionality, and T to be a totally ordered infinite set of *time instants* that are isomorphic to the real numbers. A *time interval* is thus a subset of T that is either closed or open.

In this paper, a *moving object trajectory* is represented as a sequence $(z_0, m_0, z_1, m_1, \dots, z_k, m_k)$. Such that, z_i is a time instant and m_i is a linear function of the form $at + b$, where $a, b \in \mathbb{R}$ and $t \in T$, and for each $0 \leq i < k$, $z_i < z_{i+1}$, and $m_i(z_{i+1}) = m_{i+1}(z_{i+1})$ as defined in [13].

Using the linear constraint model, we propose the following data types:

1. *Scalar type*, let Dom represent the set of scalar values.
2. *Region type*, the domain of region values is a set of linear constraints as discussed above.
3. *Trajectory type*, a trajectory value is also viewed using linear constraints as discussed above.
4. *Time type*, the domain of time values is a set of finite sets of disjoint time intervals.

In our technical model, we allow regions to be disconnected and have holes. In addition, we also allow trajectories to be discontinuous.

In the rest of our discussion we assume that each trajectory is mapped to linear constraints as follows: each motion of the trajectory is mapped to an atomic linear constraint, and the sequence of motions are then disjunctively connected to express a trajectory. Similarly, each time value is mapped to an atomic constraint as follows: each interval $[a, b]$ is mapped to the constraint $a \leq t \leq b$, and each interval (a, b) is mapped to $a < t < b$, finally interval $[a, a]$ are mapped to $t = a$.

2.2 Time and Region Expressions

We assume the existence of infinite and disjoint sets: \mathbf{P} of *relation names* and \mathbf{A} of *attribute names*. A *relation schema* consists of a relation name, and a finite set of attribute names (distinct) and type pairs. A *database schema* is a finite set of relation schemas with distinct names.

A *tuple over* a relation schema R is a mapping ω such that for each attribute name A in R of type T , $\omega(A)$ is a value in the domain of type T . We also denote $\omega(A)$ as $\omega.A$.

Definition 1. A relation (instance) of a relation schema R is a finite set of tuples of R . A database (instance) of a database schema S is a mapping d such that for every relation schema R in S , $d(R)$ is a relation instance of R .

An important aspect of the algebra is to be able to reason about regions and time values. This is achieved through the following operators. A *region operator* has as argument(s) values of type region and returns a value of type region. The region operators are union (\cup) and difference ($-$).

A *time operator* has as argument(s) values of type time and returns a value of type time. The time operators include union (\cup) and difference ($-$).

Let U_r, U_t denote the universe of region and time types, resp. Using the operators listed above, we can easily express intersection of regions r, s as $r - (r - s)$ and complement of r as $U_t - r$ if r is of type time or $U_r - r$ if r is of type region. Using those operators we can then define our language expressions as follows.

Definition 2. *The sets of time and region expressions are defined recursively below.*

1. A region expression is one of the following:
 - (a) A region constant (i.e. a generalized relation),
 - (b) An attribute of type region,
 - (c) An expression using region operators: $e_1 \cup e_2, e_1 - e_2$ where e_1, e_2 are region expressions, and
 - (d) A region expression of the form: $A.Loc()$ or $A.Loc(e)$, where e is a time expression and A an attribute of type trajectory.
2. A time expression is one of the following:
 - (a) A time value,
 - (b) An attribute of type time,
 - (c) An expression using time operators: $e_1 \cup e_2, e_1 - e_2$ where e_1, e_2 are time expressions.
 - (d) A time expression of the form: $A.Time()$ or $A.Time(e)$, where e is a region expression and A an attribute of type trajectory.

A region expression is an expression that returns a region type. Region constant is basically a generalized region relation that is a Boolean combination of linear constraints expressing the region. Combining regions with region operators also returns linear constraints that define the resulting region.

The expressions $A.Loc()$ and $A.Loc(e)$ are new expressions that are introduced to enable expressing spatial properties. $A.Loc()$ and $A.Loc(e)$ are intended to enable spatial projection so that a query expression can access the spatial part of a trajectory.

Example 1. Let r_1, r_2 be 2 regions and τ a trajectory. The following are valid region expressions:

$$r_1, r_1 \cup r_2, \tau.Loc(), \tau.Loc([3, 4]) - r_1$$

Similarly, time expressions allow reasoning about either explicit time values or to access the temporal portion of a trajectory. The expressions $A.time()$, $A.Time(e)$ are introduced to enable the latter. The following are valid time expressions:

$$[2, 2], [2, +\infty), [3, 4] \cup [3, 5], \tau.Time(), \tau.Time(r_1)$$

In the following section we present the semantics of those expressions and show how they are used in query evaluation.

2.3 Algebra Operators

Expressions are used to build formulas. A formula is atomic if it has no connectives. Atomic formulas can be later used to construct selection formulas.

Definition 3. *An atomic formula is defined as:*

1. $empty(e)$, where e is either a region or time expression,
2. $A = val$, where A is a scalar attribute,
3. $A_1 = A_2$, where A_1, A_2 are attributes of the same type,
4. $e\theta e'$, where e, e' are either both region expressions or both are time expressions, and $\theta \in \{\subseteq, =\}$, and
5. $e < e'$, where e, e' are both time expressions.

A selection formula is thus defined as:

1. $\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$, where ϕ_1, ϕ_2 are atomic or selection formulas,
2. $\neg\phi$ where ϕ is an atomic or selection formula.

In the above formulas we can also express $e > e'$, where e, e' are both time expressions as $e' < e$. Having defined our expressions and formulas, we now focus on expressing queries. In general, queries in our algebra are regular algebraic queries like those in relational databases. The main difference is the structure of our selection and projection expressions. Using the above definitions for the syntax of the algebra we can define the conditions for the algebra operators.

Selection Operator. A *selection condition* is either an atomic formula or a selection formula.

Example 2. Let *Buses* be a relation with attributes: A, B where A is a scalar and B a trajectory attribute. And let *Cities* be a relation with attributes C, D where C is a scalar and D a region attribute. The following are valid selection conditions:

- $A = BUS12$.
This selection condition selects buses with scalar attribute A equal to the scalar value $BUS12$.
- $(C = SB) \wedge (\neg empty(B.Time(D) \cap [7, 9]))$.
This selection condition selects a region with scalar attribute equals SB and finds the buses that visited SB region during the time interval $[7, 9]$.

Projection Operator. Let A_1, A_2, \dots, A_n be attributes. A projection expression over A_1, A_2, \dots, A_n is defined as:

1. \emptyset , or
2. c_1, c_2, \dots, c_k where each c_i is either an attribute in A_1, A_2, \dots, A_n or a time or region expression involving only attributes in A_1, A_2, \dots, A_n , where $k \geq 1$.

Finally, the family of *algebraic expressions* \mathcal{L} is defined inductively as follows: (assuming that e_1 and e_2 are two algebraic expressions):

1. R is an atomic expressions, where R is a relation name,
2. (Cartesian product) If e_1 and e_2 have disjoint sets of attribute names, then $(e_1 \times e_2)$ is also an expression,
3. (Selection) If F is a selection formula involving only attribute names in e_1 , then $\sigma_F e_1$ is an expression,
4. (Projection) If E is a projection expression, then $\pi_E e_1$ is an expression,

5. (Set operations) If e_1, e_2 have exactly the same set of attributes, then $e_1 - e_2, e_1 \cap e_2, e_1 \cup e_2$ are algebraic expressions, and
6. (Rename) If A, B are two attribute names such that A occurs in e_1 but B does not, then $\rho_{A \rightarrow B} e_1$ is an expression.

Example 3. In this example we illustrate the algebra operators. We assume the same relation schema of Example 2. A database with 2 relations *Buses* and *Cities*. Relation *Buses* has 2 attributes: A of type scalar and B of type trajectory. *Cities* also has with 2 attributes: C of type scalar and D of type region. The following are some example queries expressed using the algebra operators presented in this section:

Q1: Which buses were in the SB area before t_0 ?

$Q_1 : \pi_A \sigma_{\alpha \wedge \beta} Buses \times Cities,$

where $\alpha = "C = SB"$ and $\beta = "B.Time(D) < (t_0, \infty)"$.

The selection condition selects a region with scalar attribute equals SB and finds the buses that visited SB region before t_0 , the projection expression then returns the scalar attribute A in the *Buses* relation.

Q2: Was BUS12 inside SB area any time between t_0 and t_1 ?

$Q_2 : \pi_{\emptyset} \sigma_{\alpha \wedge \beta \wedge \neg empty(\gamma)} Buses \times Cities,$

where $\alpha = "A = BUS12"$, $\beta = "C = SB"$, and $\gamma = "B.Loc([t_0, t_1]) \cap D"$.

The selection condition selects a region with scalar attribute equals SB, a bus with scalar attribute equals BUS12, and checks that BUS12 was inside SB any time during the time interval $[t_0, t_1]$.

Q3: Where was BUS12 from t_0 to t_1 ?

$Q_3 : \pi_{B.Loc([t_0, t_1])} \sigma_{A=BUS12} Buses.$

The selection condition selects a bus with scalar attribute equals BUS12, then the projection expression returns the location of BUS12 during the time interval $[t_0, t_1]$.

3 A Semantics of the Algebra for Trajectories

In this section we present the semantics of the query algebra proposed in Section 2. We start with defining the semantics of the result of a time or region expression e on a tuple ω denoted by $e \llbracket \omega \rrbracket$.

1. If $e = v$ where v is a region value, then $e \llbracket \omega \rrbracket = v$,
2. If $e = l$ where l is a time value, then $e \llbracket \omega \rrbracket = l$
3. If $e = A$ where A is an attribute name of type region or time, then $e \llbracket \omega \rrbracket = \omega.A$
4. If $e = A.Loc()$, then $e \llbracket \omega \rrbracket = \psi(x_1, \dots, x_n)$ where $\psi(x_1, \dots, x_n)$ is a linear constraint over the variables x_1, \dots, x_n such that $\psi(x_1, \dots, x_n)$ is logically equivalent to $\exists t(\omega.A)$ and t is of type time.
5. If $e = A.Loc(e_t)$, then $e \llbracket \omega \rrbracket = \psi(x_1, \dots, x_n)$, where e_t is a time expression, $\psi(x_1, \dots, x_n)$ is a linear constraint over the variables x_1, \dots, x_n , such that ψ is logically equivalent to $\exists t(\omega.A \wedge e_t \llbracket \omega \rrbracket)$, and t is of type time.
6. If $e = A.Time()$, then $e \llbracket \omega \rrbracket = \psi(t)$, where $\psi(t)$ is a linear constraint over the time variable t such that $\psi(t)$ is logically equivalent to $\exists x_1 \exists x_2 \dots \exists x_n(\omega.A)$.

7. If $e = A.Time(e_r)$, then $e\llbracket\omega\rrbracket = \psi(t)$, where e_r is a region expression, and $\psi(t)$ is a linear constraint over the time variable t such that $\psi(t)$ is logically equivalent to $\exists x_1 \exists x_2 \dots \exists x_n (\omega.A \wedge e_r\llbracket\omega\rrbracket)$.
8. If $e = e_1 \cup e_2$ where e_1, e_2 are both time or region expressions. Then, $e\llbracket\omega\rrbracket = e_1\llbracket\omega\rrbracket \vee e_2\llbracket\omega\rrbracket$ where e_1, e_2 are linear constraints.
9. If $e = e_1 - e_2$ where e_1, e_2 are both time or region expressions. Then, $e\llbracket\omega\rrbracket = e_1\llbracket\omega\rrbracket \wedge \neg e_2\llbracket\omega\rrbracket$ where e_1, e_2 are linear constraints.

The above semantics is straightforward except for the new region operators: $A.Loc()$, $A.Loc(e_t)$, and the new time operators $A.Time()$, $A.Time(e_r)$. The semantics of those operators are based on quantifier elimination techniques similar to the techniques introduced in constraint databases [12]. More specifically, region operators $A.Loc()$, $A.Loc(e_t)$ require elimination of the time variable t so that the result is linear constraints over the spatial variables x_1, \dots, x_n only. Similarly, the time operators $A.Time()$, $A.Time(e_r)$ require elimination of the spatial variables x_1, \dots, x_n so that the result is linear constraints over the time variable t only.

Definition 4. Let ϕ, ψ be two linear constraints (regions) over variables x_1, \dots, x_n . Then, ϕ, ψ are semantically equivalent, denoted by $\phi = \psi$, if $\forall x_1 \dots \forall x_n \phi(x_1, \dots, x_n) \leftrightarrow \psi(x_1, \dots, x_n)$ holds.

Let d be a database. We define the semantics of a tuple ω satisfying a selection formula F expressed as $\omega \models F$ as follows:

1. $\omega \models \text{empty}(e)$ where e is a time expression, if $\exists t e\llbracket\omega\rrbracket(t)$ is not true.
2. $\omega \models \text{empty}(e)$ where e is a region expression, if $\exists x_1 \exists x_2 \dots \exists x_n e\llbracket\omega\rrbracket(x_1, \dots, x_n)$ is not true.
3. $\omega \models A = val$ if $\omega.A = val$ where A, val are both scalar.
4. $\omega \models A_1 = A_2$ where A_1, A_2 are both scalar, region, time or trajectory attributes, if $\omega.A_1 = \omega.A_2$.
5. $\omega \models e \subseteq e'$ where both e, e' are region expressions, if $\forall x_1 \dots \forall x_n (e\llbracket\omega\rrbracket(x_1, \dots, x_n) \rightarrow e'\llbracket\omega\rrbracket(x_1, \dots, x_n))$ is true.
6. $\omega \models e \subseteq e'$ where both e, e' are time expressions, if $\forall t (e\llbracket\omega\rrbracket(t) \rightarrow e'\llbracket\omega\rrbracket(t))$ is true.
7. $\omega \models e < e'$, if $\forall t \forall t' (e\llbracket\omega\rrbracket(t) \wedge e'\llbracket\omega\rrbracket(t') \rightarrow t < t')$ is true.
8. $\omega \models \phi_1 \wedge \phi_2$ where ϕ_1, ϕ_2 are both formulas, if $\omega \models \phi_1$ and $\omega \models \phi_2$.
9. $\omega \models \phi_1 \vee \phi_2$ where ϕ_1, ϕ_2 are both formulas, if $\omega \models \phi_1$ or $\omega \models \phi_2$.
10. If $\omega \models \neg\phi$ where ϕ is a formula, if $\omega \not\models \phi$.

Let R be a relation schema (name) with attributes A_1, \dots, A_n , and ω a tuple over R . Then the *result* of a projection condition P on ω expressed as $P\llbracket\omega\rrbracket$ is defined as:

1. If $P = \emptyset$ then

$$P\llbracket\omega\rrbracket = \begin{cases} \{\{\}\} & \text{if } \omega \text{ is satisfiable} \\ \{\} & \text{otherwise} \end{cases}$$

2. If $P = c_1, \dots, c_k$ then $P\llbracket\omega\rrbracket = \{(c_1\llbracket\omega\rrbracket, \dots, c_k\llbracket\omega\rrbracket)\}$.

The above discussion presented the semantics of applying the algebra operators on a tuple ω . Following that we present the result of applying the operators on a relation R .

Definition 5. Let ω_1, ω_2 be two tuples over attributes A_1, \dots, A_k . Then, ω_1, ω_2 are semantically equivalent, denoted by $\omega_1 = \omega_2$, if $\forall 1 \leq i \leq k, \omega_1.A_i = \omega_2.A_i$.

Let d be a database with a database schema S , and Q an algebra query expression. The answer Q on database d , denoted by $Q(d)$, is defined as:

1. If $Q = R$, then $Q(d) = d(R)$.
2. If $Q = Q_1 \times Q_2$, then $Q(d) = Q_1(d) \times Q_2(d) = \{(\omega_1, \omega_2) \mid \omega_1 \in Q_1(d), \omega_2 \in Q_2(d)\}$.
3. If $Q = Q_1 - Q_2$, then $Q(d) = Q_1(d) - Q_2(d) = \{\omega \mid \omega \in Q_1(d), \omega \notin Q_2(d)\}$.
4. If $Q = Q_1 \cup Q_2$, then $Q(d) = Q_1(d) \cup Q_2(d) = \{\omega \mid \omega \in Q_1(d) \text{ or } \omega \in Q_2(d)\}$.
5. If $Q = Q_1 \cap Q_2$, then $Q(d) = Q_1(d) \cap Q_2(d) = \{\omega \mid \omega \in Q_1(d), \omega \in Q_2(d)\}$.
6. If $Q = \sigma_F Q'$, then $Q(d) = \{\omega \mid \omega \in Q'(d) \wedge \omega \models F\}$, where F is a selection formula.
7. If $Q = \pi_P Q'$, then $Q(d) = \{P \parallel \omega \parallel \mid \omega \in Q'(d)\}$ where P is a projection condition.
8. If $Q = \rho_{A \rightarrow B} Q'$, where A is an attribute name in the result of Q' and B is a new attribute name, then $Q(d)$ has the same relation as $Q'(d)$ except that attribute name A is replaced by B .

Recall that \mathcal{L} denotes the algebraic query language.

Proposition 1. *The query language \mathcal{L} is closed under selection, projection, cross product, and set operations.*

Proposition [1](#) follows from the fact that under all algebraic operators the result of a query expression on a relation is still a relation.

4 Handling Uncertain Trajectories

In this section we make initial steps towards extending the language \mathcal{L} to querying uncertain trajectories. More specifically, we study a restricted class of \mathcal{L} and show how this restricted class can express probabilistic queries over moving object trajectories with uncertainty. Designing query languages for trajectories with uncertainty is relatively a new research area. Earlier language design efforts were mostly targeted towards querying precise trajectories. In [\[20\]](#) the authors introduce uncertain trajectories, and present predicates for querying those uncertain trajectories. The main differences between the approach that the authors presented in [\[20\]](#) and the one we present in this section are: (1) their trajectory model treats time differently not as part of the uncertainty modeling (2) their queries have an implicit probabilistic behavior that should be implied from the predicate, such as definitely, sometimes, and always, and (3) no precise language definition was presented in [\[20\]](#), while in our approach probability is part of the query language. In the following discussion we explore querying uncertain trajectories using \mathcal{L} .

4.1 Algebraic Queries over Uncertain Trajectories

In Section [2](#) we presented the algebraic query language \mathcal{L} . We claim that the algebra is powerful enough to be used to express both precise and uncertain trajectories. More

specifically, in this section we show the semantics of a restricted class of the algebra for querying uncertain trajectories. To start our technical presentation, we first present our data types. Roughly speaking, we replace the domain of the trajectory type introduced in Section 3 by trajectories with uncertainty. A trajectory (with uncertainty) is a sequence of motions (with uncertainty) each of which is a product of stochastic processes. In our technical presentation we limit motions to follow a uniform stochastic process (i.e. uniform probability distribution function). As discussed in [15] using uniform distribution is a reasonable assumption that is commonly used in prior work. In this section we continue to consider precise regions and precise time.

Following the approach in [15], let u, d be two real numbers where $d \geq 0$. A uniform distribution over the interval $[u - d, u + d]$ is represented by the pair (u, d) . Where u is the mean of the distribution.

Thus, a *uniform stochastic (random) process* is a pair (μ, δ) where μ, δ are (continuous, piecewise) linear functions with the time parameter t such that for every real number a , $(\mu(a), \delta(a))$ is a uniform distribution.

Therefore, as defined in [15], an uncertain trajectory *u-trajectory* is basically a sequence of uncertain motions *u-motion*, where an *n-dimensional motion with uncertainty* (or *u-motion*) is an *n*-vector (f_1, \dots, f_n) , where for each $1 \leq i \leq n$, f_i is a uniform stochastic process or a constrained stochastic process. A *trajectory with uncertainty* (or *u-trajectory*) is a sequence $(a_0, m_0, a_1, m_1, \dots, a_k, m_k)$, where k is a nonnegative integer, for each $0 \leq i \leq k$, a_i is a real number or the special symbol $-\infty$, which is smaller than all the real numbers, and m_i is a u-motion such that $a_0 < a_1 < \dots < a_k$, and for each $1 \leq i \leq k$, m_{i-1} and m_i meet at time a_i .

In our presentation we restrict \mathcal{L} so that it only allows each expression to range over at most one trajectory attribute, besides, we also limit the expressions allowed over uncertain trajectories. Our restricted algebra however inherits most of the syntax and semantics of \mathcal{L} presented earlier in Sections 2 and 3.

In this section we show how some of the expressions in \mathcal{L} can be modified to accommodate uncertainty reasoning. Specifically, we focus on the region expressions $\tau.Loc()$ and $\tau.Loc(e)$, where τ is a u-trajectory, and show how those expressions are modified to express probabilistic queries. For time expressions we only consider time values. For algebra operators we restrict to selection with probability and projection. Clearly, we consider selection with probability since selection conditions on trajectory attributes can no longer be viewed as regular selections as in relational algebra (i.e. based on truth values). Due to the fact that the key for evaluating n-dimensional trajectory queries over uncertain trajectories depends on the evaluation of the 1-dimensional trajectory queries.

In the remainder of the paper we focus on the evaluation of the 1-dimensional case. Let τ_x denote the projection of the u-trajectory τ on the *x*-dimension.

In this section we start our technical presentation with discussing our algorithms for evaluating the different types of region expressions that are required to query trajectories with uncertainty. We view a database d as defined in Section 2.

Example 4. Consider the single u-motion u-trajectory τ . Let τ_x denote the projection of τ on the *x*-dimension such that $\tau_x = (2, (\mu = t + 3, \delta = 2), 9)$ as shown in Figure 1. Note that the u-motion is constrained by the time interval $[2, 9]$.

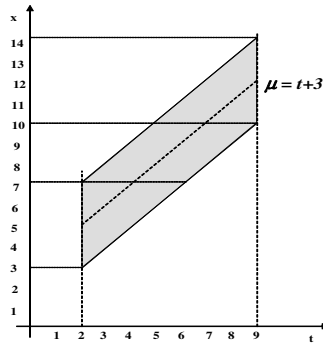


Fig. 1. $\tau_x.Loc()$ Evaluation Example

Assume we want to evaluate the query “Which buses were probably in SB area between 2:00pm and 7:00pm?”. Then, the question is what should be the answer for the query, and how to evaluate that answer?

Example 4 presents the questions that we need to consider for evaluating the queries on uncertain trajectories. In the following discussion we will go through our approach to solve those problems.

Example 5. Consider again the u-trajectory in Example 4. In this example we want to illustrate how we can evaluate a region expression $\tau_x.Loc()$. Briefly, we first need to find the projection of the u-trajectory on the x -dimension, and then compute the probability of τ_x being in different regions of this projection.

The result of $\tau_x.Loc()$ is thus the possible set of intervals where τ_x can be located during the time interval $[2, 9]$. Since τ_x is a u-trajectory so the answer for the expression should actually reflect the uncertainty aspect of the trajectory by returning the probability of being in each interval. So the expected result would be $\{(I_1, P_1), (I_2, P_2) \dots (I_k, P_k)\}$ where each I_i is a spatial interval and P_i is a probability value. Now, the key problem of evaluating $\tau_x.Loc()$ is to identify those interval-probability pairs.

From Figure 1, it is clear that τ_x is only defined in the x -interval $[3, 14]$. Thus, the probability of the trajectory being outside this interval is simply 0. Also, it is clear that any location of τ_x intersecting the interval $[7, 10]$ has a probability 1, as τ_x will certainly cross this interval. The tricky part is if τ_x is located between $[3, 7]$ or $[10, 14]$, then the probability computation is no longer trivial. Since both cases are symmetric, we will consider the case of τ_x being in the interval $[x, y]$ such that $[x, y] \subseteq [3, 7]$. The probability in this case is basically the probability of τ_x existing in the range $[x, y]$ during the time interval $[2, 9]$. This turns to be an existential range query (ER query) as defined in [15]. Employing the evaluation technique for ER queries, the probability $P([x, y] \mid [x, y] \subseteq [3, 7])$ can be efficiently computed.

From Example 4 we can see that evaluating $\tau_x.Loc()$ is a 2 fold problem:

1. identifying the different possible intervals for the u-trajectory, and
2. computing the probability associated with each interval.

In the rest of our discussion we will show how these problems are solved and how different region expressions can be consequently evaluated.

Definition 6. Assume that τ is a 1-dimensional u -trajectory with 1 u -motion $\tau = a_0, m, a_1$. Let $\alpha(a)$ denote the value of α at time instant a . The characteristic points of τ is a quadruple (c_1, c_2, c_3, c_4) representing the 4 end points of the uncertainty volume of τ defined as:

$$\begin{aligned} c_1 &= \mu(a_0) - \delta(a_0) \\ c_2 &= \mu(a_0) + \delta(a_0) \\ c_3 &= \mu(a_1) - \delta(a_1) \\ c_4 &= \mu(a_1) + \delta(a_1) \end{aligned}$$

The main role of the characteristic points is to help identify the end points of the different possible location intervals of a u -trajectory. This will later help to fully specify all the possible locations, and finally compute their probabilities.

Definition 7. A basis B is a finite set of pairs $\{(b_1, P_1), \dots, (b_k, P_k)\}$ such that (1) $k \geq 1$, (2) for each $1 \leq i \leq k$, b_i is a (spatial) interval, (3) $\bigcup_{1 \leq i \leq k} b_i = \mathbb{R}$, (4) for all $1 \leq i, j \leq k$, $i \neq j$ implies $b_i \cap b_j = \emptyset$, (5) for each $1 \leq i \leq k$, $P_i : \{[x, y] \mid [x, y] \subseteq b_i\} \rightarrow [0, 1]$ is a mapping from a set of intervals $[x, y] \subseteq b_i$ to real numbers between $[0, 1]$. We denote a finite set of basis by a basis set.

We define the *answer of a region expression over a u -trajectory* as a set of pairs $\{(id, B)\}$, where id is a scalar attribute, B is a basis.

The following is an example to illustrate the function of the characteristic points and the basis.

Example 6. Consider again the u -trajectory of Example 4 where $\tau_x = (2, (\mu = t + 3, \delta = 2), 9)$ as shown in Figure 1. Again, the goal is to evaluate the region expression $\tau_x.Loc()$.

Using the definitions of characteristic points and basis, we can now obtain a more precise representation for our evaluation paradigm as follows:

The characteristic points of τ_x are $(3, 7, 10, 14)$. These points are simply the upper and lower boundary points of the u -motion at times 2 and 9 respectively.

Employing the characteristic points, we can now define our basis set $B = \{B_1, B_2, B_3, B_4, B_5\}$ where $B_1 = ((-\infty, 3], 0)$, $B_2 = ([3, 7], P([x, y]))$, if $[x, y] \subseteq [3, 7]$, $B_3 = ([7, 10], 1)$, $B_4 = ([10, 14], P([x, y]))$, if $[x, y] \subseteq [10, 14]$, and $B_5 = ([14, \infty), 0)$, where $[x, y]$ is a possible location of τ_x .

Thus the answer to the expression $\tau_x.Loc()$ will simply return the pair (id, B) where id is the trajectory identifier which is a scalar attribute. Note that $P([x, y] \mid [x, y] \subseteq [3, 7])$ is the result of the ER query on the window $[x, y]$ during the time interval $[2, 9]$ as discussed earlier, and similarly is $P([x, y] \mid [x, y] \subseteq [10, 14])$.

The following discussion presents our approach to evaluate the answer for region expressions over uncertain trajectories. For the rest of the discussion let τ denotes a 1-dimensional u -trajectory.

Evaluating the Region Expression $\tau.Loc()$. In this section we start with the most general expression $\tau.Loc()$, where τ is a trajectory attribute. The expression has the same syntax as for precise trajectories, however its semantics is completely different.

To make the idea clear we restrict our presentation to a single u-motion u-trajectory. Briefly the algorithm operates as follows: Given a region expression $\tau.Loc()$, and a u-trajectory τ , the goal is to evaluate the answer of the region expression over the u-trajectory τ .

Algorithm: Evaluate $\tau.Loc(e, \tau, (c_1, c_2, c_3, c_4), (id, B))$

Input: Region Expression: $e = \tau.Loc()$, A 1-dimensional single u-motion u-trajectory:
 $\tau = (a_0, m, a_1)$, Characteristic points: (c_1, c_2, c_3, c_4)
Output: (id, B) : id is a scalar, and B is a basis
 Identify u-motion basis $B = \{(b_1, P_1), (b_2, P_2) \cdots (b_k, P_k)\}$;
foreach interval $b_i (1 \leq i \leq k)$ **do**
 $P_i = \text{Compute_Region_Probability}([x, y] \mid [x, y] \subseteq b_i, \tau)$;
end

Fig. 2. Algorithm: Evaluate $\tau.Loc$

The algorithm in Figure 2 presents the steps for evaluating the answer for $\tau.Loc()$, where τ is a u-trajectory. First, using the characteristic points the basis B is identified. Next, for each interval b_i in B , b_i is an interval in the resulting set of pairs $\{(b_1, P_1), \dots, (b_k, P_k)\}$ where $1 \leq i \leq k$, we compute the probability P_i of the trajectory being in this interval.

As mentioned before, the probability associated with each interval is the probability of the ER query on τ during the time interval where the u-motion considered is defined. This is a reasonable probability computation as we are interested in the probability of the object existing in the region at some time during the time interval of interest.

In the algorithm in Figure 2 the function `Compute_Region_Probability` (b, τ) simply takes an interval b and a u-trajectory τ as inputs, and executes the ER evaluation technique over the given interval for the time interval of the u-motion of interest (i.e. defined in τ representation) and returns the resulting probability.

Since, motions can have different orientations, the different cases can be treated similarly by adjusting the order of the characteristic points.

Evaluating the Region Expression $\tau.Loc(e_t)$. In this discussion we present another version of the algorithm in Figure 2. In this case, we are only interested in the regions that the u-trajectory will visit during a specified time interval. The time interval is determined by the time expression e_t similar to the time expressions presented in Section 2. More specifically, to evaluate the region expression $\tau.Loc(e_t)$ for the u-trajectory $\tau = (a_0, m_0, a_1, m_1, \dots, a_k, m_k)$, we need first to identify which part of τ is defined during the time interval satisfying e_t .

This is done as follows: let $[t_1, t_2]$ be the time interval of e_t . Then, the result is a u-trajectory for the part of τ restricted to the interval $[t_1, t_2]$. We denote the restricted τ by $\tau|_{[t_1, t_2]}$. If $t_1 \geq a_k$, then simply $\tau|_{[t_1, t_2]} = (t_1, m_k)$. Otherwise, we basically check the different motions in τ , a motion m_i is in $\tau|_{[t_1, t_2]}$ if it is defined during $[t_1, t_2]$. To determine if m_i is in $\tau|_{[t_1, t_2]}$, let $a_{k+1} = \infty$. For each time interval $[a_i, a_{i+1}]$, $0 \leq i \leq k$, let $[l_1, l_2]$ denote the time interval for $[a_i, a_{i+1}] \cap [t_1, t_2]$. If $[a_i, a_{i+1}] \cap [t_1, t_2] = \emptyset$,

then skip m_i . Otherwise, add (l_1, m_i, l_2) to $\tau \upharpoonright_{[t_1, t_2]}$. Finally, if $t_2 \geq a_k$, then add (l_1, m_k) to $\tau \upharpoonright_{[t_1, t_2]}$.

Now, the evaluation of the expression $\tau.Loc(e_t)$, can simply be viewed as several applications of the algorithm in Figure 2 on each u-motion in $\tau \upharpoonright_{[t_1, t_2]}$.

Evaluating the Region Expression $[l, u] \cap \tau.Loc()$. Here we present another region expression that extends the queries expressible over uncertain trajectories. The region expression is “[l, u] \cap $\tau.Loc()$ ” which defines the possible regions that a u-trajectory τ can visit, and that interest with a given region $[l, u]$.

Evaluating this kind of expressions is an extension over the evaluation of the expression $\tau.Loc()$. For this extension, we are limiting the basis of interest to those intersecting the given region. We can easily observe that the given region could overlap with more than one basis, computing the probability of the region can be evaluated according to the following definition.

Definition 8. Let r be a region and $B = \{(b_1, P_1), (b_2, P_2) \cdots (b_k, P_k)\}$ be a basis. The intersection probability of r and b_i (for $1 \leq i \leq k$), denoted by $r \cap_P b_i$, is the probability of the ER query with a query window $r \cap b_i$. The intersection probability of r and B , denoted by $r \cap_P B$, is thus defined as: $\max_{P_1 \leq i \leq k} (r \cap_P b_i)$ for $1 \leq l \leq n$. Where $r = \bigcup_{1 \leq l \leq n} r_l$, and $\forall (1 \leq l \leq n) r_l \subseteq b_i$ such that, $\forall (1 \leq i, j \leq k) b_i \cap b_j = \emptyset$.

Obviously, the algorithm in Figure 2 can now be employed to compute the answer for $\tau.Loc()$. Once this answer is obtained, the probability can be adjusted based on the probability of the intersection of each interval in the basis and the given region $[l, u]$ as defined in Definition 8.

In Definition 8, the primary reason of choosing the *max* function to compute the intersection probability, is because we are trying to find the probability of the u-trajectory existing in a region. This is logically equivalent to being in the region at any time instant. Naturally, this corresponds to disjunction of being at different spatial intervals of the region. Fagin showed in [7] that the *max* function is the only function to preserve logical equivalence of disjunctive case in computing ranking information of the retrieved answers. It seems that *max* is also a suitable choice for our case. Having defined our restricted set of region expressions. We now consider our algebra operators. The same operators defined in Section 2 are applicable over uncertain trajectories. The only operator that needs extension is the selection operator. The reason for the need of a modified selection operator is that currently our selection is no longer based on truth values of tuples but rather on probabilistic values.

Definition 9. Given a selection formula, a probabilistic selection formula is a selection formula that satisfies a probability threshold condition.

The following example illustrates some queries expressed using the algebra operators over uncertain trajectories.

Example 7. Consider again the u-trajectory of Example 4 where $\tau_x = ((2, (\mu = t + 3, \delta = 2), 9), (9, (\mu = \frac{-2}{3}t + 18, \delta = 2), 20))$.

Let d be a database with relations *Buses* and *Cities*. Assume the relation *Buses* has 2 attributes A that is scalar and B which is a u-trajectory attribute. And relation *Cities* has 2 attributes C which is scalar and D which is a region attribute.

The goal is to evaluate the following query: “Which buses were probably in SB area between 2:00pm and 7:00pm?”. To make our presentation clear let: a denotes the time instant 2:00pm, and b denotes the time instant 7:00pm.

Using our algebra, Q can be expressed as follows:

Let α denotes the expression ‘C=SB’ and

β denotes the expression ‘(B.Loc([a, b]) \cap D) > 0’, then,

$$Q = \pi_A \sigma_{\alpha \wedge \beta} Buses \times Cities$$

Alternatively,

$$Q = \pi_A \sigma_{\beta} Buses \times (\pi_D \sigma_{\alpha} Cities)$$

Clearly, $\pi_D \sigma_{\alpha} Cities$ can be easily evaluated using the semantics of the algebra operators. The key part is actually evaluating $\pi_A \sigma_{\beta} Buses$. The answer for $\pi_A \sigma_{\beta} Buses$ is expected to be a set of tuples with probability values. Let $\omega = (BUS12, \tau_x)$ be a tuple in $Buses$, where τ_x is as defined above. And, let the answer of $\pi_D \sigma_{\alpha} Cities$ be the linear constraints expressing the spatial interval [5, 7.5].

The evaluation of $\pi_A \sigma_{\beta} Buses$ on the tuple ω proceeds as follows:

1. Identify $\tau' = \tau|_{e_t}$ as discussed in Section 4.1. The output is thus: $\tau'_x = (2, (\mu = t + 3, \delta = 2, 7))$ which is actually a portion of the u-motion in Figure 1 over the time interval [2, 7].
2. Identify the characteristic points (3, 7, 8, 12)
3. Identify the basis $B = \{((-\infty, 3), 0), ([3, 7], P_1), ([7, 8], 1), ([8, 12], P_2), ([12, \infty), 0)\}$
4. Identify the basis B' of intervals in B intersecting with [5, 7.5] this will return $B' = \{([5, 7], P'_1), ([7, 7.5], 1)\}$. Where P'_1 is the intersection probability: $[3, 7] \cap [5, 7.5]$ (recall this is the ER query over the window $[3, 7] \cap [5, 7.5]$ and for the time interval [2, 7]). Assume P'_1 has a value 60%.

Thus, $\pi_A \sigma_{\beta} Buses = \{(BUS12, ([5, 7], 0.6)), (BUS12, ([7, 7.5], 1))\}$ Taking the maximum of the probabilities in the result, we get the an answer for $Q = (BUS12, 1)$. This evaluation is then repeated on all the tuples in $Buses$.

5 Summary

In this paper we presented an algebra for querying precise and uncertain trajectories. We introduced the notion of region and time expressions and how those expressions can express a general class of trajectory properties. We showed that a restricted version of the algebra can be employed to query trajectories with uncertainty. And presented techniques for evaluating region expressions over uncertain trajectories.

References

1. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 551–562 (2003)
2. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.S.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: Proc. Int. Conf. on Very Large Data Bases, Toronto, Canada, pp. 876–887 (2004)

3. Codd, E.F.: A relational model of data for large shared data banks. *Communications of the ACM* 13(6), 377–387 (1970)
4. du Mouza, C., Rigaux, P.: Mobility patterns. In: *Proceedings of the Second Workshop on Spatio-Temporal Database Management (STDBM 2004)*, Toronto, Canada, pp. 1–8 (August 2004)
5. Erwig, M., Guting, R.H., Schneider, M., Vazirgiannis, M.: Spatio-temporal data types: an approach to modeling and querying moving objects in databases. *GeoInformatica* 3(3), 269–296 (1999)
6. Erwig, M., Schneider, M.: Spatio-temporal predicates. *IEEE Transactions on Knowledge and Data Engineering* 14(4), 881–901 (2002)
7. Fagin, R.: Combining fuzzy information from multiple systems (extended abstract). In: *Proc. ACM Symp. on Principles of Database Systems*, pp. 216–226. ACM Press, New York (1996)
8. Forlizzi, L., Guting, R.H., Nardelli, E., Schneider, M.: A data model and data structures for moving objects databases. In: *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 319–330 (2000)
9. Grumbach, S., Su, J., TOLLU, C.: Linear constraint query languages: Expressive power and complexity. In: *LCC 1994. LNCS*, vol. 960, pp. 426–446. Springer, Heidelberg (1994)
10. Gütting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider, M., Varirgiannis, M.: A foundation for representing and querying moving objects. *ACM Transactions on Database Systems* 25(1), 1–42 (2000)
11. Kanellakis, P., Kuper, G., Revesz, P.: Constraint query languages. *Journal of Computer and System Sciences* 51(1), 26–52 (1995)
12. Kuper, G., Libkin, L., Paredarns, J. (eds.): *Constraint Databases*. Springer, Heidelberg (2000)
13. Mokhtar, H., Su, J.: A query language for moving object trajectories. In: *Proc. Int. Conf. on Statistical and Scientific Database Management*, Santa Barbara, California, June 27–29, pp. 173–182 (2005)
14. Mokhtar, H., Su, J., Ibarra, O.: On moving object queries. In: *Proc. ACM Symp. on Principles of Database Systems*, pp. 188–198 (2002)
15. Mokhtar, H.M.O., Su, J.: Universal trajectory queries for moving object databases. In: *IEEE IntConf on Mobile Data Management (MDM 2004)*, Berkeley, California, January 19 - 22, pp. 133–144 (2004)
16. Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-object representations. In: Gütting, R.H., Papadias, D., Lochovsky, F.H. (eds.) *SSD 1999. LNCS*, vol. 1651, pp. 111–132. Springer, Heidelberg (1999)
17. Pfoser, D., Tryfona, N.: Capturing fuzziness and uncertainty of spatiotemporal objects. In: *Proc. of the East European Conference on Advances in Databases and Information Systems*, pp. 112–126 (2001)
18. Sistla, A.P., Wolfson, O., Chamberlain, S., Dao, S.: Modeling and querying moving objects. In: *Proc. Int. Conf. on Data Engineering*, pp. 422–432 (1997)
19. Su, J., Xu, H., Ibarra, O.: Moving objects: Logical relationships and queries. In: *Proc. Int. Sym. on Spatial and Temporal Databases*, pp. 3–19 (2001)
20. Trajcevski, G., Wolfson, O., Zhang, F., Chamberlain, S.: The geometry of uncertainty in moving objects databases. In: Chaudhri, A.B., Unland, R., Djeraba, C., Lindner, W. (eds.) *EDBT 2002. LNCS*, vol. 2490, pp. 233–250. Springer, Heidelberg (2002)
21. Wolfson, O., Xu, B., Chamberlain, S., Jiang, L.: Moving objects databases: issues and solutions. In: *Proc. Int. Conf. on Statistical and Scientific Database Management*, pp. 111–122 (1998)

Extending Fagin's Algorithm for More Users Based on Multidimensional B-Tree

Matúš Ondreička and Jaroslav Pokorný

Charles University, Faculty of Mathematics and Physics, Prague
matus.ondreicka@gmail.com, jaroslav.pokorny@mff.cuni.cz

Abstract. We discuss the issue of searching the best K objects in more attributes for more users. Every user prefers objects in different ways. User preferences are modelled locally with a fuzzy function and globally with an aggregation function. Also, we discuss the issue of searching the best K objects without accessing all objects. We deal with the use of local preferences when computing Fagin's algorithms. We created a new model of lists for Fagin's algorithms based on B^+ -trees. Furthermore, we use a multidimensional B-tree (MDB-tree) for searching the best K objects. We developed an MD-algorithm, which can effectively find the best K objects in a MDB-tree in accordance with user's preferences and without accessing all the objects. We show that MD-algorithm achieves better results in the number of accessed objects than Fagin's algorithms.

1 Introduction

Nowadays, there exist big amounts of data in different databases describing real world objects. Following this data, users try to find objects which they need. Every object has its own properties given by a set of attributes. Based on values of these attributes, users then decide which of the objects is more or less suitable for them. Well-known examples cover internet shopping (cameras, mobile phones, etc.), even from various sources, choosing a hotel, flat, job, etc. Another example concerns multimedia repositories having multiple types of attributes, where unlike the relational model, user may want a set of objects with a grade of match [5].

We deal with finding the best K objects from the set of objects X , which have m attributes A_1, \dots, A_m . Making it possible to find the best K objects from the set of objects X , it must be possible to judge which objects are better or worse for a user U . Every user prefers objects with own preferences, which are modelled locally by means of a fuzzy function and globally by means of an aggregation function [8]. The problem of searching the best K object according to values of different attributes in the same time is indicated as a *top-k* problem.

In the paper we discuss the problem of finding the best K objects via Fagin's algorithms [3] without accessing all the objects. We deal with using local preferences when computing the algorithms, which assume that the set X is stored in m lists [3]. We describe a new model of the lists based on B^+ -trees [1], which

supports the use of local preferences [2]. Moreover, it is possible to update these lists easily and quickly.

The main contribution of the paper is use of a multidimensional B-tree (*MDB-tree*) [7] for searching the best K objects. It is not possible to use Fagin's algorithms directly in an MDB-tree. We developed an *MD-algorithm* for top-k problem, which can find the best K objects in MDB-tree effectively, in accordance to user's preferences without accessing all the objects.

MDB-tree allows to index set of objects X by more attributes in one data structure. MDB-tree also makes updating quite easy. The use of B⁺-trees and MDB-tree allows to dynamise the environment while solving a top-k problem. Moreover, these tree structures are independent from user's preferences; i.e. they are common for all users. We will show advantages of MD-algorithm in time complexity in comparison to the rest of considered algorithms.

A background concerning modelling user preferences is contained in Section 2. Section 3 is devoted to explaining principles of Fagin's algorithms. Sections 4 and 5 present use of B⁺-trees and MDB-trees, respectively, for approaching the top-k problem. In Section 6 we present the results of the tests and compare MD-algorithm with Fagin's algorithms. Finally, Section 7 provides some suggestions for a future work.

2 Model of User Preferences

When modelling user's preferences, we differentiate between local preferences and global preferences. When searching for the best K objects, a user U chooses user's preferences, which determine propriety of the object $x \in X$ in dependence with its m values of attributes a_1^x, \dots, a_m^x . In accordance to user U preferences, it is possible to find the best K object for this user.

Local preferences reflect how the object is preferred according to only i -th attribute A_i , $i = 1, \dots, m$. If attribute A_i is of numeric type, then it is possible to apply a *fuzzy function* for this attribute. In this work, a fuzzy function is understood as a mapping $f_i : R_i \rightarrow [0, 1]$, which maps the domain $dom(A_i) \subseteq R_i$ into $[0, 1]$ interval. Local preferences of user U for the attributes A_1, \dots, A_m are represented by user's fuzzy functions denoted as $f_1^U(x), \dots, f_m^U(x)$, respectively. In this case, a user's fuzzy function has a scheme

$$f_i^U(x) : a_i^x \rightarrow [0, 1], \text{ where } i = 1, \dots, m,$$

which maps every object $x \in X$ according to the value of its i -th attribute a_i^x into interval $[0, 1]$. For the worst object $x \in X$, $f_i^U(x)=0$ holds and for the best one, $f_i^U(x)=1$. Comparing $f_i^U(x_1)$ and $f_i^U(x_2)$ of two objects x_1 and x_2 , it is possible to decide which of them is suitable for the user U according to the value of i -th attribute. When $f_i^U(x_1) > f_i^U(x_2)$, then x_1 is more suitable. When $f_i^U(x_1) = f_i^U(x_2)$, then x_1 and x_2 are equally suitable.

We can understand the fuzzy function f_i^U as a new order of $dom(A_i)$. Particular values of attributes a_i^x do not affect this. For descending order of objects X according to i -th attribute A_i , only the course of fuzzy function $f_i^U(x)$ is used.

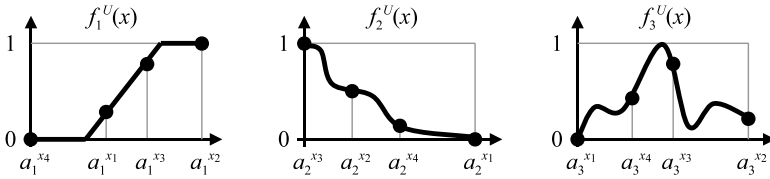


Fig. 1. Ordering interpretation of fuzzy function

Figure 1 shows the arranging interpretation of user’s fuzzy functions $f_1^U(x)$, $f_2^U(x)$, and $f_3^U(x)$. So $f_1^U(x)$ originates order of objects x_2, x_3, x_1, x_4 , for $f_2^U(x)$ order of objects x_3, x_2, x_4, x_1 , and for $f_3^U(x)$ order of objects x_3, x_4, x_2, x_1 .

Global preferences express how the user U prefers objects from X according to all attributes A_1, \dots, A_m . On the set of objects X , we consider as aggregation function $@$ with m variables p_1, \dots, p_m specified as $@(p_1, \dots, p_m) : [0, 1]^m \rightarrow [0, 1]$. For the user U with his user fuzzy functions f_1^U, \dots, f_m^U , a user rating function $@^U$ originates by means of substitution of $p_i = f_i^U(x)$, $i = 1, \dots, m$. Then, for every $x \in X$

$$@^U(x) = @(f_1^U(x), \dots, f_m^U(x)).$$

With $@^U(x)$ it is possible to evaluate global rating of each object $x \in X$. Also here $@^U(x) = 1$ stands for the best object and $@^U(x) = 0$ for the worst one. Comparing $@^U(x_1)$ and $@^U(x_2)$ of two objects x_1 and x_2 it is possible to judge which one is better for the user U .

With aggregation function, a user U can define the mutual relations of the attributes. For example, we can use arithmetic average $@^U(x) = (p_1 + \dots + p_m)/m$. By choosing the right type of representation of aggregate function, it is possible to define attributes which are of interest for the user U , or which are not. For implementation of user’s influence to the aggregate function, it is possible to use weighted average, where weights u_1, \dots, u_m of single attributes A_1, \dots, A_m determine how the user prefers single attributes,

$$@(x) = \frac{u_1 \cdot p_1 + \dots + u_m \cdot p_m}{u_1 + \dots + u_m}.$$

When the user does not care about i -th attribute A_i when rating the objects, he can then put 0 into the aggregate function, i.e. $u_i = 0$.

In this work we used functions f_1^U, \dots, f_m^U and $@^U$ to model user preferences. Every object $x \in X$ has its own special global rating $@(x)$, a measure of pertinence for the user U . In this way it is possible to find the best K objects for the user U . When there is more objects with the same rating as rating of the best K -th object, a random object is chosen.

3 Top-k Problem

The easiest solution of how to find the best K objects for the user U is the basic algorithm. For every object $x \in X$ the values of all m attributes are read and

the global rating $@^U(x)$ is calculated. Then K objects with the highest rating $@^U(x)$ are chosen.

Basic algorithm must read all m values of attributes of all the objects from the set X ; this means making $|X| \cdot m$ accesses. During the search e.g. for the best $K = 10$ objects, it is not effective to search in hundred thousands of objects.

3.1 Fagin's Algorithms

In [3], Fagin et al. describe top-k algorithms, which solve top-k problem without searching in all objects. Fagin's algorithms can find the best K objects with regard to existing aggregate function $@$ without making $|X| \cdot m$ accesses.

Fagin's algorithms assume that the objects from the set X are, together with values their m attributes A_1, \dots, A_m , stored in m lists L_1, \dots, L_m , where i -th list L_i contain pairs (x, a_i^x) . Lists L_1, \dots, L_m are sorted in descending order according to the values of attributes a_1^x, \dots, a_m^x of objects $x \in X$.

The aggregate function $@$ must be monotone according to the ordering in lists. Then Fagin's algorithms find the best K objects correctly without searching all objects [3].

Function f with m variables is *monotone* according to all variables when

$$\forall i \in \{1, \dots, m\} : p_i \leq q_i \Rightarrow f(p_1, \dots, p_m) \leq f(q_1, \dots, q_m)$$

holds. Monotone function f can be, according to all variables, nonincreasing or nondecreasing. Lists L_1, \dots, L_m are sorted in descending order according to the values of attributes of objects. For the right functionality of Fagin's algorithm it is needed to use this nondecreasing aggregate function $@$ (e.g. arithmetic or weighted average).

To the objects which are stored as (x, a_i^x) in the lists, it is possible to access by two different methods. *Direct access* allows in the i -th list L_i directly get the object x regardless of the position of the pair (x, a_i^x) in the list L_i . *Sequential access* allows the use of the objects in the list step by step from the top to the bottom by single pairs (x, a_i^x) .

According to the way of accessing to the lists L_1, \dots, L_m , two main algorithms TA (threshold algorithm) and NRA (no random access) are showed in [3].

Algorithm NRA uses the objects in the lists L_1, \dots, L_m from the top to the bottom during the sequential accessing. It gets pairs (x, a_i^x) in the decreasing order according to the values a_i^x . For these objects $x \in X$ algorithm NRA does not know all the values of its attributes; therefore, it counts the *worst rating* $W(x)$ and the *best rating* $B(x)$ of the objects x . The monotonicity of $@$ then implies $W(x) \leq @(x) \leq B(x)$. Then it is possible to guess the value of not yet seen objects from the lists L_1, \dots, L_m and algorithm NRA can stop earlier than it comes to the end of all the lists.

Algorithm TA needs to access the lists L_1, \dots, L_m sequentially and also directly. With the sequential access, TA searches the lists L_1, \dots, L_m from the top to the bottom and gets pairs (x, a_i^x) . For every object x obtained by a direct access to the other lists the algorithm gets the missing values of attributes of the object x . TA uses the list T_K , in which it keeps the best actual K objects ordered

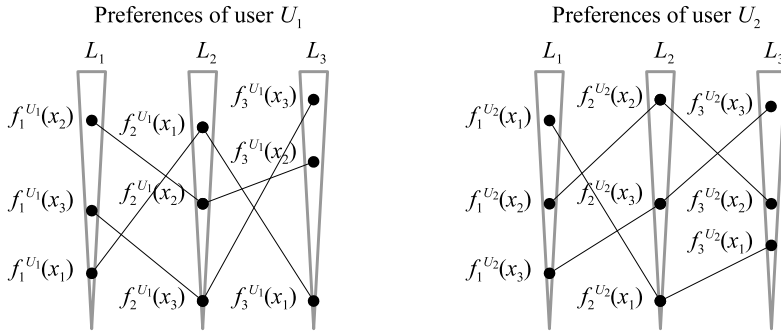


Fig. 2. Lists of two users with different preferences

according to their value. During the run of TA a threshold is counted, which is calculated by means of inserting the last seen attributes values $a_1^{LAST}, \dots, a_m^{LAST}$ in the lists L_1, \dots, L_m , respectively, at the sequential access to the aggregate function $@(a_1^{LAST}, \dots, a_m^{LAST})$. When K -th object of the list T_K has higher value than threshold, TA stops and returns the list T_K as the best K objects. TA can finish before it comes to the end of all the lists.

3.2 Multi-user Solution

Original Fagin’s algorithms offer the possibility of rating objects only globally with an aggregate function $@$, find the best K object for the user U only according to his global preference. For the multi-user solution with support local preferences, it is necessary that every i -th list L_i contains pairs $(x, f_i^U(x))$ in descending order by $f_i^U(x)$ before the algorithm computation.

However, every user U prefers objects $x \in X$ locally with different fuzzy functions $f_1^U(x), \dots, f_m^U(x)$. For finding the best K objects for the user U , it would be needed to create descending ordered lists L_1, \dots, L_m according to preferences f_1^U, \dots, f_m^U always before the start of Fagin’s algorithm. For a different user, different lists L_1, \dots, L_m will be needed to be created again. For example, in Figure 2 three descending ordered lists L_1, L_2, L_3 of two users U_1 and U_2 , who prefer objects with different fuzzy functions locally, are shown. User U_1 prefers objects x_1, x_2, x_3 with $f_1^{U_1}, f_2^{U_1}, f_3^{U_1}$ and user U_2 with $f_1^{U_2}, f_2^{U_2}, f_3^{U_2}$.

Before the start of finding, it is necessary to get all the values of attributes of all objects $x \in X$ and evaluate values $f_1^U(x), \dots, f_m^U(x)$. In this case it is meaningless to use Fagin’s algorithm, which is trying to find the best K objects without making $|X| \cdot m$ accesses.

4 Fagin’s Lists Based on B⁺-Tree

The problem of creating lists according to user’s preferences before starting Fagin’s algorithm is possible to solve by the use of B⁺-trees. In B⁺-tree [1], the

objects from X are indexed according to the values of only one attribute. We use a variant of the B^+ -tree, whose leaf nodes are linked in two directions. Every leaf node of B^+ -tree contains pointer to its left and right neighbor.

Let the objects of the set X be indexed in the B^+ -tree according to the values of attribute A . Every object $x \in X$ is indexed by the key $k \in \text{dom}(A)$, a value of which is equal to a value of attribute a^x of object x , i.e. $k = a^x$. For a key k , in general, more objects with the same A attribute value can exist in X . Therefore, we use the *object array*. In the object array, objects with the same value of the key k , are stored. For every the key k there is a pointer to this object array.

4.1 A List Modelling with B^+ -Tree

Fagin's algorithm needs during its computation a sequential accessing the lists L_1, \dots, L_m . From each i -th list L_i it is needed sequentially to get objects $x \in X$ in the descending order according to $f_i^U(x)$, i.e. the pairs $(x, f_i^U(x))$. Therefore it is not necessary to get the whole list before starting the algorithm.

Since the leaf nodes of the B^+ -tree are linked, it is possible to cross the B^+ -tree through the leaf level and to get all the keys. In this case, the B^+ -tree is traversed top to bottom only one time (starting at the root). With the pointers to the object arrays it is possible to get objects from X ordered according to the value of A attribute. When we cross the leaf level from left to the right, it is possible to get sequentially pairs (x, a^x) in ascending order according to a^x and from the right to the left vice versa.

When the user's fuzzy function f^U is monotone on its domain, then for getting the pairs from the B^+ -tree using users preference f^U the following holds:

1. Let the f^U be nondecreasing. Thus $a^x \leq a^y \Rightarrow f^U(x) \leq f^U(y)$ holds for any two objects $x, y \in X$. Then by crossing the leaf level of the B^+ -tree from the right to the left it is possible to get the pairs $(x, f^U(x))$ of the list L in the descending order according to the user's preference f^U .
2. Let the f^U be nonincreasing. Thus $a^x \geq a^y \Rightarrow f^U(x) \leq f^U(y)$ holds for any two objects $x, y \in X$. Then by crossing the leaf level of the B^+ -tree from the left to the right it is possible to get the pairs $(x, f^U(x))$ of the list L in the ascending order according to the user's preference f^U .

During getting the object x from the B^+ -tree, we get the pair (x, a^x) , then $f^U(x)$ is calculated and the pair $(x, f^U(x))$ is formatted. These pairs of the list L are evaluated according to f^U , and contain local user preference.

For getting objects from the B^+ -tree we use the term *way*. A way is an oriented line segment, which follows crossing the leaf level of B^+ -tree during getting the objects. Every way w has its beginning w^{beg} , end w^{end} , and direction (to the left or to the right). Then, we get the objects, pairs (x, a^x) , with attribute values $w^{beg} \geq a^x \geq w^{end}$ ($w^{beg} \leq a^x \leq w^{end}$) according to a direction of w .

In general, f^U is not monotone in its domain and for getting pairs of list L according to f^U we can't use only one way. The domain of definition f^U can be divided into continuous intervals I_1, \dots, I_n , that f^U is monotone on every of these intervals. According to the intervals I_1, \dots, I_n it is possible to create the

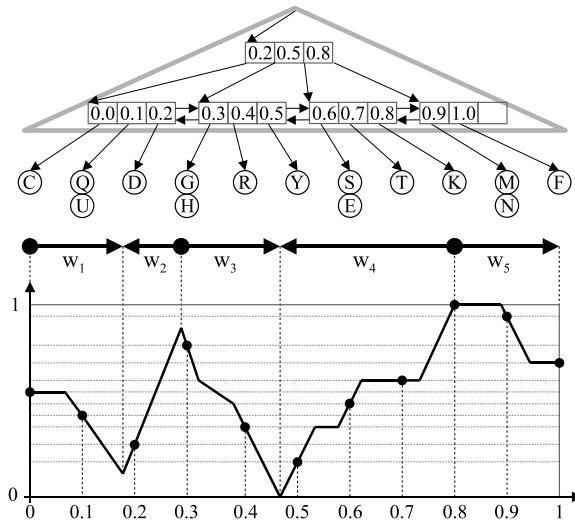


Fig. 3. Creating ways according to progress of fuzzy function

ways w_1, \dots, w_n , that every way w_i has its begin and end border points of interval I_i . The direction of the way w_i is to the left, when f^U is nondecreasing on the interval I_i , or to the right, when it is f^U nonincreasing there. Then ways w_1, \dots, w_n specify how to cross the leaf level of the B^+ -tree during getting objects according to the users preference f^U . For example, in Figure 3, the course of the fuzzy function originated five ways w_1, \dots, w_5 .

When there are more ways w_1, \dots, w_n , for each way B^+ -tree is traversed top to bottom only one time and then crossed only through the leaf level. On every way there can exist (first) object $x \in X$ with different value $f^U(x) \in [0, 1]$. Objects from the particular ways have to be obtained in overall descending order by $f^U(x)$ as pairs $(x, f^U(x))$ of the list L . Consequently, during getting the pairs from the B^+ -tree along more various ways it is needed to access all the ways concurrently. For example, in Figure 3 we show a fuzzy function, serving to get objects according to ways w_1, \dots, w_5 from the B^+ -tree. We get objects $x_K, x_M, x_N, x_G, x_H, x_F, x_T, x_S, x_E, x_C, x_Q, x_U, x_R, x_D, x_Y$.

Next procedures describe getting pairs $(x, f^U(x))$ from the B^+ -tree.

```

Input: BplusTree  $B^+$ -tree, FuzzyFunction  $f^U$ ;
begin
  activateWays( $B^+$ -tree,  $f^U$ );
  while(exist at least one active way)do
    getRatedObject( $B^+$ -tree,  $f^U$ );
  endwhile;
end.

```

procedure activateWays(BplusTree B^+ -tree, FuzzyFunction f^U);

1. According to the course of f^U create ways w_1, \dots, w_n , mark them as active;
 2. According to all the ways w_1, \dots, w_n , try to get objects x_1, \dots, x_n in B^+ -tree.
The ways which were not successful in this process are marked as inactive.
 3. Recalculate $f^U(x_j)$ of each gained object x_j (from active way w_j);
- end;

procedure getRatedObject(BplusTree B^+ -tree, FuzzyFunction f^U);

1. When there is not active way, it is not possible to return any pair;
 2. From active ways choose object x_j with the highest value $f^U(x_j)$.
Create the pair $(x_j, f^U(x_j))$ and delete object x_j from the way w_j ;
 3. When is possible, according to the way w_j get next object x'_j in the B^+ -tree and recalculate its $f^U(x'_j)$. Otherwise, if next gained object has value of attribute after the w_j^{end} , mark w_j as an inactive way;
 4. Return the pair $(x_j, f^U(x_j))$;
- end;

4.2 Application of the Model in Fagin's Algorithms

We suppose that the objects from X have m values attributes. Then Fagin's algorithms need access sequentially to m lists L_1, \dots, L_m during a computation. We use as the lists m of B^+ -trees B_1, \dots, B_m . In B^+ -tree B_i all objects are indexed by values of i -th attribute A_i . It is needed to create m of B^+ -trees before the first run of TA or NRA. Afterwards this structure is common for all users.

A particular user U only specifies his local preferences with fuzzy functions f_1^U, \dots, f_m^U and global preferences with function $@^U$. Then the algorithm sequentially gets pairs $(x, f_i^U(x))$ from B^+ -trees B_1, \dots, B_m according to preferences of user U , find best K object for user U . Then it is running again for each new user's preferences or for preferences of next user.

Algorithm TA also uses the direct access to the lists L_1, \dots, L_m , where for object x it is needed to get its unknown value a_i^x from L_i . B^+ -trees are not able to operate this, because it is not possible to get directly the value. A realization of direct access can use, for example, an associative array.

5 Application of Multidimensional B-Tree

This section concerns the use of *MDB-tree* [7] to the solving top-k problem for more users, because MDB-tree is composed of B^+ -trees, which we can use for obtaining pairs $(x, f_i^U(x))$ by user's local preferences f_1^U, \dots, f_m^U .

MDB-tree allows to index set of objects X by attributes A_1, \dots, A_m , $m > 1$, in one data structure. In this case, MDB-tree has m levels and values of one attribute are stored in each level. We use a variant of MDB-tree, nodes of which are B^+ -trees. i -th level of MDB-tree is composed from B^+ -trees containing key values from $dom(A_i)$. Analogically to B^+ -trees, we introduce the *object array*. In the object array, we store objects with the same values of all the m attributes

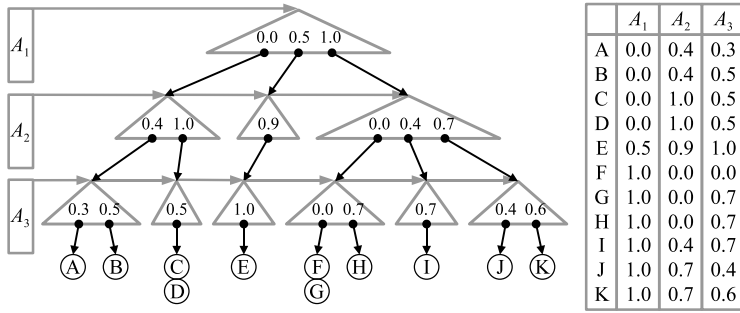


Fig. 4. Set of eleven objects with values of three attributes stored in MDB-tree

a_1, \dots, a_m . We use the mark $\rho(k)$ for the pointer of the key k in B^+ -tree. If k_i is the key in B^+ -tree and this B^+ -tree is in m -th level of MDB-tree, then $\rho(k_i)$ refers to object array, i.e. ($i = m$). If this B^+ -tree is in i -th level of MDB-tree and $i < m$, then $\rho(k_i)$ refers to B^+ -tree in the next level of MDB-tree.

For the explicit identification of B^+ -tree in MDB-tree we use the order of keys, which is called *tree identifier* in this paper. An identifier of B^+ -tree in i -th level is (k_1, \dots, k_{i-1}) . Tree identifier of B^+ -tree in the first level is (\emptyset) . For example, in Figure 4, $(k_1, k_2) = (1.0, 0.0)$ is the identifier of B^+ -tree at the third level, which contains keys 0.0, 0.7 and refers to objects x_F, x_G, x_H .

5.1 Search of MDB-Tree

Getting all objects from MDB-tree is possible with procedure `searchMDBtree`, which recursively searches MDB-tree in depth-first search.

```

Input: MDBtree MDB-tree;
Output: List T;
var List T;                {temporary list of objects}
begin
  searchMDBtree(MDB-tree, ( $\emptyset$ ));
  return T;
end.

procedure searchMDBtree(MDBtree MDB-tree, Identifier  $(k_1, \dots, k_{i-1})$ );
  while(there is another key in  $B^+$ -tree) do
    In  $B^+$ -tree of MDB-tree with identifier  $(k_1, \dots, k_{i-1})$ 
    chose another key  $k_i$  with highest value of  $A_i$  in the  $B^+$ -tree;
    if( $\rho(k_i)$  refers to  $B^+$ -tree) then
      searchMDBtree(MDB-tree,  $(k_1, \dots, k_{i-1}, k_i)$ );
    if( $\rho(k_i)$  refers to object array) then
      Append objects from this array to the list T
      also with theirs attributes values  $k_1, \dots, k_{i-1}, k_i$ ;
    endwhile;
  end.
end.
    
```

Searching the whole MDB-tree it is possible to find the best K objects according to the aggregate function $@$. For every new object x from the MDB-tree all its attribute values are known. Thus, it is possible to calculate $@(x)$ immediately. Like in TA, we can use the temporary list T_K , in which the best actual K objects are stored in descending order according to their rating $@(x)$. Rating of the K -th best object in T_K we denote M_K . For every new object obtained from MDB-tree the following will be done:

```

var List TK;           {temporary list of objects}
procedure addRatedObject(RatedObject  $x$ , Aggregation  $@$ , int  $K$ );
  if( |TK| <  $K$  )then
    Insert object  $x$  to the list TK to the right place according to  $@(x)$ ;
  else
    if(  $@(x) > M_K$  )then begin
      Delete  $K$ -th object from the list TK;
      Insert object  $x$  to the list TK on the right place according to  $@(x)$ ;
    end;
end.

```

This procedure obtains all objects of MDB-tree. On the other hand, Fagin's algorithms can find the best K objects without searching all the objects from X which provides a new motivation for development a more optimal solution of top-k problem based on MDB-tree. One of the very useful properties of NRA algorithm is using the best rating $B(x)$ of objects.

In MDB-tree we use the best rating $B(S)$ of B^+ -tree S . Analogically to NRA algorithm we assume, that aggregate function $@(p_1, \dots, p_m)$ is nondecreasing in all variable p_1, \dots, p_m . Also we assume, that all values of attributes of objects are normalized into $[0, 1]$ interval, i.e. $dom(A_j) \subseteq [0, 1]$, $j = 1, \dots, m$.

Definition 1. By the *best rating* $B(S)$ of B^+ -tree S with identifier (k_1, \dots, k_{i-1}) in the i -th level of MDB-tree we understand a rating not yet known object x , calculated with $@(a_1^x, \dots, a_m^x)$, where the first $i - 1$ attributes values of object x are k_1, \dots, k_{i-1} and values of other attributes are 1, i. e.

$$B(S) = @(k_1, \dots, k_{i-1}, 1, \dots, 1)$$

For every B^+ -tree S in MDB-tree there is a uniquely defined subset of X , X^S , which we call a *set of available objects* from S . Getting all objects from X^S is possible with procedure `searchMDBtree`, which starts in S . Let S be the B^+ -tree of i -th level with identifier (k_1, \dots, k_{i-1}) . Then X^S contains all the objects $x \in X$, for which the values of the first $i - 1$ attributes a_1^x, \dots, a_{i-1}^x are equal to values of k_1, \dots, k_{i-1} from identifier of S . For example, in Figure 4, the X^S of S with identifier (0.0) contains objects x_A, x_B, x_C, x_D .

Statement 1. Let S be the B^+ -tree of MDB-tree. Then

$$\forall x \in X^S : @(x) \leq B(S).$$

Proof of Statement 1 is straightforward; it follows from the Definition 1. \square

During getting objects from MDB-tree the procedure `searchMDBtree` starts in B⁺-tree S in i -th level with identifier (k_1, \dots, k_{i-1}) . In S we get step by step keys $k_i \in \text{dom}(A_i)$, with which we can access other parts of MDB-tree, from where we get another objects. When the $\rho(k_i)$ refers to S with identifier $(k_1, \dots, k_{i-1}, k_i)$, the procedure `searchMDBtree` should start there. By Statement 1, we would get objects $x \in X^S$ with rating $@(x) \leq B(S)$. When the list T_K already contains K objects, it is possible to decide if it is desirable to continue with the new key to the other parts of MDB-tree.

Statement 2. Let p be the key in B⁺-tree S with identifier (k_1, \dots, k_{i-1}) chosen in a run of procedure `searchMDBtree`. The $\rho(p)$ refers to B⁺-tree P with identifier (k_1, \dots, k_{i-1}, p) in the next level of MDB-tree. Let M_K be the rating of the K -th best object in T_K . Then

$$B(P) \leq M_K \Rightarrow \forall x \in X^P : @(x) \leq M_K.$$

Proof. When the $\rho(p)$ refers to P , by Statement 1, $\forall x \in X^P : @(x) \leq B(P)$ holds. If list T_K already contains K objects, the M_K value is known. $B(P) \leq M_K$ implies $x \in X^P : @(x) \leq B(P) \leq M_K$. □

Consequence. When the $\rho(p)$ refers to P with identifier (k_1, \dots, k_{i-1}, p) and at the same time $B(P) \leq M_K$ holds, none of objects from X^P is not going to have better evaluation than K -th object from T_K . So it not meaningful to get objects from P , it is not necessary to start procedure `searchMDBtree` (in B⁺-tree P).

Notice 1. Let p be the key in B⁺-tree S chosen in a while cycle in run of procedure `searchMDBtree`. Let q be the key chosen in next its while cycle. Because the keys are gained from S in descending order, then $p \geq q$ holds.

Notice 2. Let S be the B⁺-tree with identifier (k_1, \dots, k_{i-1}) in i -th level of MDB-tree. p and q are the keys gained from S , the pointers $\rho(p)$ and $\rho(q)$ refer to B⁺-trees P and Q , respectively, in the next level of MDB-tree or to object arrays P and Q (S is in the last level of MDB-tree, $i = m$). Then

$$p \geq q \Rightarrow @(k_1, \dots, k_{i-1}, p, 1, \dots, 1) \geq @(k_1, \dots, k_{i-1}, q, 1, \dots, 1) \Rightarrow B(P) \geq B(Q).$$

Statement 3. Let the keys from the B⁺-tree S with the identifier (k_1, \dots, k_{i-1}) be the keys obtained one by one by a run of procedure `searchMDBtree`. The pointer $\rho(p)$ refers to B⁺-tree P in the next level or to the object array P . Let M_K be the rating of the K -th best object in T_K . If $B(P) \leq M_K$ holds, then the procedure `searchMDBtree(MDB-tree, (k1, ..., ki-1))` can stop in S .

Proof. Since the keys are gained from S in descending order, by Notice 1, for each next key q gained $q \leq p$ holds. The pointers $\rho(p)$ and $\rho(q)$ refer to B⁺-trees P and Q , respectively, in the next level of MDB-tree or they refer to the object arrays P and Q . By Notice 2 $B(P) \geq B(Q)$ holds. By Statement 2, for all of the objects $x \in X^Q$ we have $@(x) \leq B(Q) \leq B(P) \leq M_K$, and so $@(x) \leq M_K$. For all next keys $q \leq p$ obtained from S it is not necessary to access B⁺-tree Q (or the object array Q) by the procedure `searchMDBtree(MDB-tree, (k1, ..., ki-1, q))`, because no such object $x \in X^Q$ can not get in T_K . It means that it is not necessary to obtain another keys from B⁺-tree S . □

5.2 MD-Algorithm for Top-k Problem

Using the previous statements results in an *MD-algorithm*, which is able to find in MDB-tree K best objects with respect to a monotone aggregate function $@$, without getting all the objects.

Input: MDBtree $MDB-tree$, Aggregation $@$, int K ;

Output: List TK;

var List TK; {temporary list of objects}

begin

 findTopK($MDB-tree$, (\emptyset) , $@$, K);

 return TK;

end.

procedure findTopK(MDBtree $MDB-tree$, Identifier (k_1, \dots, k_{i-1}) ,

 Aggregation $@$, int K);

while(there is next key in B^+ -tree)do {with identifier (k_1, \dots, k_{i-1}) }

$k_i = \text{getNextKey}(MDB-tree, (k_1, \dots, k_{i-1}))$;

 { $\rho(k_i)$ refers to B^+ -tree P or it refers to the object array P }

 if($|\text{TK}| = K$ and $B(P) \leq M_K$)then return;

 if(P is B^+ -tree)then

 findTopK($MDB-tree$, $(k_1, \dots, k_{i-1}, k_i)$, $@$, K);

 if(P is object array)then

 while(there is the next object x in P)do

 if($|\text{TK}| < K$)then

 insert object x to TK to the right place according to $@(x)$;

 else

 if($@(x) > M_K$)then

 begin

 Delete K -th object from the list TK;

 Insert object x to the list TK on the right place according to $@(x)$;

 end;

 endwhile;

 endwhile;

end.

procedure getNextKey(MDBtree $MDB-tree$, Identifier (k_1, \dots, k_{i-1}));

 In B^+ -tree of $MDB-tree$ with identifier (k_1, \dots, k_{i-1}) chose the next key k_i with next highest value of A_i in that B^+ -tree;

 return k_i ;

end.

5.3 Multi-user Solution

MD-algorithm does not solve the problem of the local use preferences. It finds the K best objects only with respect to a monotone aggregate function. According to use of the model of the user preferences, the values of attributes could be

rated locally according to the users fuzzy functions f_1^U, \dots, f_m^U . After that we can calculate the global rating of x by user aggregate function $@^U$. We use the lists based on B^+ -tree for gaining the keys from B^+ -trees. It is possible to obtain the keys from B^+ -tree with the help of the fuzzy function f^U in descending order.

The A_i values of objects $x \in X$ are in MDB-tree stored (as keys) in B^+ -trees in i -th level of MDB-tree. This justifies why in each B^+ -tree of i -th level we use user fuzzy function f_i^U to obtain required pairs. Thus we obtain the pairs $(k, f_i^U(k))$ from each B^+ -tree in i -th level of MDB-tree in descending order by $f_i^U(k)$. During the calculation of global rating of x the procedure directly substitutes values of keys k_i rated by fuzzy functions $f_i^U(k_i)$ into aggregate function $@$. It results into a global user rating function $@^U(x) = @^U(f_1^U(k_1), \dots, f_m^U(k_m))$. Analogically, we calculate the best rating of B^+ -tree S of i -th level, $B(S) = @(f_1^U(k_1), \dots, f_{i-1}^U(k_{i-1}), 1, \dots, 1)$.

It is possible to use application of the local user preferences directly in MD-algorithm by finding the K best objects in MDB-tree. The following procedure `getNextKey` changes the original MD-algorithm.

```

procedure getNextKey(MDBtree  $MDB\text{-}tree$ , Identifier  $(k_1, \dots, k_{i-1})$ ,
    FuzzyFunction  $f_i^U$ );

```

```

    In  $B^+$ -tree of  $MDB\text{-}tree$  with identifier  $(k_1, \dots, k_{i-1})$  chose the next key  $k_i$ 
    with next highest value of  $i$ -th user fuzzy function  $f_i^U(k_i)$  in that  $B^+$ -tree.;
    return  $k_i$ ;

```

end.

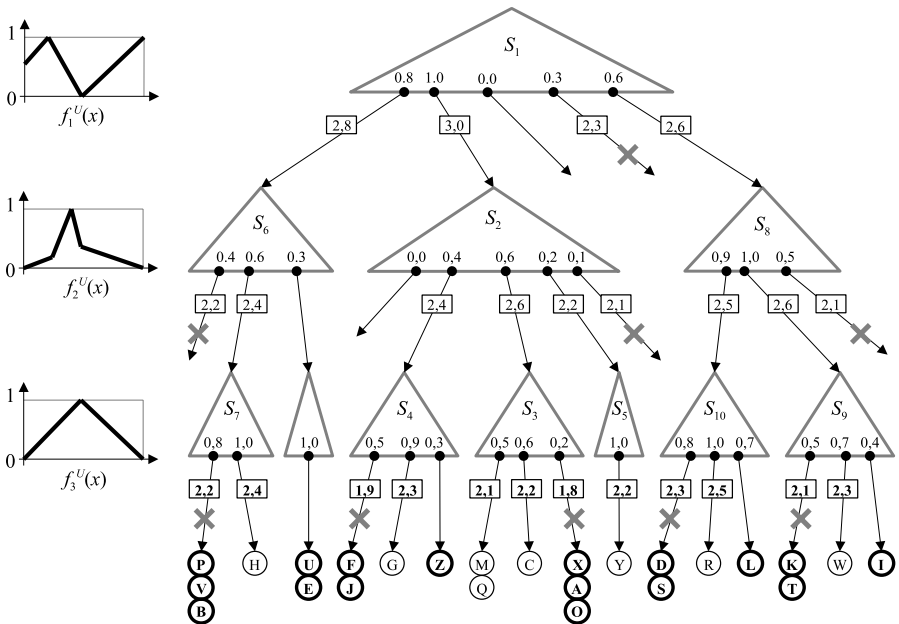


Fig. 5. Example of finding top three objects in MDB-tree

The next example, in Figure 5, illustrates how MD-algorithm finds for three best objects in MDB-tree, with respect to the preference of one concrete user U . The local preferences of user U express the objects according to three attributes as it is shown on left side in Figure 5. For the illustration, we assume that the user U for the determination of the attribute weights has used the aggregate function $@(a_1, a_2, a_3) = a_1 + a_2 + a_3$, i. e. rating of ideal object is 3.

At the beginning procedure `findTopK` is starting in B^+ -tree at the first level. After the procedure stops the list T_K contains three best objects for the user U , even in descending order $@^U(x_R) = 2.5$, $@^U(x_H) = 2.4$ and $@^U(x_G) = 2.3$.

For illustration, that MD-algorithm is able to find K best objects in MDB-tree without visiting all objects in MDB-tree, the objects, which did not have received by MD-algorithm during its calculation, are highlighted in Figure 5.

6 Implementation and Experiments

We have implemented top-k algorithms TA, NRA, and MD-algorithm using lists based on B^+ -trees [6]. The implementation has been developed in Java. The main research point has been to estimate the number of accesses into data structures during calculation of algorithms. The needed data structures have been created in memory (not on disk). It was sufficient solution, because in memory it is possible to calculate easily the numbers of accesses by the simple modifications of the algorithms.

Objects from X with their m attributes values are stored in these data structures. Obtaining one attribute value of one object we conceive as *one access* into data structures. For testing there have been used three sets of 100 000 objects with 5 attributes values with normal, regular, and exponential distribution of attributes values. As user's local preferences we used fuzzy functions whose course was based on $f(x) = x$, as the aggregate function an arithmetic average has been used. In this paper, we use these simple user's preference, because every user can to prefer objects differently, and it is not possible to show there all dependences.

Figure 6 shows results of tests for normal and regular distribution.

The best of results have been achieved with MD-algorithm for the set of objects with the regular distribution of the attributes values. Searching for the

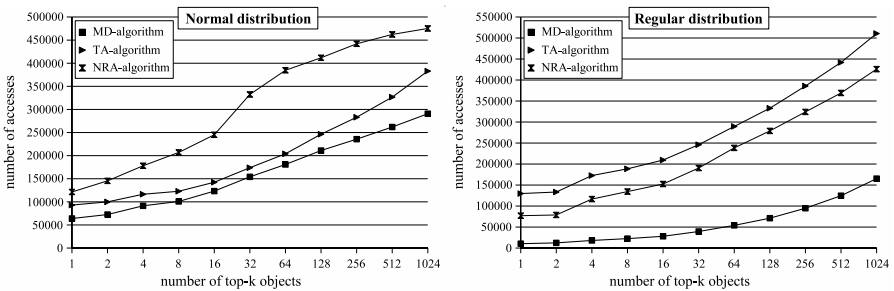


Fig. 6. Distribution of the attributes values

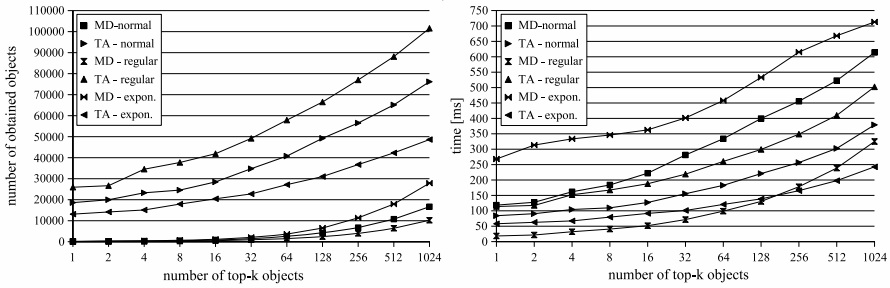


Fig. 7. The number of obtained objects and computation time of algorithms

best object needed approximately 10 times less accesses than algorithms TA and NRA. For algorithms TA and MD-algorithm it is possible to determine also the number of obtained objects from associated data structures. Figure 7 illustrates the results of this comparison. The most significant difference in the rate of the objects obtained by the algorithms TA and MD-algorithm has occurred for regular distribution of the attributes values. For choosing the best objects, MD-algorithm has needed more than 400 times less objects than the algorithm TA.

Testing has shown that the NRA algorithm is slower. While TA algorithm and MD-algorithm with $K = 1024$ we achieved time less than 1s. The calculation of the algorithm has taken several minutes. This fact has been caused by often re-calculation of $W(x)$ and $B(x)$ rating of the objects. Figure 7 illustrates the times of rate for algorithms TA and MD.

We can conclude with a hypothesis that MD-algorithm is more suitable in the case of regular distribution of the attributes values. Moreover, MD-algorithm achieves much better results than other tested algorithms for K best objects just if K is very little.

Concerning other associated results, in [6] we discovered that MD-algorithm has the best results, when the objects stored in MDB-tree are distributed regularly. When attributes of objects have different size of their domains, it is better for MD-algorithm to build MDB-tree with smaller domains in its higher levels and attributes with bigger domains in its lower levels. Then MD-algorithm can to decide early, that it is not necessary to obtain next keys from some B^+ -tree in higher level of MDB-tree. In [6] we were testing Fagin’s algorithms and MD-algorithm also for real data (flats for rent). Because this set of objects has more attributes with different size of domains, it was possible to build regular MDB-tree. MD-algorithm achieves also much better results than other tested algorithms for suitable user’s preferences and small K ($K < 30$). We think, that MD-algorithm is more suitable as Fagin’s algorithms just for real data.

This results confirms the original idea of MDB-trees [7], which was developed for range queries and for minimalizaion number of accesses into data structures.

7 Conclusion

We implemented top-k algorithms TA, NRA and MD-algorithm with support of user's preferences. Results of MD-algorithm have shown to be comparable with those obtained by Fagin's algorithms.

An ideal solution of top-k problem would be to implement Fagin's algorithms with indexes based on B⁺-trees directly in a RDBMS or to implement MD-algorithm with an MDB-tree together. Because of a support of the local users preferences this solution would require to implement indexes with the help of B⁺-trees in the way to be able easily obtain pairs $(x, f_i^U(x))$ while computing Fagin's algorithms or MD-algorithm.

In [4] the authors describe heuristics for Fagin's algorithms. Similarly, a motivation for future research could be to develop some heuristics for MD-algorithm, which improve performance parameters of MD-algorithm. For example, we it is possible to monitor a distribution of the key values in nodes for a selected set of objects stored in MDB-tree, etc.

A combination of MD-algorithm with Fagin's algorithms might be also interesting. For example, in distributed systems we can find some best objects on two different servers by Fagin's algorithms and based on these results to find K best objects by MD-algorithm.

Acknowledgments

This research has been partially supported by the National Program of Research, Information Society Project No. 1ET100300419.

References

1. Bayer, R., McCreight, E.: Organization and Maintenance of Large Ordered Indices. *Acta Informatica* 1(3), 173–189 (1972)
2. Eckhardt, A., Pokorný, J., Vojtáš, P.: A system recommending top-k objects for multiple users preference. In: *Proc. of 2007 IEEE International Conference on Fuzzy Systems*, London, England, July 24–26, 2007, pp. 1101–1106 (2007)
3. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences* 66, 614–656 (2003)
4. Gurský, P., Lencses, R., Vojtáš, P.: Algorithms for user dependent integration of ranked distributed information. In: *Proceedings of TED Conference on e-Government (TCGOV 2005)*, pp. 123–130 (2005)
5. Chaudhuri, S., Gravano, L., Marian, M.: Optimizing Top-k Selection Queries over Multimedia Repositories. *IEEE Trans. On Knowledge and Data Engineering* 16(8), 992–1009 (2004)
6. Ondreička, M.: Extending Fagin's algorithm for more users. Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague (in Slovak) (2008)
7. Scheuerman, P., Ouksel, M.: Multidimensional B-trees for associative searching in database systems. *Information systems* 34(2) (1982)
8. Vojtáš, P.: Fuzzy logic aggregation for semantic web search for the best (top-k) answer. In: *Capturing Intelligence*, ch.17, vol. 1, pp. 341–359 (2006)

Increasing Expressiveness of Composite Events Using Parameter Contexts*

Indrakshi Ray and Wei Huang

Department of Computer Science
Colorado State University
Fort Collins CO 80528-1873

Abstract. The event-condition-action paradigm (also known as *triggers* or *ECA rules*) gives a database “active” capabilities – the ability to react automatically to changes in the database or the environment. Events can be primitive or composite. Primitive events cannot be decomposed. Different types of primitive events can be composed using event composition operators to form composite events. When a composite event occurs, it is possible that many instances of some constituent primitive event occurs. The *context* determines which of these primitive events should be considered for evaluating the composite event. Researchers have identified four different contexts, namely, *recent*, *chronicle*, *continuous*, and *cumulative* that can be associated with a composite event. Associating a single context with a complex composite event is often times not adequate. Many real-world scenarios cannot be expressed if a composite event is associated with a single context. To solve this problem, we need to associate different contexts for the various constituent primitive events. We show how this can be done by providing a formal semantics for associating contexts with primitive events. Finally, we give algorithms for event detection that implement these semantics.

1 Introduction

Traditional database management system are *passive*: the database system executes commands when requested by the user or application program. However, there are many applications where this passive behavior is inadequate. Consider for example, a financial application: whenever the price of stock for a company falls below a given threshold, the user must sell his corresponding stocks. One solution is to add monitoring mechanisms in the application programs modifying the stock prices that will alert the user to such changes. Incorporating monitoring mechanisms in all the relevant application programs is non trivial. The alternate option is to poll periodically and check the stock prices. Polling too frequently incurs a performance penalty; polling too infrequently may result in not getting the desirable functionalities. A better solution is to use active databases.

Active databases move the reactive behavior from the application into the database. This reactive capability is provided by *triggers* also known as *event-condition-action rules* or simply *rules*. In other words, triggers give active databases the capability to

* This work was supported in part by AFOSR under contract number FA9550-07-1-0042.

monitor and react to specific circumstances that are relevant to an application. An active database system must provide trigger processing capabilities in addition to providing all the functionalities of a passive database system.

Different types of events are often supported by a database application: (i) data modification/retrieval events caused by database operations, such as, insert, delete, update, or select, (ii) transaction events caused by transaction commands, such as, begin, abort, and commit, (iii) application-defined events caused by application programs, (iv) temporal events which are raised at some point of time, such as December 25, 2007 at 12:00 p.m. or 2 hours after the database is updated, and (v) external events that occur outside the database, such as, sensor recording temperature above 100 degrees Fahrenheit. Various mechanisms are needed for detecting these different types of events. The types of events that are of interest depend on the specific application and the underlying data model. We do not distinguish between these different types of events in the rest of the paper.

An application may be interested in an occurrence of a single event or in a combination of events. The occurrence of a single event is referred to as a *primitive event*. Primitive events are atomic in nature – they cannot be decomposed into constituent events. Example of a primitive event is that the temperature reaches 100 degrees Fahrenheit. Sometimes an application is more interested in the occurrence of a combination of primitive events. Such an event that is formed by combining multiple primitive events using event composition operators is termed a *composite event*. For example, an application may be interested in the event that occurs when the stock prices of IBM drop after that of Intel. This is an example of a composite event that is made up of two primitive events: stock price of IBM drops and stock price of Intel drops. The event composition operator in this case is the temporal operator “after”.

Many instances of a constituent primitive event may occur before the occurrence of a composite event. In such cases, we need to identify which primitive event(s) to pair up to signal the occurrence of the composite event. An example will help to explain this. Say, that we are interested in detecting the occurrence of the composite event e defined as follows: $e = e_1; e_2$. This means that event e occurs whenever primitive event e_2 occurs after the primitive event e_1 . Suppose $e_1 = \text{Intel stock price drops}$ and $e_2 = \text{IBM stock price drops}$. Assume that the following sequence of primitive events occur: *Intel stock price drops, Intel stock price drops, IBM stock price drops*. In other words event e_1 occurs twice before the occurrence of e_2 . In such cases, which instance(s) of the primitive event e_1 should we consider in the evaluation of the composite event e . Should we consider the first occurrence of e_1 , or the most recent occurrence, or both?

Chakravarthy et al. [4] have solved this problem by formalizing the notion of *context*. They proposed four kinds of contexts, namely, *recent*, *chronicle*, *continuous* and *cumulative*, can be associated with composite events. Recent context requires that only the recent occurrences of primitive events be considered when evaluating the composite event. In the previous example, if recent context were to be used, then the last occurrence of e_1 will be paired up with the occurrence of e_2 and the composite event e will be signaled only once. Chronicle requires that the primitive events be considered in a chronological order when evaluating the composite event. In this context, the composite event will consist of the first occurrence of e_1 followed by the occurrence of e_2 and

will be signaled only once. Continuous requires that the primitive events in a sliding window be considered when evaluating the composite event. In this case, the composite event will be signaled twice. In other words, there will be two occurrence of the composite event. In the first case, the first occurrence of e_1 will be paired with e_2 . In the second case, the next occurrence of e_1 will be composed with e_2 . Cumulative requires that all the primitive events be considered when evaluating the composite event. In this case, only one composite event will be signaled. However, it will take into account both the occurrences of e_1 . Although the four contexts proposed by Chakravarthy et al. [4] can model a wide range of scenarios, they fail to model many situations. For example, consider the following ECA rule that is used in a hospital. *event: admit emergency patient and doctor enters emergency department, condition: true, and action: alert doctor about patients' conditions*. Here *event* is a composite one made up of two primitive events $E1 = \text{admit patient}$ and $E2 = \text{doctor enters emergency room}$. We want this rule to be triggered every time a new patient is admitted and the doctor enters the emergency room. For event $E1$ we want to use the cumulative context and for event $E2$ we want the recent context. Such possibilities cannot be expressed by Chakravarthy's work because the entire composite event is associated with a single context.

We argue that associating a single context with a composite event is often times very restrictive for new applications, such as those executing in the pervasive computing environments. Such applications will have composite events made up of different types of primitive events. Often times, the type of event determines which context should be used. For example, it makes sense to use recent context for events based on streaming data, chronicle context for events involving processing customer orders. Since the constituent primitive events in composite events are of different types, requiring them to be associated with the same context is placing unnecessary restrictions on the composite event and prohibiting them from expressing many real-world situations.

We propose an alternate solution. Instead of associating a single context with a composite event, we associate a context with each constituent primitive event. This allows different types of primitive events to be combined to form a composite event. It also allows us to express many real-world situations, such as the healthcare example stated above. We discuss how this can be done and provide the formal semantics of associating contexts with primitive events for each event composition operator. We prove that our approach is more expressive than Chakravarthy et al. [4]. We give algorithms showing how composite event detection can take place when the primitive events have varying contexts.

The rest of the paper is organized as follows. Section 2 describes few related works in this area. Section 3 illustrates our notion of contexts. Section 4 formally defines our notion of contexts. Section 5 gives algorithms for detecting composite events that use our definition of contexts. Section 6 concludes the paper with pointers to future directions.

2 Related Work

A number of works [1,2,3,4,5,6,7,8] have been done in event specification and detection in active databases. Some active databases detect events using detection-based

semantics [34]; others use interval-based semantics [12]. But not much work appears in the area of parameter contexts.

In COMPOSE [67] and SAMOS [5] systems, the parameters of composite events detected are computed using the unrestricted context. In the unrestricted context, the occurrences of all primitive events are stored and all combinations of primitive events are used to generate composite events. For instance, consider the composite event $E = P; Q$; Let p_1, p_2 be instances of P and q_1, q_2 be instances of Q . Consider the following sequence of events: p_1, p_2, q_1, q_2 . This will result in the generation of four composite events in the unrestricted context: (p_1, q_1) , (p_2, q_1) , (p_1, q_2) , and (p_2, q_2) . Unrestricted contexts have two major drawbacks. The first is that not all occurrences may be meaningful from the viewpoint of an application. The other is the big computation and space overhead associated with the detection of events.

The SNOOP system [1234] discusses an event specification language for active databases. It classifies the primitive events into database, temporal, and explicit events. Composite events are built up from these primitive events using the following five event constructors: *disjunction*, *sequence*, *any*, *aperiodic*, *cumulative aperiodic*, *periodic* and *cumulative periodic* operators. One important contribution of SNOOP is that it proposes the notion of parameter contexts. The parameter context defines which instance of the primitive events must be used in forming a composite event. The authors propose four different parameter contexts which were discussed earlier. Although the SNOOP System discussed how to specify consumption of events in the four different parameter contexts, a parameter context can be specified only at the top-level event expression, which means that the entire composite event is associated with a single context. We argue that associating a single context with a composite event is often times not very meaningful. This is because the type of event often determines which context should be used. Since the constituent primitive events in a composite event are of different types, requiring them to be associated with the same context is placing unnecessary restrictions on the composite event and prohibiting them from expressing many real-world situations.

Zimmer and Unland [9] provides an in-depth discussion of the semantics of complex events. They provide a meta-model for describing various types of complex events. Each event instance belongs to a type. In a complex event, it is possible that many event instances belonging to a particular type occurs. The event instance selection decides which instance to consider in the composite type. They have the operators *first*, *last*, and *cumulative*. Event instance consumption decides whether an event instance can be reused in the composite event or consumed by the composite event. Event instance consumption can be of three types: *shared*, *exclusive*, and *ext-exclusive*. The shared mode does not delete any instance of the event. The exclusive mode deletes only those event instances that were used in triggering the composite event. The ext-exclusive deletes all occurrence of the event. Although the authors provide many different possibilities using the combinations of event instance selection and event instance consumption, their formal semantics are not presented. Moreover, it is hard to understand the impact of these semantics on the implementation. For instance, since shared events are never consumed, it is unclear as to how long they must be stored.

3 Our Model

Time in our model is represented using the totally ordered set of integers. We represent time using temporal intervals and use an interval-based semantics to describe our work. We begin by giving a definition of event occurrence and event detection. The occurrence of an event typically occurs over a time interval. The detection of an event occurs at a particular point in time.

Definition 1. [Occurrence of an event]. *The occurrence of an event E_i is denoted by the predicate $O(E_i, [t_{si}, t_{ei}])$ where $t_{si} \leq t_{ei}$, and t_{si} , t_{ei} denote the start time, end time of E_i respectively. The predicate has the value true when the event E_i has occurred within the time interval $[t_{si}, t_{ei}]$ and is false otherwise. Primitive events are often instantaneous – in such cases the start time t_{si} equals the end time t_{ei} , that is, $t_{si} = t_{ei}$.*

Thus, in our model, events occur over a temporal interval. Since it is not always possible to define a total order on temporal intervals, we propose the notion of overlapping and non-overlapping events.

Definition 2. [Overlap of Events]. *Two events E_i and E_j whose occurrences are denoted by $O(E_i, [t_{si}, t_{ei}])$ and $O(E_j, [t_{sj}, t_{ej}])$ respectively are said to overlap if the following condition is true: $\exists t_p$ such that $(t_{si} \leq t_p \leq t_{ei}) \wedge (t_{sj} \leq t_p \leq t_{ej})$. Otherwise the two events are said to be non-overlapping. The overlap relation is reflexive and symmetric but not transitive.*

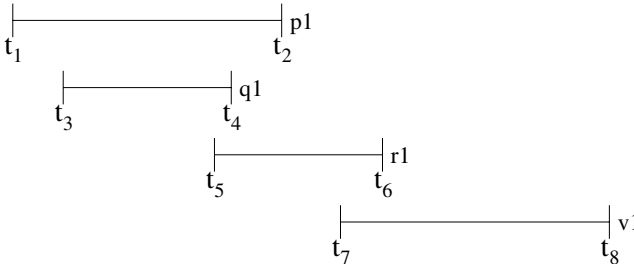


Fig. 1. Example of Event Occurrence

Example 1. Consider the events show in Figure 1. The event $p1$ overlaps with $q1$ and $r1$. $q1$ overlaps with $p1$ and $r1$. $r1$ overlaps with all other events. $v1$ overlaps with $r1$ only. The event $p1$ and $v1$ may correspond to admitting patients *Tom* and *Dick* respectively. The event $q1$ and $r1$ may correspond to alerting the physician and the nurse respectively.

Definition 3. [Detection of an event]. *The detection of an event E_i is denoted by the predicate $D(E_i, t_{di})$ where $t_{si} \leq t_{di} \leq t_{ei}$ and t_{si} , t_{di} , t_{ei} represent the start time, detection time, end time of event E_i respectively. In almost all events (except the composite ones connected by a ternary operator), the detection time is the same as the termination time, that is, $t_{di} = t_{ei}$.*

We assume that the detection time of all events in the system form a total order. This is not an unrealistic assumption since detection of an event takes place at an instant of time.

Definition 4. [Event Ordering]. We define two ordering relations on events. We say an event E_i occurs after event E_j if $t_{di} > t_{dj}$. We say an event E_n follows event E_m if $t_{sn} > t_{em}$, that is event E_n starts after E_m completes. Note that the follows relation can be defined for non-overlapping events but occurs after can be defined for any pair of events.

For the example given in Example 1, $p1, r1, v1$ occurs after $q1$. $v1$ follows $p1$ and $q1$. For composition operators, such as conjunction, occurs after relation is important. For others, such as sequence, follows on relation is more important.

Next, we give our definitions of primitive and composite events.

Definition 5. [Primitive Event]. A primitive event is an atomic event which cannot be decomposed.

In Example 1, $q1$ (alerting the physician) and $r1$ (alerting the nurse) are examples of primitive events. In general, a primitive event can be a database event, a temporal event, or an explicit event. Examples of database primitive event include database operations, such as, select, update, delete, insert, commit, abort, and begin. Example of a temporal event is 06/31/06 midnight. Example of an explicit event is when the sensor reading equals 100 Celsius.

Definition 6. [Composite Event]. A composite event E is an event that is obtained by applying an event composition operator op to a set of constituent events denoted by E_1, E_2, \dots, E_n . This is formally denoted as follows $E = op(E_1, E_2, \dots, E_n)$. The event composition operator op may be binary or ternary. The constituent events E_1, E_2, \dots, E_n may be primitive or composite event.

Consider the composite event $E = P; Q$ where P, Q , and E correspond to admitting patient, alerting physician, and treating the patient respectively. In this case, the sequence operator $;$ is used to compose the primitive events – the physician is alerted after the patient is admitted. A composite event may be associated with different instances of primitive events. Some of these instances have a special significance because they are responsible for starting, terminating or detecting the composite event. These instances, referred to as *initiator*, *terminator* and *detector*, are formally defined below.

Definition 7. [Initiator]. Consider a composite event $E = op(E_1, E_2, \dots, E_n)$ that occurs over the time interval $[t_s, t_e]$. The first detected constituent event instance e_i ($i \in [1, 2, \dots, n]$) that starts the process of parameter computation for this composite event is known as the initiator.

Definition 8. [Detector]. Consider a composite event $E = op(E_1, E_2, \dots, E_n)$ that occurs over the time interval $[t_s, t_e]$. The constituent event instance e_i ($i \in \{1, 2, \dots, n\}$) that detects the occurrence of the composite event E is called the detector.

Definition 9. [Terminator]. Consider a composite event $E = E_1 op_1 E_2 op_2 \dots op_x E_n$ that occurs over a time interval $[t_s, t_e]$. The constituent event instance e_i ($i \in [1, 2, \dots, n]$) that ends in time t_e which terminates the composite event is called the terminator.

Consider the composite event $P;Q$ where P and Q correspond to admitting patient and alerting physician respectively. The instance of event P that starts the parameter computation for the composite event will be called the the initiator. An instance of Q that occurs after the initiator will cause the composite event to be detected. This instance of Q will be called the detector. The same instance of Q also terminates the composite event and hence also acts as the terminator.

In this paper, we focus our attention to binary event composition operators. The same event instance is the *detector* as well as the *terminator* for these binary operators. Hence, in our discussions we will use the terms initiators and terminators only.

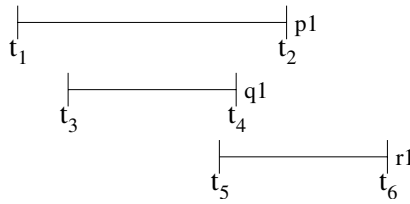


Fig. 2. Example of Initiator and Detector in Example 2

Example 2. Consider the following composite event $E = P \wedge Q \wedge R$. In the hospital example, the events P , Q and R may correspond to admitting the patient, alerting the doctor and alerting the nurse respectively. Let $p1$, $q1$ and $r1$ represent the event instances of P , Q and R respectively. The occurrences of these event instances are shown in Figure 2. The detection and termination occur at the same time instance for all of these events. $q1$ is the initiator and $r1$ is the terminator for event E . Note that, although $p1$ is started before $q1$, $p1$ is not considered to be the initiator. This is because $q1$ is detected before $p1$ and is therefore responsible for starting the parameter computation of the composite event. The occurrence time of E is denoted by the interval $[t_{sp}, t_{er}]$.

The initiator of each event can be associated with different contexts. The contexts specify which instance(s) of the initiator should be paired with a given terminator instance. The terminator context determines whether the same terminator instance can be used with one or more initiators. The formal definition of initiator and terminator contexts are given below.

Initiator in Recent Context. An instance of the initiator event starts the composite event evaluation. Whenever a new instance of the initiator event is detected, it is used for this composite event and the old instance is discarded. An instance of the initiator event is used at most once in the composite event evaluation. After an instance of initiator event has been used for a composite event, it is discarded.

Consider the composite event $T;M$ where T signifies that the patient’s temperature rises above 102 degrees fahrenheit, M denotes administering treatment to reduce the fever. If we are only concerned about the latest temperature of the patient, we would use initiator in recent context.

Initiator in Chronicle Context. Every instance of the initiator event starts a new composite event. The oldest initiator event is used for the composite event. The instance of initiator event is discarded after using it for composite event calculation.

Consider an out-patient check clinic where the patients come for routine check-up. An example composite event is $P;Q$ where P represents the event of patients getting admitted and Q denotes the availability of the health care professional to serve the patient. Since the patients must be served in the order in which they arrive, we will use the initiator in chronicle context.

Initiator in Continuous Context. Every new event instance of the initiator starts a composite event after discarding the previous instance of the initiator. An instance of the initiator event can be used multiple times in the composite event evaluation. The same initiator can be paired with multiple terminators. The initiator is discarded only after another initiator event occurs.

Consider, for example, the composite event $E = P;Q$ where events P , Q and E correspond to doctor entering the emergency room, admitting a patient, and treating the patient respectively. The initiator event corresponds to the doctor entering the emergency room. Since we are interested in the latest occurrence of this event, we can use either recent or continuous context. However, since the same initiator may have to be paired with multiple terminators, we need to use the continuous context.

Initiator in Cumulative Context. The first instance of the initiator event starts the composite event. The subsequent occurrences of the initiator events will all be used in this same composite event. The instances of initiator events are discarded after use.

Consider, for example, the composite event $E = P;Q$ where events P , Q , and E correspond to admitting a patient, doctor entering the emergency room, and notifying the doctor about the condition of the most critical patient. Here, we would like to use the initiator in the cumulative context because the condition of all the patients must be evaluated to find the most critical patient.

Terminator in Continuous Context. Each terminator can be used multiple times in the composite event evaluation. That is, a terminator can be paired with multiple initiators.

Consider, for example, the composite event $E = P;Q$ where events P , Q , and E correspond to admitting a patient, doctor entering the emergency room, and notifying the doctor about the condition of each patient. The doctor entering the emergency room is the terminator event. Since this terminator may have to be paired with multiple initiators corresponding to the different patients admitted, we would need to use continuous context for the terminator.

Terminator in Chronicle Context. Each terminator is used only once in the composite event evaluation. A terminator can be paired with only one initiator. Here the different contexts are not distinguished. This is because the very first occurrence of the terminator will terminate the event.

Consider the composite event $E = P;Q$ where events P , Q , and E correspond to doctor entering the emergency room, admitting a patient, and administering treatment respectively. Since each terminator event of admitting a patient will be used only once, we need to use chronicle context for this.

In this paper, we consider only three binary event composition operators, namely, *disjunction*, *sequence*, and *conjunction*.

Disjunction $E_1 \vee E_2$

This event is triggered whenever an instance of E_1 or E_2 occurs. Since only one event constitutes this composite event, there is no context associated with this single event. This is because the very first instance of E_1 or E_2 will be the initiator as well as the terminator of this composite event.

Sequence $E_1; E_2$

This event is triggered whenever an instance of E_2 follows an instance of E_1 . In this case, an instance of E_1 will be the initiator and an instance of E_2 will be the terminator. Since there may be multiple instances of initiators involved, context determines which instance of E_1 gets paired with which instance of E_2 . To illustrate our ideas, we use an example of composite event $E = A; B$. The events instances are detected as follows: $b1, a1, a2, b2, b3, a3, b4$. The title of the rows is the context of event A , while the title of the columns is the context of event B . We use the following abbreviations to refer to parameter contexts: R – Recent Initiator Context, C – Chronicle Initiator Context, O – Continuous Initiator Context, U – Cumulative Initiator Context, TC – Chronicle Terminator Context and TO – Continuous Terminator Context.

Table 1. Detection of “ $A; B$ ” in Occurrence of “ $b1, a1, a2, b2, b3, a3, b4$ ” in Our Approach

A ; B	TC	TO
R	(a2,b2) (a3,b4)	(a2,b2) (a3,b4)
C	(a1,b2) (a2,b3) (a3,b4)	(a1,b2) (a2,b2) (a3,b4)
O	(a2,b2) (a2,b3) (a3,b4)	(a2,b2) (a2,b3) (a3,b4)
U	(a1,a2,b2) (a3,b4)	(a1,a2,b2) (a3,b4)

Note that not all these cases can be modeled using SNOOP. For instance, the case for continuous initiator and chronicle terminator cannot be expressed by SNOOP. A real-world example rule in an Information Technology Department will motivate the need for this case: *Event E*: A service call comes after an operator enters the on-call status, *Condition C*: true, *Action A*: provide the service. In such cases, the event $E = E_1; E_2$ where $E_1 =$ operator enters on-call status and $E_2 =$ service call comes.

Conjunction $E_1 \wedge E_2$

This event is triggered whenever both instances of E_1 and E_2 occur. For composite event with “and” operator $E(A \wedge B)$, either event A or event B can be an initiator event or a terminator event. When no instance of event B occurs before any instance of event A , event A is considered as an initiator event and event B is considered as a terminator

event. When event B occurs before event A , event B is considered as an initiator event and event A is considered as a terminator event.

Table 2 and Table 3 show the result of “ $A \wedge B$ ” in different combinations of contexts in our approach. The column heading R/TC indicates that when event B is the initiator it uses “Recent Initiator Context” but when it is the terminator it uses “Chronicle Terminator Context”. The other row and column headings are interpreted in a similar manner. Many of the cases shown in Tables 2 and 3 are needed in real-world scenarios and cannot be modeled by SNOOP. For instance, consider the following rule that is used in a hospital: *Event E*: admit emergency patient and doctor enters emergency department, *Condition C*: true, *Action A*: alert doctor about patients’ conditions. Here $E = E_1 \wedge E_2$ is a composite event where $E_1 = \text{admit patient}$ and $E_2 = \text{doctor enters emergency room}$. For event E_1 we want to use Cumulative Initiator/Chronicle Terminator context. That is, when an instance of event E_1 is the initiator we want to use the cumulative context and when it is the terminator we want to use the chronicle context. For event E_2 , we want to use the Recent Initiator/Chronicle Terminator context. Such possibilities cannot be expressed by SNOOP because the entire composite event is associated with a single context.

4 Formal Semantics

In this section, we give the formal definition of each operator when different contexts are associated with each constituent event. For lack of space, we do not provide the formal semantics for the conjunction operator.

Disjunction operator $E1 \vee E2$

For the disjunction operator, the context of the operand is not taken into account for defining the event. This is because the very first event occurrence of either of the events is the initiator as well as the terminator. This operator is commutative.

$$O(E_1 \vee E_2, [t_s, t_e]) = (O(E_1, [t_{s1}, t_{e1}]) \wedge (t_s \leq t_{s1} \leq t_{e1} \leq t_e)) \vee (O(E_2, [t_{s2}, t_{e2}]) \wedge (t_s \leq t_{s2} \leq t_{e2} \leq t_e))$$

Sequence Operator $E1; E2$

Different contexts will result in different semantics for the sequence operator. Since an initiator can be associated with 4 different contexts and a terminator can be associated with 2 contexts, we have 8 different possibilities. Each event occurrence associated with the context is described using the following notation: $O(E1_{Context}; E2_{Context}, [t_{s1}, t_{e2}])$. Their formal definitions are given below:

$$\begin{aligned} &O(E1_R; E2_{TC}), [t_{s1}, t_{e2}] \\ &= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1_R, [t_{s1}, t_{e1}]) \wedge O(E2_{TC}, [t_{s2}, t_{e2}])) \\ &= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1, [t_{s1}, t_{e1}]) \wedge O(E2, [t_{s2}, t_{e2}]) \\ &\quad \wedge (\neg \exists (O(E1', [t'_{s1}, t'_{e1}]) \wedge (t_{s1} \leq t'_{s1} \leq t'_{e1}) \wedge (t_{e1} < t'_{e1} < t_{s2}))) \\ &\quad \wedge (\neg \exists (O(E2', [t'_{s2}, t'_{e2}]) \wedge (t_{e1} < t'_{s2} \leq t'_{e2}) \wedge (t'_{e2} < t_{e2})))) \end{aligned}$$

$$\begin{aligned} &O((E1_R; E2_{TO}), [t_{s1}, t_{e2}]) \\ &\text{The same as } O(E1_R; E2_{TC}), [t_{s1}, t_{e2}] \end{aligned}$$

Table 2. Detection of “ $A \wedge B$ ” in Occurrence of “ $a1, a2, b1, b2, b3, b4, a3, a4$ ” in Our Approach

$A \wedge B$	R/TC	C/TC	O/TC	U/TC
R/TC	(a2,b1) (b4,a3)	(a2,b1) (b2,a3) (b3,a4)	(a2,b1) (b4,a3) (b4,a4)	(a2,b1) (b2,b3,b4,a3)
C/TC	(a1,b1) (a2,b2) (b4,a3)	(a1,b1) (a2,b2) (b3,a3) (b4,a4)	(a1,b1) (a2,b2) (b4,a3) (b4,a4)	(a1,b1) (a2,b2) (b3,b4,a3)
O/TC	(a2,b1) (a2,b2) (a2,b3) (a2,b4)	(a2,b1) (a2,b2) (a2,b3) (a2,b4)	(a2,b1) (a2,b2) (a2,b3) (a2,b4)	(a2,b1) (a2,b2) (a2,b3) (a2,b4)
U/TC	(a1,a2,b1) (b4,a3)	(a1,a2,b1) (b2,a3) (b3,a4)	(a1,a2,b1) (b4,a3) (b4,a4)	(a1,a2,b1) (b2,b3,b4,a3)
R/TO	(a2,b1) (b4,a3)	(a2,b1) (b2,a3) (b3,a3) (b4,a3)	(a2,b1) (b4,a3) (b4,a4)	(a2,b1) (b2,b3,b4,a3)
C/TO	(a1,b1) (a2,b2) (b4,a3)	(a1,b1) (a2,b2) (b3,a3) (b4,a3)	(a1,b1) (a2,b2) (b4,a3) (b4,a4)	(a1,b1) (a2,b2) (b3,b4,a3)
O/TO	(a2,b1) (a2,b2) (a2,b3) (a2,b4)	(a2,b1) (a2,b2) (a2,b3) (a2,b4)	(a2,b1) (a2,b2) (a2,b3) (a2,b4)	(a2,b1) (a2,b2) (a2,b3) (a2,b4)
U/TO	(a1,a2,b1) (b4,a3)	(a1,a2,b1) (b2,a3) (b3,a3) (b4,a3)	(a1,a2,b1) (b4,a3) (b4,a4)	(a1,a2,b1) (b2,b3,b4,a3)

$$O(E1_C; E2_{TC}, [t_{s1}, t_{e2}])$$

$$= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1_C, [t_{s1}, t_{e1}]) \wedge O(E2_{TC}, [t_{s2}, t_{e2}]))$$

$$= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1, [t_{s1}, t_{e1}]) \wedge O(E2, [t_{s2}, t_{e2}])$$

$$\wedge (\neg \exists (O(E1', [t'_{s1}, t'_{e1}]) \wedge (t_{s1} \leq t'_{s1} \leq t'_{e1} \wedge (t'_{e1} < t_{e1}))))$$

$$\wedge (\neg \exists (O(E2', [t'_{s2}, t'_{e2}]) \wedge (t_{e1} < t'_{s2} \leq t'_{e2} \wedge (t'_{e2} < t_{e2}))))))$$

$$O(E1_C; E2_{TO}, [t_{s1}, t_{e2}])$$

$$= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1_C, [t_{s1}, t_{e1}]) \wedge O(E2_{TO}, [t_{s2}, t_{e2}]))$$

$$= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1, [t_{s1}, t_{e1}]) \wedge O(E2, [t_{s2}, t_{e2}])$$

$$\wedge (\neg \exists (O(E2', [t'_{s2}, t'_{e2}]) \wedge (t_{e1} < t'_{s2} \leq t'_{e2} \wedge (t'_{e2} < t_{e2}))))))$$

$$O(E1_O; E2_{TC}, [t_{s1}, t_{e2}])$$

$$= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1_O, [t_{s1}, t_{e1}]) \wedge O(E2_{TC}, [t_{s2}, t_{e2}]))$$

$$= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1, [t_{s1}, t_{e1}]) \wedge O(E2, [t_{s2}, t_{e2}])$$

$$\wedge (\neg \exists (O(E1', [t'_{s1}, t'_{e1}]) \wedge (t_{s1} \leq t'_{s1} \leq t'_{e1} \wedge (t_{e1} < t'_{e1} < t_{s2}))))))$$

Table 3. Detection of “ $A \wedge B$ ” in Occurrence of “ $a1, a2, b1, b2, b3, b4, a3, a4$ ” in Our Approach

A \ B	R/TO	C/TO	O/TO	U/TO
R/TC	(a2,b1) (b4,a3)	(a2,b1) (b1,a3) (b2,a4)	(a2,b1) (b4,a3) (b4,a4)	(a2,b1) (b1,b2,b3,b4,a3)
C/TC	(a1,b1) (a2,b1) (b4,a3)	(a1,b1) (a2,b1) (b1,a3) (b2,a4)	(a1,b1) (a2,b1) (b4,a3) (b4,a4)	(a1,b1) (a2,b1) (b1,b2,b3,b4,a3)
O/TC	(a2,b1) (a2,b2) (a2,b3) (a2,b4) (b4,a3)	(a2,b1) (a2,b2) (a2,b3) (a2,b4) (b4,a3)	(a2,b1) (a2,b2) (a2,b3) (a2,b4) (b4,a3)	(a2,b1) (a2,b2) (a2,b3) (a2,b4) (b4,a3)
U/TC	(a1,a2,b1) (b4,a3)	(a1,a2,b1) (b1,a3) (b2,a4)	(a1,a2,b1) (b4,a3) (b4,a4)	(a1,a2,b1) (b1,b2,b3,b4,a3)
R/TO	(a2,b1) (b4,a3)	(a2,b1) (b1,a3) (b2,a3) (b3,a3) (b4,a3)	(a2,b1) (b4,a3) (b4,a4)	(a2,b1) (b1,b2,b3,b4,a3)
C/TO	(a1,b1) (a2,b1) (b4,a3)	(a1,b1) (a2,b1) (b1,a3) (b2,a3) (b3,a3) (b4,a3)	(a1,b1) (a2,b1) (b4,a3) (b4,a4)	(a1,b1) (a2,b1) (b1,b2,b3,b4,a3)
O/TO	(a2,b1) (a2,b2) (a2,b3) (a2,b4) (b4,a3)	(a2,b1) (a2,b2) (a2,b3) (a2,b4) (b4,a3)	(a2,b1) (a2,b2) (a2,b3) (a2,b4) (b4,a3) (b4,a4)	(a2,b1) (a2,b2) (a2,b3) (a2,b4) (b4,a3)
U/TO	(a1,a2,b1) (b4,a3)	(a1,a2,b1) (b1,a3) (b2,a3) (b3,a3) (b4,a3)	(a1,a2,b1) (b4,a3) (b4,a4)	(a1,a2,b1) (b1,b2,b3,b4,a3)

$$O(E1_O; E2_{TO}, [t_{s1}, t_{e2}])$$

The same as “ $O(E1_O; E2_{TC}, [t_{s1}, t_{e2}])$ ”

$$O(E1_U; E2_{TC}, [t_{s1}, t_{e2}])$$

$$= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1 - U, [t_{s1}, t_{e1}]) \wedge O(E2 - TC, [t_{s2}, t_{e2}]))$$

$$= \exists t_{e1}, t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge (\forall [t'_{s1}, t'_{e1}]) (O(E1, [t'_{s1}, t'_{e1}]) \wedge (t_{s1} \leq t'_{s1} < t'_{e1} \leq t_{e1})))$$

$$\wedge O(E2, [t_{s2}, t_{e2}]) \wedge (\neg \exists (O(E2', [t'_{s2}, t'_{e2}]) \wedge (t_{e1} < t'_{s2} \leq t'_{e2}) \wedge (t'_{e2} < t_{e2}))))$$

$O(E1_U; E2_{TO}, [t_{s1}, t_{e2}])$
 The same as $O(E1_U; E2_{TC}, [t_{s1}, t_{e2}])$.

5 Algorithms for Event Detection

Definition 10. [Event Tree]. An event tree $ET_e=(N, E)$ corresponding to a composite event type E is a directed tree where each node E_i represents an event type. The root node corresponds to event type E , the internal nodes represent the constituent composite event types and the leaf nodes correspond to primitive event types that make up E . The edge (E_i, E_j) signifies that node E_i is a constituent of the composite event E_j . E_i in this case is referred to as the child node and E_j as the parent.

An event can trigger multiple rules. Thus, different event trees can have nodes corresponding to the same event. In such cases, to save storage space, the event trees can be merged to form an event graph. An event graph may contain events belonging to different rules. The nodes corresponding to events which fire one or more rules are labeled d with the rule-ids of the corresponding rule.

Definition 11. [Identical Composite Event Types]. Two composite event types $C = op(E1_{con1}, E2_{con2}, \dots, En_{conn})$ and $C' = op'(E1'_{con1'}, E2'_{con2'}, \dots, En'_{conn'})$ are said to be identical if they satisfy the following conditions:

1. the constituent events and their associated contexts are identical in both the cases, that is, $Ei_{coni} = Ei'_{coni'}$ for $i \in \{1, 2, \dots, n\}$.
2. the constituent events are composed using the same event composition operator, that is, $op = op'$.

Two or more event trees can be merged to form an event graph if they have any common nodes.

Definition 12. [Event Graph]. An event graph $EG=(N, E)$ is a directed graph where node E_i represents an event E_i and edge (E_i, E_j) signifies that the event corresponding to node E_i is a constituent of the composite event corresponding to node E_j . Each node E_i is associated with a label $label_{E_i}$. $label_{E_i}$ is a set of rule-ids (possibly empty) that indicate the rules that will fire when the event corresponding to node $label_{E_i}$ happens.

In the following table, Table et_{e_i} stores the parameters of composite event e_i instance. Table let_{e_j} stores the parameters of left event instance of event e_j . Table ret_{e_j} stores the parameters of right event instance of event e_j .

Algorithm 1

Processing Event e_i for Rules

Input: EG - event graph.(2) e_i - event that has occurred, (3)**ET** - Set of event tables where each table corresponds to an event in EG

Output: **ET** - Set of updated event tables for each event in EG .

Procedure *ProcessEvent*(e_i, \mathbf{ET}, EG)

begin

For each rule-id $r \in label_{e_i}$

 signal the rule and send event parameters to condition evaluator

For all outgoing edges (e_i, e_j) from e_i in EG

begin

 /*propagate parameters in node e_i to the parent node e_j

 and detect composite event */

DetectCompositeEvent($EG, \mathbf{ET}, e_i, param_i, e_j, ct_{\perp}, ct_r$)

end

 Delete row containing parameters of instance e_i from table et_{e_i}

end

The above algorithm *ProcessEvent* describes the actions taken when an event e_i has been detected. Recall that $label_{e_i}$ contains the set of rules that are triggered by e_i . The parameters of e_i are then passed on to the condition evaluator which determines whether the rules listed in $label_{e_i}$ can be triggered or not. The node e_i is marked as occurred indicating that this event has taken place. The parameters of event e_i are then passed onto its parents and the *DetectCompositeEvent* procedure is called. The parameters of event e_i are then removed from the event table et_{e_i} .

Algorithm 2

Detecting Composite Event e_i for Rules

Input: (1) EG – event graph for security level L , (2) \mathbf{ET} – Set of event tables where each table corresponds to an event in EG , (3) e_i – child event that has occurred, (4) $param_i$ – parameters of event e_i , (5) e_j – parent event, (6) ct_{\perp} – context type required for the left child of e_j , and (7) ct_r – context type required for the right child of e_j .

Output: \mathbf{ET} – Set of updated event tables for each event in EG .

Procedure *DetectCompositeEvent*($EG, \mathbf{ET}, e_i, param_i, e_j, ct_{\perp}, ct_r$)

begin

case node e_j

\forall : store $param_i$ in a new row in composite event table et_{e_j}

ProcessEvent(e_j, \mathbf{ET}, EG)

 ; : **if** e_i =left child of e_j

case ct_{\perp}

Recent or Continuous:

if table et_{e_j} is empty

 insert $param_i$ in a new row in table et_{e_j}

else

 Replace existing row in table et_{e_j} with $param_i$

Chronicle:

 store $param_i$ in a new row in table et_{e_j}

Cumulative:

 store $param_i$ in the same row in table et_{e_j}

end case

```

if  $e_i$  =right child of  $e_j$ 
  if left child of  $e_j$  is marked as occurred
    begin
      if (ct_r==Continuous)
        begin
          replace the row with  $param_i$  from table  $ret_{e_j}$ 
          for each row in table  $let_{e_j}$ 
            begin
              copy the rows from  $let_{e_j}$  and  $ret_{e_j}$  into composite table  $et_{e_j}$ 
               $ProcessEvent(e_j, \mathbf{ET}, EG)$ 
            end
          if (ct_l!=Continuous)
            Delete the row from the left event table  $let_{e_j}$ 
          end
        end
      else
        begin
          store  $param_i$  in a row in table  $ret_{e_j}$ 
          copy the first row from  $let_{e_j}$  and the row from  $ret_{e_j}$  into table  $et_{e_j}$ 
          Delete the used row from  $ret_{e_j}$ 
           $ProcessEvent(e_j, \mathbf{ET}, EG)$ 
          if (ct_l!=Continuous)
            Delete the used row from left event table  $let_{e_j}$ 
          end
        end
      end
    end
   $\wedge$ : /* Details omitted */
end case
end

```

6 Conclusion

Many real-world event processing applications often require processing composite events. Composite events are made by applying event composition operator to primitive events. Typically many instances of a primitive event occur before a composite event can be detected. The issue is which primitive event(s) should we consider while computing the composite event. Earlier work on event contexts identified four different types of contexts that can be associated with a primitive event. Associating a single context with a composite event is often not adequate for expressing many real-world scenarios. We have shown how different contexts can be associated with the individual primitive events in a composite event to overcome this situation. We have also provided a formal semantics for event composition and algorithms to enforce it. A lot of work remains to be done. We have focussed only on binary operators for event composition. We would like to extend this work to other operators as well. Although our event composition uses an underlying notion of time, we have not explicitly modeled it. In future, we would like to model time and consider temporal operators as well. We would also like to address how time synchronization can be achieved in a distributed environment.

References

1. Adaikkalavan, R., Chakravarthy, S.: Formalization and Detection of Events Using Interval-Based Semantics. In: Proceedings of the 11th International Conference on Management of Data, Goa, India, January 2005, pp. 58–69 (2005)
2. Adaikkavalan, R., Chakravarthy, S.: SnoopIB: Interval-based Event Specification and Detection for Active Databases. *Data and Knowledge Engineering* 59(1), 139–165 (2006)
3. Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.K.: Composite Events for Active Databases: Semantics, Contexts and Detection. In: Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, September 1994, pp. 606–617. Morgan Kaufmann, San Francisco (1994)
4. Chakravarthy, S., Mishra, D.: Snoop: An Expressive Event Specification Language for Active Databases. *Data and Knowledge Engineering* 14(1), 1–26 (1994)
5. Gatzju, S., Dittrich, K.R.: Detecting Composite Events in Active Database Systems Using Petri Nets. In: Proceedings of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems, Houston, TX, February 1994, pp. 2–9 (1994)
6. Gehani, N., Jagadish, H.V., Shmueli, O.: Event Specification in an Active Object-Oriented Database. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, CA, June 1992, pp. 81–90. ACM Press, New York (1992)
7. Gehani, N., Jagadish, H.V., Shmueli, O.: COMPOSE: A System for Composite Specification and Detection. In: Adam, N.R., Bhargava, B.K. (eds.) *Advanced Database Systems*. LNCS, vol. 759, pp. 3–15. Springer, Heidelberg (1993)
8. Paton, N.W., Diaz, O.: Active Database Systems. *ACM Computing Surveys* 31(1), 63–103 (1999)
9. Zimmer, D., Unland, R.: On the Semantics of Complex Events in Active Database Management Systems. In: Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, pp. 392–399. IEEE Computer Society Press, Los Alamitos (1999)

Reclassification of Linearly Classified Data Using Constraint Databases

Peter Revesz and Thomas Triplet

University of Nebraska - Lincoln, Lincoln NE 68588, USA
revesz@cse.unl.edu, ttriplet@cse.unl.edu

Abstract. In many problems the raw data is already classified according to a variety of features using some linear classification algorithm but needs to be reclassified. We introduce a novel reclassification method that creates new classes by combining in a flexible way the existing classes without requiring access to the raw data. The flexibility is achieved by representing the results of the linear classifications in a linear constraint database and using the full query capabilities of a constraint database system. We implemented this method based on the MLPQ constraint database system. We also tested the method on a data that was already classified using a decision tree algorithm.

1 Introduction

Semantics in data and knowledge bases is tied to classifications of the data. Classifications are usually done by classifiers such as decision trees [7], support vector machines [10], or other machine learning algorithms. After being trained on some sample data, these classifiers can be used to classify even new data.

The reclassification problem is the problem of how to reuse the old classifiers to derive a new classifier. For example, if one has a classifier for disease A and another classifier for disease B, then we may need a classifier for patients who (1) have both diseases, (2) have only disease A, (3) have only disease B, and (4) have neither disease. In general, when combining n classifiers, there are 2^n combinations to consider. Hence many applications would be simplified if we could use a single combined classifier.

There are several natural questions about the semantics of the resultant reclassification. For example, how many of the combination classes are really possible? Even if in theory we can have 2^n combination classes, in practice the number may be much smaller. Another question is to estimate the percent of the patients who fall within each combination class, assuming some statistical distribution on the measured features of the patients.

For example, Figures 1 and 2 present two different ID3 decision tree classifiers for the country of origin and the miles per gallon fuel efficiency of cars. For simplicity, the country of origin is classified as *Europe*, *Japan*, or *USA*, and the fuel efficiency is classified as *low*, *medium*, and *high*. Analysis of such decision trees is difficult. For instance, it is hard to tell by just looking at these decision trees whether there are any cars from Europe with a low fuel efficiency.

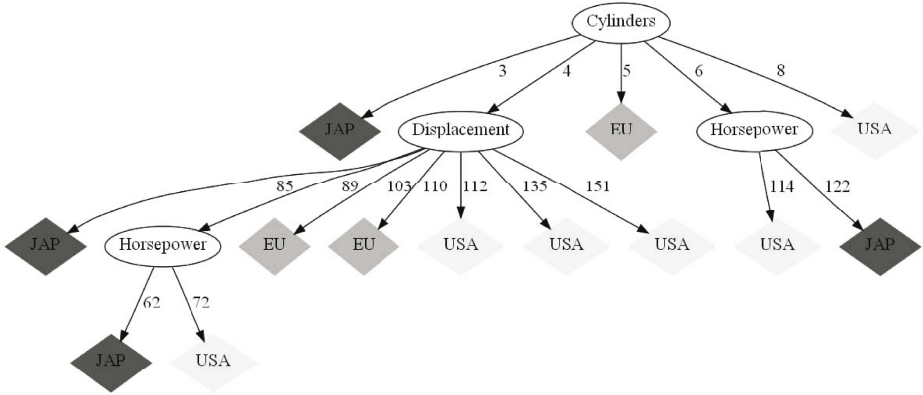


Fig. 1. Decision tree for the country of origin of cars. The tree is drawn using Graphviz [2].

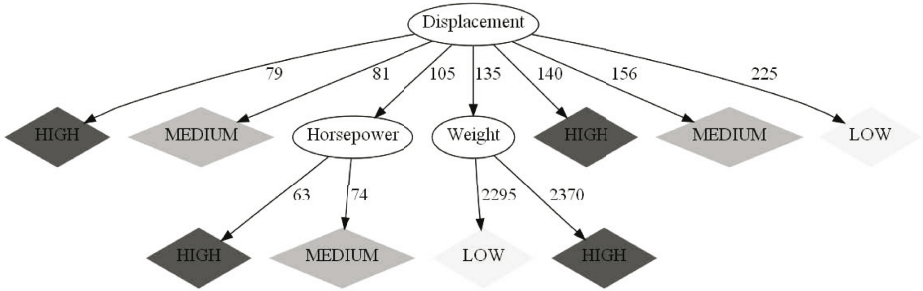


Fig. 2. Decision tree for the miles per gallon fuel efficiency of cars. The tree is drawn using Graphviz [2].

When a decision tree contains all the attributes mentioned in a query, then the decision tree can be used to efficiently answer the query. Here the problem is that the attributes mentioned in the query, that is, country/region of origin and MPG (miles per gallon) fuel efficiency, are not both contained in either decision tree. Reducing this situation to the case of a single decision tree that contains both attributes would provide a convenient solution to the query.

We propose in this paper a novel approach to reclassification that results in a single classifier and enables to answer several semantic questions. Reusability and flexibility are achieved by representing the original classifications in linear constraint databases [6,8] and using constraint database queries.

Background and Contribution. Earlier papers by Geist [4] and Johnson et al. [5] talked about the representation of decision trees in constraint databases. However, they did not consider support vector machines (SVMs) or the reclassification problem. The reclassification problem is a special problem, and its detailed consideration and explanation of the semantic issues regarding reclassifications

are important contributions of this paper. In particular, we point out that the reclassification problem is solved nicely with the use of constraint databases. Furthermore, our experiments compare several different possible approaches to the reclassification problem. The experiments support the idea that the constraint database approach to reclassification is an accurate and flexible method.

The rest of the paper is organized as follows. Section 2 presents a review of linear classifiers. Section 3 presents the reclassification problem. Section 4 describes two novel approaches to the reclassification problem. The first approach is called *Reclassification with an oracle*, and the second approach is called *Reclassification with constraint databases*. Section 5 describes some computer experiments and discusses their significance. Finally, Section 6 gives some concluding remarks and open problems.

2 Review of Linear Classifiers

The problem is the following: we want to classify items, which means we want to predict a characteristic of an item based on several parameters of the item. Each parameter is represented by a variable which can take a finite number of values. The set of those variables is called *feature space*. The actual characteristic of the item we want to predict is called the *label* or *class* of the item. To make the predictions, we use a machine learning technique called *classifier*. A classifier maps a feature space X to a set of labels Y . A linear classifier maps X to Y by a linear function.

Example 1. Suppose that a disease is conditioned by two antibodies A and B. The feature space X is $X = \{Antibody_A, Antibody_B\}$ and the set of labels is $Y = \{Disease, no_Disease\}$. Then, a linear classifier is:

$$y = w_1.Antibody_A + w_2.Antibody_B + c$$

where $w_1, w_2 \in \mathcal{R}$ are constant weights and $c \in \mathcal{R}$ is a threshold constant. The $y \in Y$ value can be compared with zero to yield a classifier. That is,

- If $y \leq 0$ then the patient has *no_Disease*.
- If $y > 0$ then the patient has *Disease*.

In general, assuming that each parameter can be assigned a numerical value x_i , a linear classifier is a linear combination of the parameters:

$$y = f\left(\sum_j w_j x_j\right) \quad (1)$$

where f is a linear function. $w_i \in \mathcal{R}$ are the weights of the classifiers and entirely define it. $y \in Y$ is the predicted label of the instance.

Decision Trees: the ID3 Algorithm. Decision trees, also called *active classifier* were particularly used in the nineties by artificial intelligence experts. The main reasons are that they can be easily implemented (using ID3 for instance) and that they give an *explanation* of the result.

Algorithmically speaking, a decision tree is a tree:

- An internal node tests an attribute,
- A branch corresponds to the value of the attribute,
- A leaf assigns a classification.

The output of decision trees is a set of logical rules (disjunction of conjunctions). To train the decision tree, we can use the ID3 algorithm, proposed by J.R. Quinlan *et al.* [7] in 1979 in the following three steps:

1. First, the best attribute, A , is chosen for the next node. The best attribute maximizes the information gain.
2. Then, we create a descendant for each possible value of the attribute A .
3. This procedure is eventually applied to non-perfectly classified children.

This best attribute is the one, which maximizes the information gain. The information gain is defined as follows:

$$Gain(S, A) = entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} \cdot entropy(S_v)$$

S is a sample of the training examples and A is a partition of the parameters. Like in thermodynamics, the entropy measures the *impurity* of S , purer subsets having a lower entropy:

$$entropy(S) = - \sum_{i=0}^n p_i \cdot \log_2(p_i)$$

S is a sample of the training examples, p_i is the proportion of i -valued examples in S and n is the number of attributes.

ID3 is a greedy algorithm, without backtracking. This means that this algorithm is sensible to local optima. Furthermore, ID3 is inductively biased: the algorithm favors short trees and high information gain attributes near the root. At the end of the procedure, the decision tree perfectly suits the training data including noisy data. This leads to complex trees, which usually lead problems to generalize new data (classify unseen data). According to Occams razor, shortest explanations should be preferred. In order to avoid over-fitting, decision trees are therefore usually pruned after the training stage by minimizing $size(tree) + error_rate$, with $size(tree)$ the number is leaves in the tree and $error_ate$ the ratio of the number of misclassified instances by the total number of instances (also equal to $1 - accuracy$).

Note. The ID3 decision tree and the support vector machine are linear classifiers because their effects can be represented mathematically in the form of Equation (II).

3 The Reclassification Problem

The need for reclassification arises in many situations. Consider the following.

Example 2. One study found a classifier for the origin of cars using

$$X_1 = \{acceleration, cylinders, displacement, horsepower\}$$

and

$$Y_1 = \{Europe, Japan, USA\}$$

where acceleration from 0 to 60 mph is measured in seconds (between 8 and 24.8 seconds), cylinders the number of cylinders of the engine (between 3 and 8 cylinders), displacement in cubic inches (between 68 and 455 cubic inches) and horsepower the standard measure of the power of the engine (between 46 and 230 horsepower).

A sample training data is shown in the table below.

Origin

Acceleration	Cylinders	Displacement	Horsepower	Country
12	4	304	150	USA
9	3	454	220	Europe

Another study found another classifier for the fuel efficiency of cars using

$$X_2 = \{acceleration, displacement, horsepower, weight\}$$

and

$$Y_2 = \{low, medium, high\}$$

where the weight of the car is measured in pounds (between 732 and 5140 lbs.).

A sample training data for the second study is shown in the table below.

Efficiency

Acceleration	Displacement	Horsepower	Weight	MPG
20	120	87	2634	medium
15	130	97	2234	high

Suppose we need to find a classifier for

$$X = X_1 \cup X_2 = \{acceleration, cylinders, displacement, horsepower, weight\}$$

and

$$Y = Y_1 \times Y_2 = \{Europe - low, Europe - medium, Europe - high, \\ Japan - low, Japan - medium, Japan - high, \\ USA - low, USA - medium, USA - high\}$$

Building a new classifier for (X, Y) seems easy, but the problem is that there is no database for (X, Y) . Finding such a database would require a new study with more data collection, which would take a considerable time. That motivates the need for reclassification. As Section 4.1 shows, a classifier for (X, Y) can be built by an efficient reclassification algorithm that uses only the already existing classifiers for (X_1, Y_1) and (X_2, Y_2) .

4 Novel Reclassification Methods

We introduce now several new reclassification methods. Section 4.1 describes two variants of the *Reclassification with an oracle* method. While oracle-based methods do not exist in practice, these methods give a limit to the best possible practical methods. Section 4.2 describes the practical *Reclassification with constraint databases* method. A comparison of these two methods is given later in Section 5.

4.1 Reclassification with an Oracle

In theoretical computer science, researchers study the computational complexity of algorithms in the presence of an oracle that tells some extra information that can be used by the algorithm. The computational complexity results derived this way can be useful in establishing theoretical limits to the computational complexity of the studied algorithms.

Similarly, in this section we study the reclassification problem with a special type of oracle. The oracle we allow can tell the value of a missing attribute of each record. This allows us to derive essentially a theoretical upper bound on the best reclassification that can be achieved. The reclassification with oracle method extends each of the original relations with the attributes that occur only in the other relation. Then one can take a union of the extended relations and apply any of the classification algorithms one chooses. We illustrate the idea behind the *Reclassification with an oracle* using an extension of Example 2 and ID3.

Example 3. First, we add a weight and an MPG attribute to each record in the *Origin* relation using an oracle. Suppose we get the following:

Origin

Acceleration	Cylinders	Displacement	Horsepower	Weight	Country	MPG
12	4	304	150	4354	USA	low
9	3	454	220	3086	Japan	medium

Second, we add a cylinders and a country attribute to each record in the *Efficiency* relation using an oracle. Suppose we get the following:

Efficiency

Acceleration	Cylinders	Displacement	Horsepower	Weight	Country	MPG
20	6	120	87	2634	USA	medium
15	4	130	97	2234	Europe	high

After the union of these two relations, we can train an ID3 decision tree to yield a reclassification as required in Example 2.

A slight variation of the *Reclassification with an oracle* method is the *Reclassification with an X-oracle*. That means that we only use the oracle to extend the original relations with the missing X attributes. For example, in the car example, we use the oracle to extend the *Origin* relation by only *weight*, and the *Efficiency* relation by only *cylinders*.

When we do that, then the original classification for *MPG* (derived from the second study) can be applied to the records in the extended *Origin* relation. Note that this avoids using an oracle to fill in the *MPG* values, which is a Y or target value. Similarly, the original classification for *country* (derived from the first study) can be applied to the records in the extended *Efficiency* relation.

The *Reclassification with an X-oracle* also is not a practical method except if the two original studies have exactly the same set of X attributes because oracles do not exist and therefore can not be used in practice.

4.2 Reclassification with Constraint Databases

The *Reclassification with Constraint Databases* method has two main steps:

Translation to Constraint Relations. We translate the original linear classifiers to a constraint database representation. Our method does not depend on any particular linear classification method. It can be an ID3 decision tree method [7] or a support vector machine classification [10] or some other linear classification method.

Join. The linear constraint relations are joined together using a constraint join operator [68].

Example 4. Figure 1, which we saw earlier, shows an ID3 decision tree for the country of origin of the cars obtained after training by 50 random samples from a cars database [1]. A straightforward translation from the original decision tree to a linear constraint database does not yield a good result for problems where the attributes can have real number values instead of only discrete values. Real number values are often used when we measure some attribute like weight in pounds or volume in centiliters.

Hence we improve the naive translation by introducing comparison constraints $>$, $<$, \geq , \leq to allow continuous values for some attributes.

That is, we translate each node of the decision tree by analyzing all of its children. First, the children of each node are sorted based on the possible values of the attribute. Then, we define an interval around each discrete value based on the values of the previous and the following children. The lower bound of the interval is defined as the median value between the value of the current child and the value of the previous child. Similarly, the upper bound of the interval is defined as the median value of the current and the following children. For instance, assume we have the values $\{10, 14, 20\}$ for an attribute for the children. This will lead to the intervals $\{(-\infty, 12], (12, 17], (17, +\infty)\}$.

In the following, let a be the acceleration, c the number of cylinders, d the displacement of the engine, h the horsepower, w the weight of the car, $country$ the origin of the car, and mpg the miles per gallon of the car. We use the depth-first algorithm with the above heuristic on the cars data from [1] to generate the following MLPQ [9] constraint database:

```
Origin(a,c,d,h,country) :- c = 3, country = 'JAPAN'.
Origin(a,c,d,h,country) :- c = 4, d > 111, country = 'USA'.
Origin(a,c,d,h,country) :- c = 4, d > 96, d <= 111, country = 'EUROPE'.
Origin(a,c,d,h,country) :- c = 4, d > 87, d <= 96, h > 67, country = 'USA'.
                                :
```

Similarly, we used another decision tree to classify the efficiency of the cars. Translating the second decision tree yielded the following constraint relation:

```
Efficiency(a,d,h,w,mpg) :- d <= 103, mpg = 'low'.
Efficiency(a,d,h,w,mpg) :- d > 103, d <= 112, h < 68, mpg = 'high'.
Efficiency(a,d,h,w,mpg) :- d > 420, mpg = 'low'.
                                :
```

Now the reclassification problem can be solved by a constraint database join of the *Origin* and *Efficiency* relations. The join is expressed by the following Datalog query:

```
Car(a,c,d,h,w,country,mpg) :- Origin(a,c,d,h,country),
                                Efficiency(a,d,h,w,mpg).
```

The reclassification can be used to predict for any particular car its country of origin and fuel efficiency. For example, if we have a car with $a = 19.5$, $c = 4$, $d = 120$, $h = 87$, and $w = 2979$, then we can use the following Datalog query:

```
Predict(country,mpg) :- Car(a,c,d,h,w,country,mpg),
                        a = 19.5, c = 4, d = 120, h = 87, w = 2979.
```

The prediction for this car is that it is from Europe and has a low fuel efficiency. Note that instead of Datalog queries, in the MLPQ constraint database system one also can use logically equivalent SQL queries to express the above problems.

Semantic Analysis of the Reclassification. Returning to the semantics questions raised in the introduction, one can test each constraint row of the *Cars* relation whether it is satisfiable or not. That allows the testing of which combination classes (out of the nine target labels) is possible. Moreover, the size of each region

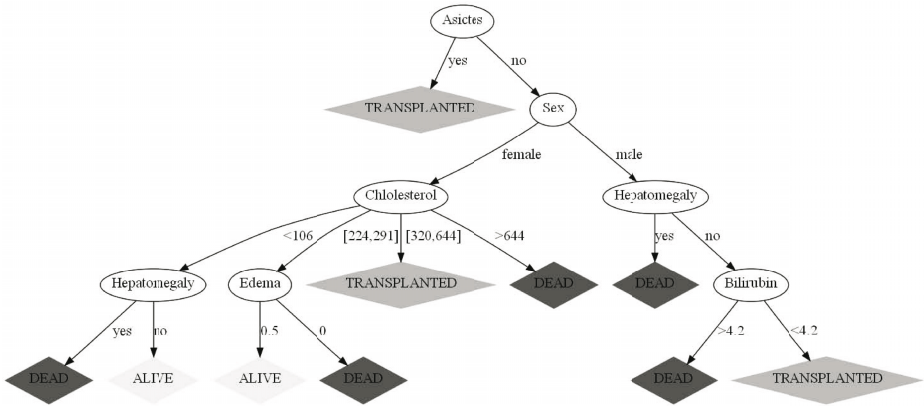


Fig. 3. Tree representation of the constraints using for the prediction of the status of a patient using PBC data. Note that this tree is different from the decision tree generated using the standard ID3 algorithm (here the values of the attributes are defined using constraints). The tree is drawn using Graphviz [2].

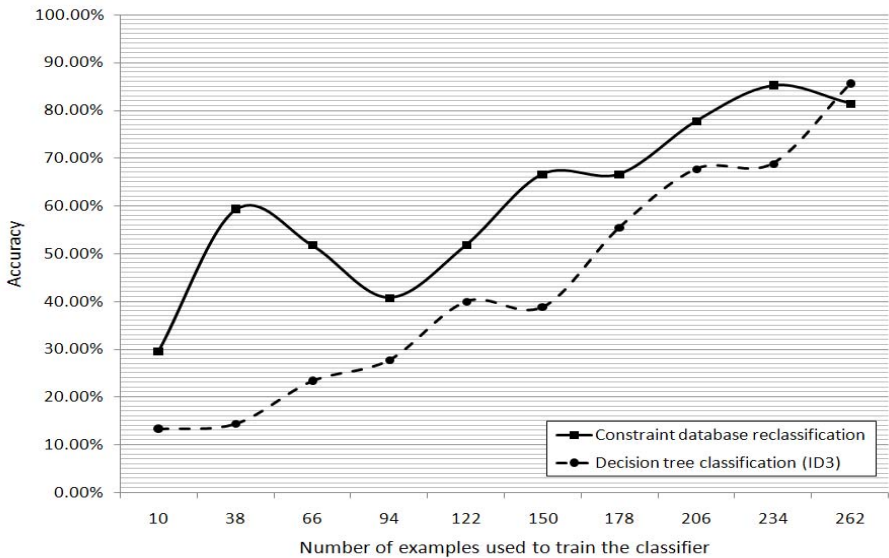


Fig. 4. Comparison of the *Reclassification with constraint databases* (solid line) and the original *Classification with a decision tree (ID3)* for the prediction of the class DISEASE-STAGE of the patients (dashed line) methods using PBC data

can be calculated. Assuming some simple distributions of the cars within the feature space, one can estimate the number of cars in each region. Hence one also can estimate the percent of cars that belong to each combination class.

5 Experiments and Discussion

The goal of our experiments is to compare the *Reclassification with constraint databases* and the *Reclassification with an oracle* methods. It is important to make the experiments such that those abstract away from the side issue of which exact classification algorithm (ID3, SVM etc.) is used for the original classification.

In our experiments we use the ID3 method as described in Example 4. Therefore, we compared the *Reclassification with constraint databases* assuming that ID3 was the original classification method with the *Reclassification with an oracle* assuming that the same ID3 method was used within it. We also compared *Reclassification with constraint databases* with the original linear classification (decision tree) for each class. We chose the ID3 decision tree method for the experiments because it had a non-copyrighted software. Using ID3 already gave interesting results that helped compare the relative accuracy of the methods. Likely a more complex decision tree method would make the accuracy of all the practical algorithms, including the reclassification methods, proportionally better without changing their relative order.

5.1 Experiment with the Dataset “Primary Biliary Cirrhosis”

The *Primary Biliary Cirrhosis (PBC)* data set, collected between 1974 and 1984 by the Mayo Clinic about 314 patients [3], contains the following features:

1. case number,
2. days between registration and the earliest of death, transplantation, or study analysis time,
3. age in days,
4. sex (0=male, 1=female),
5. ascites present (0=no or 1=yes),
6. hepatomegaly present (0=no or 1=yes),
7. spiders present (0=no or 1=yes),
8. edema (0 = no edema, 0.5 = edema resolved with/without diuretics, 1 = edema despite diuretics),
9. serum bilirubin in mg/dl,
10. serum cholesterol in mg/dl,
11. albumin in mg/dl,
12. urine copper in $\mu\text{g}/\text{day}$,
13. alkaline phosphatase in Units/liter,
14. SGOT in Units/ml,
15. triglycerides in mg/dl,
16. platelets per cubic ml/1000,
17. prothrombin time in seconds,
18. status (0=alive, 1=transplanted, or 2=dead),
19. drug (1=D-penicillamine or 2=placebo), and
20. histologic stage of disease (1, 2, 3, 4).

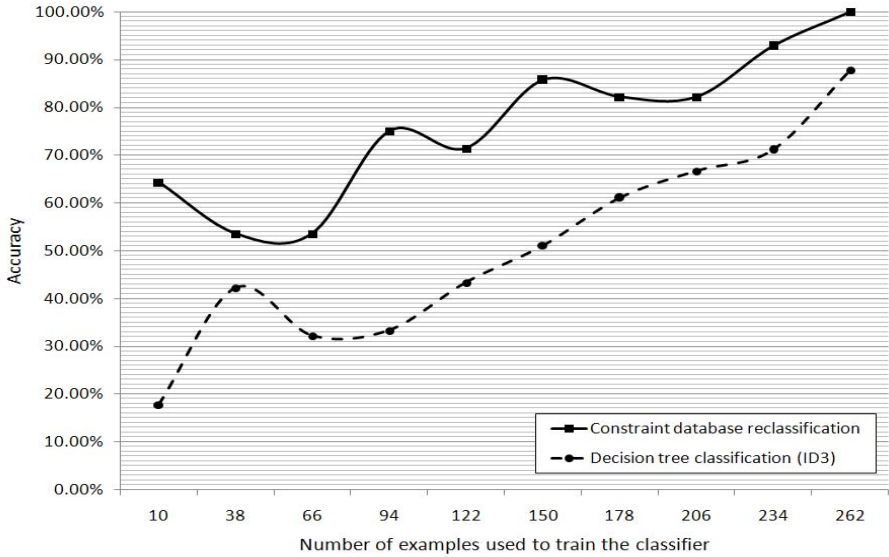


Fig. 5. Comparison of the *Reclassification with constraint databases* (solid line) and the original *Classification with a decision tree (ID3)* for the prediction of the class STATUS of the patients (dashed line) methods using PBC data

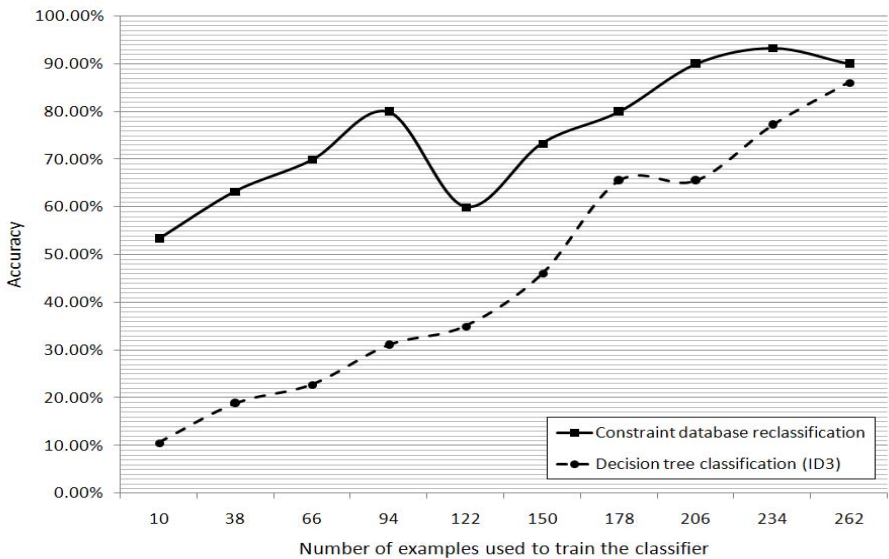


Fig. 6. Comparison of the *Reclassification with constraint databases* (solid line) and the original *Classification with a decision tree (ID3)* for the prediction of the class DRUG of the cars (dashed line) methods using PBC data

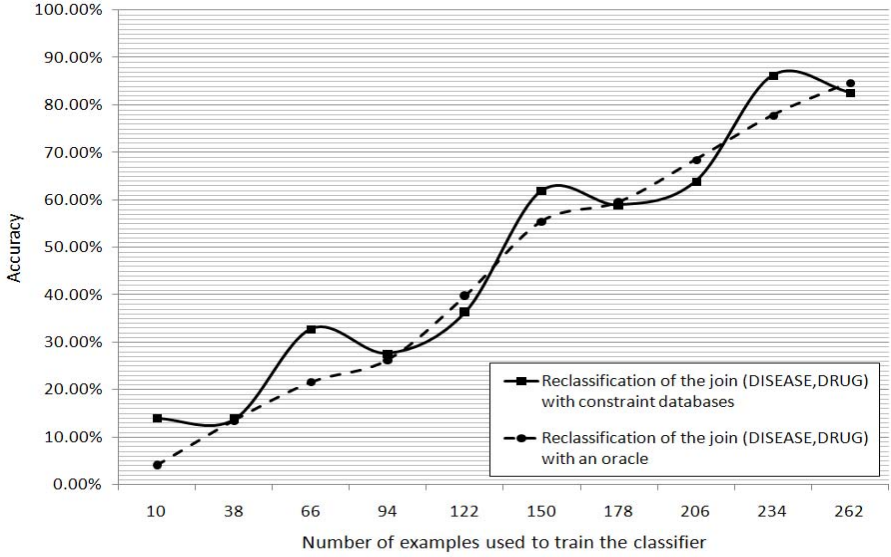


Fig. 7. Comparison of the *Reclassification with constraint databases* (solid line) and the *Reclassification with an oracle* (dashed line) methods using PBC data

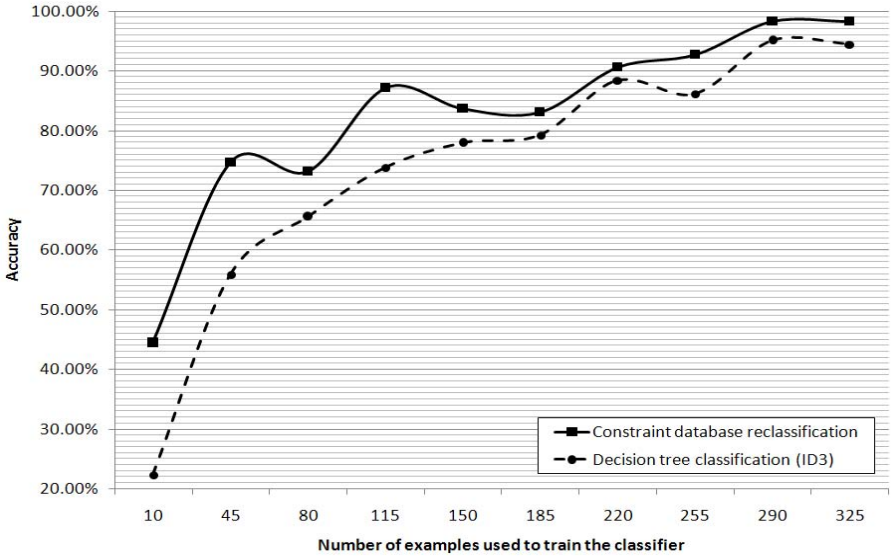


Fig. 8. Comparison of the *Reclassification with constraint databases* (solid line) and the original *Classification with a decision tree (ID3)* for the prediction of the class ORIGIN of the cars (dashed line) using cars data

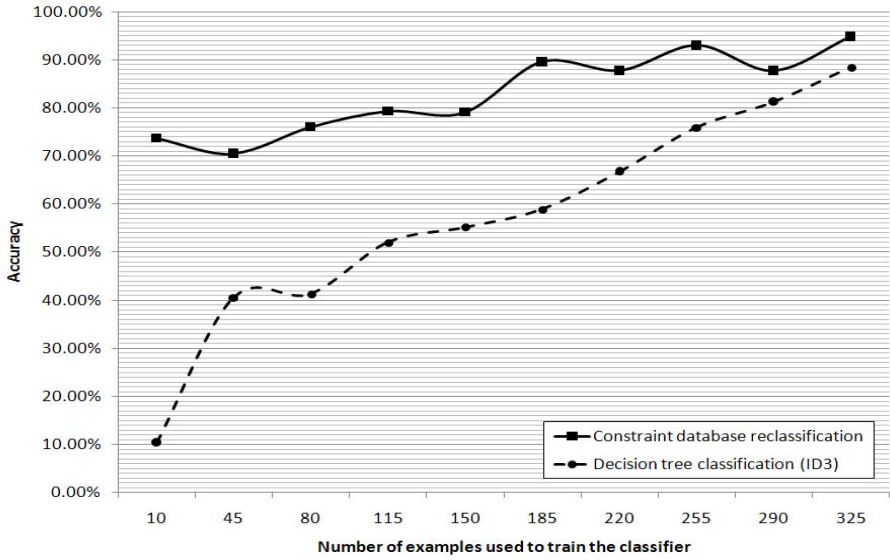


Fig. 9. Comparison of the *Reclassification with constraint databases* (solid line) and the original *Classification with a decision tree (ID3)* for the prediction of the class MPG efficiency of the cars (dashed line) using cars data

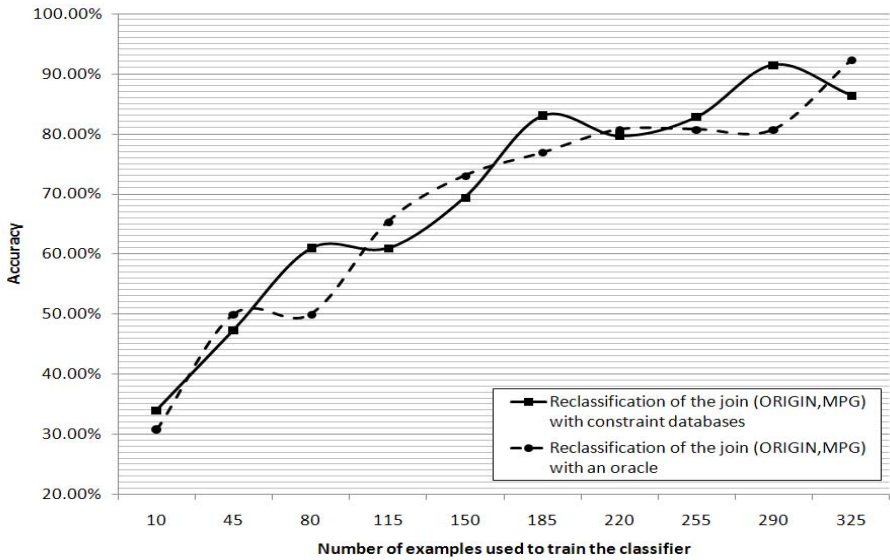


Fig. 10. Comparison of the *Reclassification with constraint databases* (solid line) and the *Reclassification with an oracle* (dashed line) using cars data

We generated the following three subsets from the original data set: DISEASE with features (3, 4, 5, 7, 8, 9, 10, 13, 14, 16, 17, 20), DRUG with features (3, 4, 6, 7, 8, 9, 10, 11, 13, 16, 17, 19), and STATUS with features (3, 4, 5, 6, 7, 8, 9, 10, 11, 16, 17, 18).

In each subset, we used the first eleven features to predict the twelfth, that is, the last feature.

The results of the experiment are shown in Figures 4, 5, 6 and 7.

It can be seen from Figures 4, 5 and 6 that the accuracy of the *Reclassification with constraint databases* has significantly improved compared to the original linear classification (ID3) of a single class.

Figure 7 shows that the *Reclassification with constraint databases* and the *Reclassification with an oracle* perform very similarly. Hence the practical *Reclassification with constraint databases* method achieves what can be considered as the theoretical limit represented by the *Reclassification with an oracle* method. Note that by theoretical limit we mean only a maximum achievable with the use of the ID3 linear classification algorithm. Presumably, if we use for example support vector machines, then both methods will improve proportionally.

5.2 Experiment with the Dataset “Cars”

In this experiment we used the car data set (pieces of which we used in the examples of this paper) from [1] and the MLPQ constraint database system [9]. The results of the experiment are shown in Figures 8, 9 and 10.

This second experiment agrees with the first data set in that constraint database-based reclassification performs better than the original linear decision tree-based classification.

6 Conclusions

The most important conclusion that can be drawn from the study and the experiments is that the *Reclassification with constraint databases* method improves the accuracy of linear classifier such as decision trees. The proposed method is also close to the theoretical optimal when joining two classes and is safe to use in practice.

There are several open problems. We plan to experiment with other data sets and use the linear Support Vector Machine algorithm in addition to the ID3 algorithm in the future. Also, when an appropriate data can be found, we also would like to test the *Reclassification method with X-oracles*.

Acknowledgement. The work of the first author of this paper was supported in part by a Fulbright Senior U.S. Scholarship from the Fulbright Foundation-Greece.

References

1. Donoho, D., Ramos, E.: The CRCARS dataset. Exposition of Statistical Graphics Technology, Toronto (1983)
2. Ellson, J., Gansner, E., Koutsofios, E., North, S., Woodhull, G.: Graphviz and dynagraph – static and dynamic graph drawing tools. In: Junger, M., Mutzel, P. (eds.) Graph Drawing Software, pp. 127–148. Springer, Heidelberg (2003)
3. Fleming, T.R., Harrington, D.P.: Counting Processes and Survival Analysis. Wiley, New York (1991)
4. Geist, I.: A framework for data mining and KDD. In: Proc. ACM Symposium on Applied Computing, pp. 508–513. ACM Press, New York (2002)
5. Johnson, T., Lakshmanan, L.V., Ng, R.T.: The 3W model and algebra for unified data mining. In: Proc. IEEE International Conference on Very Large Databases, pp. 21–32 (2000)
6. Kuper, G.M., Libkin, L., Paredaens, J. (eds.): Constraint Databases. Springer, Heidelberg (2000)
7. Quinlan, J.: Induction of decision trees. *Machine Learning* 1(1), 81–106 (1986)
8. Revesz, P.: Introduction to Constraint Databases. Springer, Heidelberg (2002)
9. Revesz, P., Chen, R., Kanjamala, P., Li, Y., Liu, Y., Wang, Y.: The MLPQ/GIS constraint database system. In: Proc. ACM SIGMOD International Conference on Management of Data (2000)
10. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, Heidelberg (1995)

Evaluating Performance and Quality of XML-Based Similarity Joins

Leonardo Ribeiro and Theo Härder

AG DBIS, Department of Computer Science,
University of Kaiserslautern, Germany
{ribeiro,haerder}@informatik.uni-kl.de

Abstract. A similarity join correlating fragments in XML documents, which are similar in structure and content, can be used as the core algorithm to support data cleaning and data integration tasks. For this reason, built-in support for such an operator in an XML database management system (XDBMS) is very attractive. However, similarity assessment is especially difficult on XML datasets, because structure, besides textual information, may embody variations in XML documents representing the same real-world entity. Moreover, the similarity computation is considerably more expensive for tree-structured objects and should, therefore, be a prime optimization candidate. In this paper, we explore and optimize tree-based similarity joins and analyze their performance and accuracy when embedded in native XDBMSs.

1 Introduction

Data cleaning deals with the identification and correction of data inconsistencies. Frequently, due to such inconsistencies (e.g., mis-spellings), multiple representations of real-world objects appear in data collections. Such redundancy may lead to wrong results, confuse consistency maintenance, and, when integrated from various data sources, may artificially inflate data files. Therefore, detection of duplicates by correlation of (string) attribute values is a long-term research goal in the relational world [5,6,4,2,1], often denoted as the *fuzzy duplicate problem*.

As a classical solution, relational DBMSs have correlated records by using *similarity joins*. A similarity join pairs tuples from two relations whose specified attribute values are similar. These values can be composed by a single column or by the concatenation of column sequences (e.g., $R[Name]$ and $R[Name, Address]$). Using a *similarity function*, a pair of tuples is qualified if a similarity value greater than a given threshold is returned.

Over the recent years, XML is increasingly used as standard for information representation, in addition to provide a common syntax for data exchange. In this context, similarity operators must deal with tree-structured documents instead of table-structured objects. However, extending such operations to XML brings a new quality dimension to the correlation problem. Because similar or even the same information could be embodied by quite different structures or fragments in XML documents, textual techniques developed for relational data are not

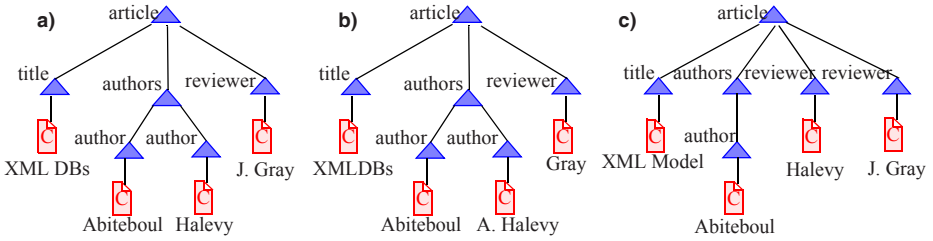


Fig. 1. Sample XML document fragments

sufficient, as disclosed by the example in Fig. 1. Consider a situation where tree *a*) has to be correlated to tree *b*). Although they are obviously identical (for human observers), they would not be classified as equal because of variations in the XML *content part*. However, the use of an appropriate textual similarity predicate would easily classify them as matching candidates. On the other hand, although *c*) refers to another article, the comparison of *a*) with *c*) using the same similarity predicate would probably lead to an erroneous classification, i.e., to a match, when only textual similarity is considered. In this case, the information needed to lower the overall similarity is conveyed in the XML *structure part*. Due to the increased modeling flexibility of XML, e.g., by optional elements and attributes, even data sources sharing a same DTD may not have an identical tree structure. Therefore, accurate and efficient DBMS methods to correlate XML data need to cope with additional complexities induced by the XML structure.

Our Contribution. We propose built-in support for similarity joins in an XDBMS query engine and design a foundational framework for approximation operators based on three main conceptual components: system embedding, candidate pair generation, and quality measures. System embedding means subtree access using index-based location of qualified XML fragments. We complement this component with the support of various types of predicates to select specific parts of a subtree for similarity evaluation. Using techniques from the relational world such as signature schemes and equi-joins, we facilitate candidate pair generation. For quality measures, we concentrate on set-overlap-based similarity functions. In this setting, we introduce a novel mechanism, *extended pq-grams*, to derive tokens combining structure and content of XML tree structures. For this new concept, we propose three versions of token generation which jointly use textual and structural information. Furthermore, we explore a tokenization scheme called *path-gram*. Our solution has important advantages over previous work on XML data: it unifies string- and tree-based techniques in a single processing and scoring model thereby providing efficiency and improving the overall matching quality; the resulting operation can be combined with regular XML queries (e.g., by delivering the resulting nodes in document order) to compose processing logic which enables more complex data cleaning solutions. Finally, we give detailed quantitative results by conducting empirical experiments and performance measurements using our prototype XDBMS called XTC [10].

2 Related Work

Chaudhuri et al. [4] introduced the idea of extending the set of physical operators to provide support for similarity joins inside database engines. A core component of this approach uses signature schemes, which is used to avoid unnecessary comparisons [2]. Alternatively, similarity joins can be specified by using languages such as SQL [5,6,11]. Such a specification offers high flexibility and enables system-controlled optimization as key advantages. However, the related approaches have some serious performance limitations. First, they require considerable effort in a pre-processing stage where a number of auxiliary tables have to be constructed, e.g., tables to store tokens and cardinalities. Second, the set of candidates consists of all pair of tuples that share at least one token. Even by using well-known techniques such as stop-word removal, the set of candidates can potentially become very large and can be frequently populated by tuples that do not make it to the final result. A solution proposed for this problem uses sampling methods to one or both of the join partners [5], and thus are compromising the *exactness* of the result, i.e., they may miss some valid results.

A very large body of work is available on textual similarity [4,13] for which we only mention the well-known concept of *edit distance* [11]. More important for our work is the *tree edit distance* which—using tree edit operations such as node insertion, node deletion, and node renaming—is defined as the cost-minimal operation sequence transforming a tree into the one to be compared [15]. Unfortunately, all algorithmic results have more than $O(n^2)$ runtime complexity and are, therefore, impractical for XDBMS use with potentially large trees.

Close to our idea, Guha et al. [7] present a framework for XML similarity joins based on tree edit distance. To improve scalability, they optimized and limited distance computations by using lower and upper bounds for the tree edit distance as filters and a pivot-based approach for partitioning the metric space. However, these computations are still in $O(n^2)$ and heavily depend on a good choice of parameters (reference set). Avoiding this cost/scalability penalty and manual parameter choices, applying *pq*-grams [3] to derive an efficient tree-based distance approximation seems to be more interesting for XDBMS use. Moreover, the use of structural information enabled by a generalization of *q*-grams allows leveraging a large body of similarity join techniques addressing performance enhancements (e.g., *signature schemes* [2,4] and pipelined evaluation [4]) as well as versatile applications of similarity functions (e.g., set-overlap-based similarity methods [4,13]). Moreover, in contrast to [7], this framework easily deals with modifications of the underlying tree structures.

3 Concepts Used for Similarity Join Computation

An XML document is modeled as an ordered labeled tree. We distinguish between element nodes and text nodes, but not between element nodes and attribute nodes; each attribute is child of its owning element. Disregarding other node types such as Comment, we consider only data of string type.

3.1 Similarity Joins on XML Collections

A general tree similarity join takes as input two collections of XML documents (or document fragments) and outputs a sequence of all pairs of trees from the two collections that have similarity greater than a given threshold. The notion of similarity between trees is numerically assessed by a similarity function used as join predicate and applied on the specified node subsets of the respective trees.

Definition 1 (General Tree Similarity Join). *Let F_1 and F_2 be two forests of XML trees. Given two trees T_1 and T_2 , we denote by $\text{sim}(T_1, T_2)$ a similarity function on node sets of T_1 and T_2 , respectively. Finally, let γ be a constant threshold. A tree similarity join between F_1 and F_2 returns all pairs $(T_1, T_2) \in F_1 \times F_2$ such that $\text{sim}(T_1, T_2) \geq \gamma$.*

Note that the similarity function is applied to node sets instead to trees. When comparing trees, we need the flexibility to evaluate their similarity using node subsets that do not have containment relationships among them, e.g., node sets only consisting of text nodes. If structure matters, the *node labeling scheme* allows identifying containment relationships among a set of nodes.

3.2 Set-Overlap-Based Similarity Measures

Given two sets representing two objects, different ways to measure their overlap raise various notions of similarity (or dissimilarity). There are several proposals for such measures, among others the *Jaccard similarity*, binary cosine similarity, and the Hamming distance. We observed that the method used to map an object to a set also has influence on the notion of similarity, because it determines which properties of the object are under consideration by the similarity measure. For example, given an XML tree, we can produce sets representing its textual information or its structural information (in the approximation sense). Therefore, the overall set-overlap-based similarity calculation unfolds two operations that can be independently dealt with: *conversion* of objects to sets and, afterwards, *set-overlap* measurement. As an example, consider the well-known Jaccard similarity that, for two sets let r and s , is given by: $\text{Jacc}(r, s) = \left| \frac{r \cap s}{r \cup s} \right|$.

3.3 Signature Schemes

To avoid bulky similarity evaluation for each pair of sets, a *signature scheme* is commonly used. Given a collection of sets as input, a signature scheme produces a shorter representation of each set, called signature, that roughly maintains their pairwise similarity according to a given measure. An essential *correctness requirement* is that *false negatives* must not be produced [2]: for any two sets r , s , and their respective signatures $\text{Sig}(r)$, $\text{Sig}(s)$, we have $\text{Sig}(r) \cap \text{Sig}(s) \neq \emptyset$ whenever $\text{sim}(r, s) \geq \gamma$.

One previously proposed signature scheme is the prefix-filter [4], which is based on the following intuition: for two sets r and s of size c under a same *total order*, if $|r \cap s| \geq \gamma$, then subsets consisting of the first $c - \gamma + 1$ elements of

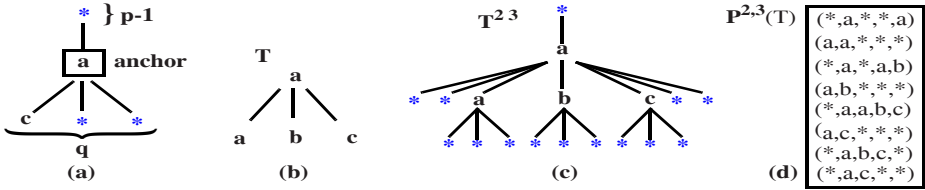


Fig. 2. Steps for the generation of pq -gram tokens

r and s should intersect. Minor variations of this basic idea are used to handle weighted sets and normalized similarity functions. Please, see [4] for details. The ordering of the sets is picked to keep the elements with smallest frequencies in the signature, i.e., the elements are ordered by increasing frequency values.

4 Mapping Trees to Sets by Token Generation

The technique of decomposing a string in substrings of length q , the so-called q -grams, is widely used in the approximate string matching area [11]. The main idea is to assess the “closeness” between two strings by using the overlap of their sets of q -grams [14]. Hence, q -grams provide a natural choice for their use in conjunction with set-overlap-based similarity measures. Moreover, this approach carries over to tree-structured data as well: trees can be split into subtrees of a same shape and the structural similarity between two trees can be calculated on basis of the number of common subtrees. Next, we review a generalization of q -grams for structural similarity assessment and then present approaches for combining structural and textual similarity into a single measure.

4.1 The Concept of pq -Grams

The concept of pq -grams was presented by Augsten et al. [3] to map a *tree structure* to a set of tokens. All subtrees of a specific shape are denoted pq -grams of the corresponding tree. This shape is defined by two positive integer values p and q : a pq -gram consists of an *anchor node* together with $p-1$ ancestors and q children, as visualized by a sample pq -gram in Fig. 2a. The concatenation of the node labels of a pq -gram forms a pq -gram token (pq -gram, for short). To be able to obtain a set of pq -grams from any tree shape, an expanded tree $T^{p,q}$ is (conceptually) constructed from the original T by inserting *null nodes* as follows: $p-1$ ancestors to the root node; $q-1$ children before the first and after the last child of each non-leaf node and q children to each leaf node. Fig. 2b and c show tree T and its expanded form $T^{2,3}$. A pq -gram profile is obtained by collecting all the pq -grams of an expanded tree resulting in a profile cardinality $|P^{p,q}(T)| = 2l + kq - 1$ for a tree with l leaf nodes and k non-leaf nodes. Fig. 2d shows the pq -gram profile of $T^{2,3}$.

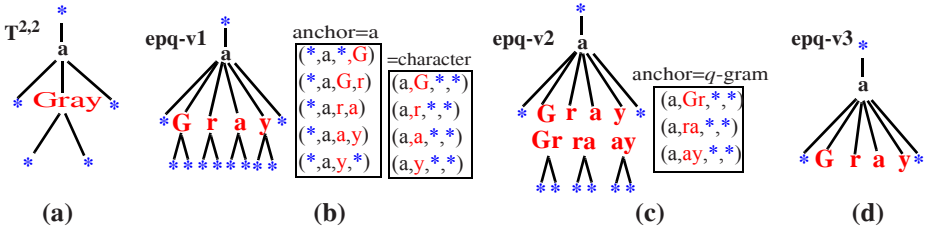


Fig. 3. Versions of extended pq -grams

The set of pq -grams can be easily calculated using a preorder traversal of the corresponding tree [3]. In our case, the input is a stream of structure nodes in *document order* provided by the underlying document access mechanism.

4.2 Adding Content Information

We now study ways to simultaneously deal with structural and textual similarity. The notion of pq -grams captures variations in the XML structure part, but entirely ignores the presence of text variations, i.e., deviations in XML text nodes. Hence, it falls short in identifying duplicates in datasets with poor textual data quality. If we disregard the structure using q -grams only, we may incur in the problem described in Sect. 1: unrelated content may be compared leading to wrong matching results. Hence, an intuitive approach is to produce tokens (grams) that jointly capture structural and textual properties of a tree.

To achieve the objective above, we propose an extension of the original definition of pq -grams. When the objects of interest are represented by strings, their sets of q -grams are used to enrich the respective pq -grams, leading to *extended pq -grams* (epq -grams for short). For epq -gram generation, we focus on the conceptual representation of strings in an expanded tree, denoted by $ET^{p,q}$. This approach allows us to use an almost identical algorithm to produce epq -grams from a stream of nodes, with minor variations to handle text nodes. Furthermore, the generated grams seamlessly reduce to normal pq -grams when the input stream only contains structural nodes. There are several conceivable ways to represent strings as nodes in an expanded tree. Next, we analyze three versions.

The first alternative consists of considering each character of a string as a *character node*. Hence, whenever a parent of a text node is selected as an anchor node, q character nodes are selected to form a new epq -gram version called epq -v1. Given a $T^{2,2}$ in Fig. 3a, the corresponding $ET^{2,2}$ for epq -v1 together with the resulting epq -profile are shown in Fig. 3b. Note that the epq -profile is separated into two subsets: epq -grams having a as anchor node, the other having character nodes as anchor node. When the node labels are concatenated, sequences of character nodes form q -grams, which are combined with structural information. Unfortunately, epq -v1 always forms 1-grams when the character node is the anchor, which is independent of the choice of q . Note that for $q = 1$, different strings containing an identical (*multi*-) set of characters have the same q -gram

profile and, hence, maximum similarity. For $q > 1$, more complex patterns must be present to get this (unwanted) effect [14].

To prevent from the potential drawback of epq -v1, we propose a hybrid approach, called epq -v2 (see Fig. 3c). Now, character nodes are used when the parent is the anchor node, and q -gram nodes when the text node itself is the anchor. As a result, all epq -grams with textual information have q -grams of the same size (epq -grams having a as anchor node are the same to those of epq -v1).

The previous versions may consume substantial space because of large profile sizes. This observation motivates the third approach, epq -v3, which is derived by using character nodes, but pruning their q -null children from the expanded tree (see Fig. 3d). Compared to the previous versions, this approach roughly produces only half of the epq -grams embodying text; therefore, textual similarity receives less weight. However, this property can be compensated by the fact that tokens containing text are likely to be less frequent than structure-only tokens. The rationale is that by using common notions of weights that are inversely proportional to frequency, e.g., IDF, we can balance the effect of the q -gram reduction. In Sect. 6, we empirically evaluate this conjecture.

Theorem 1 shows the relation between the resulting profile cardinality and the number of non-leaf nodes, empty nodes, and text nodes.

Theorem 1. *Let $p > 0$, $q > 0$, and T be a tree with e empty nodes, k non-leaf nodes and t text nodes. Assume that all text nodes have a fixed length of n . The size of the extended epq -gram profile (version 1) is: $|(EP_{v1})^{p,q}(T)| = kq + 2e + 2tn - 1$.*

Proof (Sketch). Theorem 1 can be shown by structural induction similarly to the strategy used in [3]. The deletion of leaves should be done in two stages: first deletion of empty nodes and then deletion of text nodes. Deleting a text node decreases the cardinality of the pq -gram profile by $2n$ if the text node has siblings, otherwise by $2n + q - 2$; deletion of an empty node decreases the cardinality by q if the node has no siblings, otherwise by 2.

Similarly, the profile cardinality can be derived for version 2 and 3 leading to $|(EP_{v2})^{p,q}(T)| = kq + 2e + t(2n - q + 1) - 1$ and $|(EP_{v3})^{p,q}(T)| = kq + 2e + tn - 1$.

Especially when applied to text nodes with long strings, the epq -gram profile can have a cardinality considerably larger than that of normal pq -grams. However, our similarity join methods aim at *data-centric* XML datasets which usually have strings of moderate length. Further, we can use DB accesses to obtain shorter strings in selected parts of a *document-centric* XML for similarity evaluation on content and structure; in the remainder, the evaluation is done on structure only (see Sect. 5.2).

Path-Grams. Additionally, we have explored another approach to combining structure and content information. For each text node, we generated the normal set of q -grams and appended the root-to-leaf path of the containing element node. *Path-gram (PG)* denotes this method of composing information used for similarity computation. In an XDBMS environment, its evaluation can be supported by the documents path synopsis [9] which delivers such path information

for free. Finally, because the path-gram generation is performed on the basis of each text node in isolation, we (conceptually) extended each string by prefixing and suffixing it with $q-1$ null characters (a null character is a special symbol not present in any string) in a way similar to [6].

5 Tree Similarity Joins and Their XDBMS Integration

A tree similarity join needs to locate the root nodes of the candidate subtrees to be compared. For this reason, we use XPath or XQuery expressions which declaratively specify two sets of nodes defining both sides of the join operands. For example, a tree similarity join (*TSJ*) can be used to validate incoming tree structures against an *assume-to-be-correct* (*atbc*) reference data source. Intuitively, we could denote such an operation by $(X)TSJ(Y)$ where X specifies the reference structure (e.g., *doc(atbc.xml)/atbc/article[reviewer = J.Gray]*) and Y the subtrees to be correlated (e.g., *doc(pub.xml)/pub/paper*).

We have designed a family of tree similarity joins, where we focused on the combination of structural and textual similarity. Hence, *TSJ_{EPQ}* and its versions $v1 - v3$ jointly consider tokenized text nodes and element nodes using *epq*-grams (*TSJ_{v1-3}* for short). Furthermore, *TSJ_{PG}* exploits *path*-gram tokens in a similar way. In contrast, *TSJ_{CS}* independently evaluates textual and structural similarity, which can be achieved in either sequence. The final result is the (weighted) average of each evaluation.

5.1 TSJ Processing

The various phases of XDBMS-based join processing are illustrated in Fig. 4. *Subtree access* indicates that the subtrees qualified for *TSJ* have to be identified and fetched from their disk-based storage locations to a memory-resident working area for further processing. This aspect is more than essential, because, in large XML documents, inappropriate selection of subtrees to be checked may consume the largest fraction of the overall response time for a similarity join. Moreover, repeated subtree access may become necessary if the intermediate token sets are too large to be kept in memory. Hence, it is of particular importance that document scans can be avoided and that index-based scans minimize physical disk accesses. In addition, support of predicates based on node type and subtree paths is also required (see Sect. 5.2).

Token generation is essentially determining the quality of the overall *TSJ* processing (see Sect. 6.2). For each qualified subtree, token sets in the form of *q*-grams, *pq*-grams, or *epq*-grams have to be generated whose sizes depend on

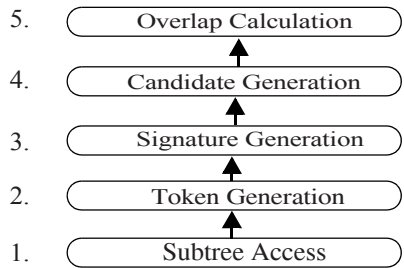


Fig. 4. Phases of TSJ processing

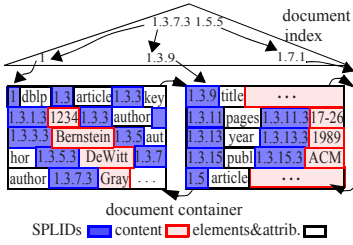


Fig. 5. Stored XML document

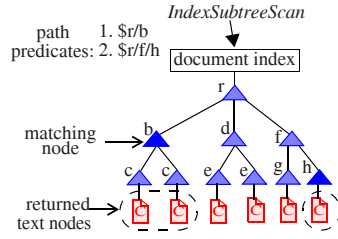


Fig. 6. Path search arguments

the underlying structure and content of the tree and the tokenization method itself (their fundamentals are sketched in Sect. 4). The primary goal of *signature generation* is to filter the potentially large token sets and to derive a compact representation facilitating candidate generation. A signature scheme such as prefix filter considerably reduces the original set of tokens; for example using the (unweighted) Jaccard similarity with a threshold of 0.9, the obtained signature has a cardinality of roughly 10% of the original token set. In the fourth phase, *candidate generation* applies to the sets of signatures delivered by both join operands. Using equi-joins, this method selects those signature sets sharing at least one element. Hence, candidate pairs are prepared for the final processing phase, where *overlap calculation* delivers quantities to which similarity metrics can be directly applied. Depending on threshold values, some pairs of subtrees are satisfying the similarity condition. The nodes of the matching subtrees are then forwarded to the output structure.

5.2 Integration into XTC

As a testbed for XML research, we have developed XTC as a prototype XDBMS exhibiting full DB capabilities [10]. Because similarity join functionality is domain-related and optimal results strongly depend on application characteristics, we did not deeply integrate the *TSJ* operator. Instead we implemented this operator using a plug-in at a higher layer of abstraction while, however, exploiting the efficient XDBMS core functionality such as indexes, scans, etc., to access and extract disk-based XML fragments and provide them for *TSJ* processing.

For all storage structures, we heavily rely on B*-trees as the proven base structure guaranteeing balanced trees, logarithmic accesses, and dynamic reorganization. Most important for fine-granular and flexible processing is the node labeling scheme used for XML trees. A substantial amount of empirical experiments with various schemes led us to the conclusion that prefix-based labeling is superior to all competitor schemes [8]. In XTC, we implemented (after less convincing experiments with other schemes) a variant of prefix-based node labeling, called SPLID scheme (Stable Path Labeling Identifier).

In Fig. 5, we have illustrated the storage structure of a sample XML document, where the B*-tree is composed by the *document index* and the *document container* as a set of doubly-chained pages. Because the document is stored in

document order, the SPLIDs in the container pages lend themselves to prefix compression which typically reduces the storage space needed for SPLIDs to 20% of the original size [8]. Furthermore, element and attribute names are replaced by means of a vocabulary of VocIDs (consisting of two bytes). We can optionally apply content compression based on Huffman codes, thus resulting in a space-economical native XML storage representation. In addition to the document index needed to directly access structure or text nodes via their SPLIDs, we provide a variety of indexes for element/attribute or path access, content search, or even CAS queries (content&structure). All of them are based on B*-trees and use lists of SPLIDs for document access. For example, an element index would deliver for value *author* the locations for all author elements (1.3.3, 1.3.5, 1.3.7, ...), whereas a content index would provide the locations of the indexed text values in the document, e.g., 1.3.7.3, ... for Gray.

The most important access mechanism for *TSJ* is the *IndexSubtreeScan* [12]. With a reference list of SPLIDs, typically delivered from a suitable index, it locates the related nodes for the join operand via the document index. Then, the respective subtrees are scanned and their nodes are delivered to the token generation process. Optionally, *IndexSubtreeScan* may take a set of predicates to return selected parts of a subtree. Predicates based on node type are used to retrieve only structural nodes or textual nodes from a subtree. A more complex type of predicate is the so-called *path predicate* where a path expression is used to locate specific nodes of subtrees and then only the text nodes attached to these nodes are delivered for token set generation. Note that all structural nodes are returned, regardless of whether they are contained in the matching nodes or not. Fig. 6 illustrates an example with two path predicates. Such predicates are used by instances of *TSJ*_{v1-3} to avoid long strings in document-centric XML.

6 Experiments

After having introduced the algorithms and their system embedding, we are ready to present our experimental results. The main goal of our evaluation is to comparatively measure performance and accuracy of the *TSJ* family.

We start with three well-known datasets: *DBLP* containing computer science publications, *IMDB* (www.imdb.com) storing information about movies, and *NASA* presenting astronomical data. *DBLP* and *NASA* are already available in XML format. For the *IMDB* dataset, we generated

an XML document, whose DTD is shown in Fig. 7. Statistics describing all datasets are given in Table 1. (We consider only article subtrees in *DBLP*.) *DBLP*

```

<!ELEMENT imdb (movie)>
<!ELEMENT movie (title, production_year?,
                 cast*, crew*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT production_year (#PCDATA)>
<!ELEMENT cast (actor*, actress*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT actor (name,role?,note?)>
<!ELEMENT actress (name,role?,note?)>
<!ELEMENT crew (producer*, director*)>
<!ELEMENT producer (name, type?)>
<!ATTLIST producer type CDATA >
<!ELEMENT director (name, note?)>

```

Fig. 7. DTD of the IMDB dataset

Table 1. Dataset statistics

dataset	#sub-trees	avg nodes/subtrees	distinct tags/paths	max/avg path length	structure/content ratio	avg node string size	max string size	avg tree string size
DBLP	328838	24	26/37	6/3.02	1.083	19.99	665	239.65
IMDB	380000	56	13/14	5/4.88	1.64	12.9	224	277.85
NASA	2435	371	69/78	8/7.76	1.43	33.22	14918	5069.31

and *IMDB* show data-centric characteristics whereas *NASA* is more document-centric. We then produce “dirty” copies of these datasets by performing controlled transformations on content and structure. We implemented a program for error injection in XML datasets, which enabled the variation of “dirtiness” by specifying parameters: percentage of duplicates to which transformations are applied (*erroneous duplicates*) and extent of transformations applied to each erroneous duplicate (*error extent*). Injected errors on text nodes consist of word swappings and character-level modifications (insertions, deletions, and substitutions). Structural modifications consist of node insertion and deletion, position swapping, and relabeling of nodes. Insertion and deletion operations follow the semantics of the tree edit distance algorithm [15]. Swapping operations replace the whole subtree under the selected node, while relabeling only changes the nodes name (with a DTD-valid substitute). Frequency of modifications (textual and structural) is uniformly distributed among all nodes of each subtree.

In all evaluations, we used the IDF-weighted Jaccard similarity. The ordering of the prefix-filter signature elements is defined by also using IDF weights $w(t)$ specified as $w(t) = 1 + \log\left(\frac{|T_1|+|T_2|}{ft}\right)$, where ft is the total number of subtrees in T_1 and T_2 , which contain t as a token. We use q -grams of size 2 for text nodes and pq -grams with p of size 2 and q of size 2 for structural nodes. We observed best results with this setting, especially in the accuracy experiments. All tests were run on an Intel Pentium IV computer (two 3.2 GHz CPUs, 1GB main memory, 80GB external memory, Java Sun JDK 1.6.0) as the XDBMS server machine where we configured XTC with a DB buffer of 250 8KB-sized pages.

6.1 Performance Results

In the first experiment, we analyze and compare execution time and scalability of our similarity join algorithms and their suboperator components, subtree scan and signature generation, for an increasing number of input trees which were selected from *IMDB* and *DBLP* datasets [4]. We only report the results for *IMDB*, since the results for *DBLP* are consistently similar to them in all experiments.

Subtree scan operators are the main component of the XTC system embedding. Therefore, we can properly observe the integration effects of *TSJ* into the

¹ *NASA* has a richer structure, however, *TSJ_{PG}* and *TSJ_{CS}* do not support path predicates, which is necessary for queries against document-centric XML.

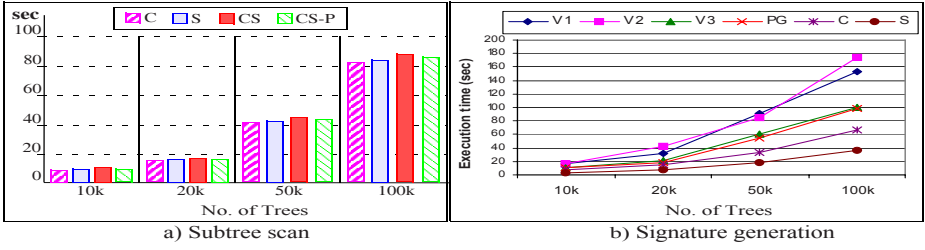


Fig. 8. Suboperator performance results

XTC architecture by analyzing it separately. Fig. 8a compares the execution time of the 4 types of tree scan access used by our operators: content only (C), structure only (S), and content and structure without and with path search arguments (CS and $CS-P$, respectively). The search argument used is the path expression `/movie/cast/author`. All tree scan operators perfectly scale with the input dataset size, which emphasizes that we have achieved a smooth embedding of these operators into the XTC architecture. Furthermore, there is no significant performance variation among the suboperators. Finally, the path search argument used by $CS-P$ does not present any impact on performance.

Fig. 8b shows the results of the prefix-filter signature generation time of the three versions of epq -grams, $path$ -gram, and the two components of TSJ_{CS} , content-only (q -grams) and structure-only (pq -grams). Again, all suboperators scale very well with the input size. The relative results reflect almost exactly the token set sizes produced by each tokenization method. Since the token sets have to be ordered during the prefix-filter computation, larger sets cause higher overhead. This fact is emphasized by the best performance of the signature generation for pq -grams, which produces the shortest token sets, even though the algorithm used for textual token set generation is much simpler. Furthermore, the calculation of pq -gram sets is entirely performed using SPLIDs which are highly optimized in XTC and, consequently, does not negatively impact the performance. The signature generation of $path$ -gram is slower than that of q -gram, because it operates on extended strings resulting in larger token sets.

The results for the complete TSJ evaluation of are depicted in the Fig. 9. For this experiment, we used datasets containing 50% of erroneous duplicates. The error extent was configured to be 30%, i.e., the percentage of erroneous structural and textual nodes in duplicate subtrees. We use a threshold fixed at 0.85. TSJ_{CS} is evaluated by first evaluating text nodes and afterwards structural nodes; this processing sequence was superior in our experiments. In addition, TSJ_{CS}

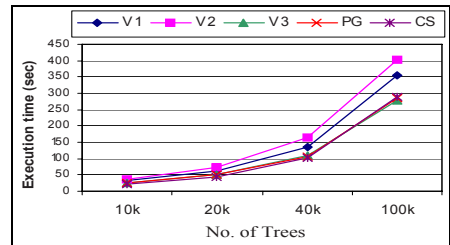


Fig. 9. Full TSJ evaluation

is evaluated in conjunctive mode, where only the subtrees returned by the textual operator are considered by the structural operator.

We observe that all operators scale and do not present dramatic performance variations. TSJ_{PG} , TSJ_{v3} , and TSJ_{CS} are superior and very close, whereas TSJ_{v1} and TSJ_{v2} are 15%–25% slower. Because they produce larger token sets, the number of signature collisions is higher, thereby requiring more similarity evaluations. Finally, we mention that the size of the input dataset determines only partially the execution time. In addition, the number of similar tree pairs occurring in the experiment significantly influences the execution time required.

6.2 Accuracy Evaluation

We now evaluate the quality of our similarity operators. For this experiment, we generate each dataset by first randomly selecting 500 subtrees from the original dataset and then generating 4500 duplicates from them (9 per subtree). As query workload, we randomly select 100 subtrees from the above dataset. For each queried input subtree T , the trees T_R in the result returned by our similarity join are ranked according to their calculated similarity with T . In this experiment, we do not use any threshold parameter, and therefore the rank reported is *complete*. Further, during data generation, we keep track of all duplicates generated from each subtree; those duplicates form a partition and carry the same identifier.

We use well-known evaluation metrics from Information Retrieval to evaluate the quality of our methods: the *non-interpolated Average Precision* (AP), the *maximum F1 score*, and the interpolated precision at recall levels 0.0, 0.1, ..., 1.0. AP is $\frac{1}{\#relevanttrees} \times \sum_{r=1}^N [P(r) \times rel(r)]$, where r is the rank, N the total number of subtrees returned. $P(r)$ is the number of relevant subtrees ranked before r , divided by the total number of subtrees ranked before r , and $rel(r)$ is 1 if the subtree at rank r is relevant and 0 otherwise. The F1 measure is the harmonic mean of precision and recall over the ranking. The interpolated precision at recall r is the highest precision found for recall levels higher than r . We report the mean of the AP and F1 measure over the query workload.

We applied the TSJ_{CS} operator using the weights 0.5 (0.5) and 0.25 (0.75) for structural (textual) similarity score in the weighted average calculation and represented them in our experimental charts as *CS-0.5* and *CS-0.25*, respectively. Following a strategy similar to [1], we classify our test datasets into *dirty*, *moderate*, and *low* error datasets according to the parameters used in the data generation as shown in Table 2. Errors in duplicate subtrees (i.e., the error extent)

Table 2. Duplicate dataset classes

Class	Name	Percentage of	
		erroneous duplicates	error extent
Low	L1	10	10
Low	L2	30	10
Moderate	M1	30	30
Moderate	M2	60	10
Dirty	D1	60	30
Dirty	D2	90	30
-	E1-E4	50	10-50

are applied to structural nodes at the same rate as to the textual nodes. For example, L1 contains 10% of alteration on its text nodes as well as 10% of

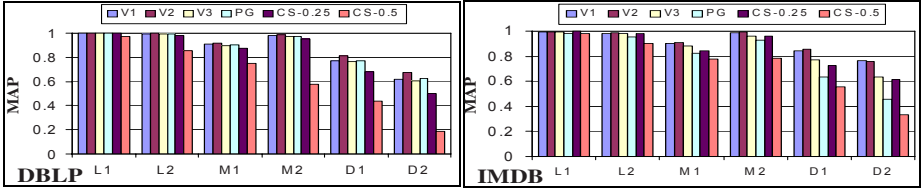


Fig. 10. MAP values for DBLP and IMDB datasets

alteration on its structural nodes. We also generated datasets containing only one specific type of structural error and with a fixed textual error extent to evaluate the effect of specific structural deviations.

Fig. 10 shows the mean AP (MAP) values for the classes of datasets generated from *DBLP* and *IMDB*. All operators perform well on the L1 dataset. The performance of *CS-0.5*, however, suffers a considerable degradation already on L2 and presents a very poor accuracy for dirty datasets. On the other hand, *CS-0.25* shows much more resilience to errors. In fact, its performance is comparable to the *epq*-gram-based operators for low and moderate data set classes. These results demonstrate the higher importance of textual tokens for the similarity evaluation. Because *text* tends to be more selective than structure, our results confirm the common notion that less frequent elements contribute more to similarity assessment. In general, the *epq*-grams operators outperformed all competitors, with *TSJ_{v2}* being the best and *TSJ_{v3}* being the worst among them. Clearly, the reduction of the number of appearances of each character node in version 3 (decreased by 1 achieved by pruning of *qnull*, see Sect. 4.2) negatively impacts the quality of the results.

The next experiment we report explores the sensitivity of our operators to specific types of structural deviations. We consider four types of errors: *Add nodes*, that inserts nodes along a root-to-leaf path, thereby increasing the depth of a subtree; *Tree-up* moves nodes up in a subtree modifying ancestor-descendant relationships; *Swap* changes the ordering of nodes (it may also move nodes to the sibling of its parent); and *Rename* that changes the nodes name. We do not include deletion of trees in this experiment, because this kind of error also (substantially) changes the content of a subtree and, therefore, would blur the results with text-related modifications. For these datasets, we restricted the erroneous duplicates to 50%, the textual error extent to 10%, and increased the structural error extent from 10% to 50% in steps of 10%.

The results are shown in Fig. 11. Our first observation pertains to the poor performance of *TSJ_{PG}* for all types of errors. Because this method relies on root-to-leaf information to generate its token set, it suffers large accuracy degradation as the structural errors increase. *CS-0.25* shows the best results for *Add Nodes* and *Rename* datasets. Indeed, they are practically not affected by these types of structural errors. Because the evaluation of textual and structural similarity is done independently and the structural score has less weight, in addition, the method is resilient to structural errors. However, a natural question that may arise is whether or not it is a good behavior for a similarity measure to report high scores

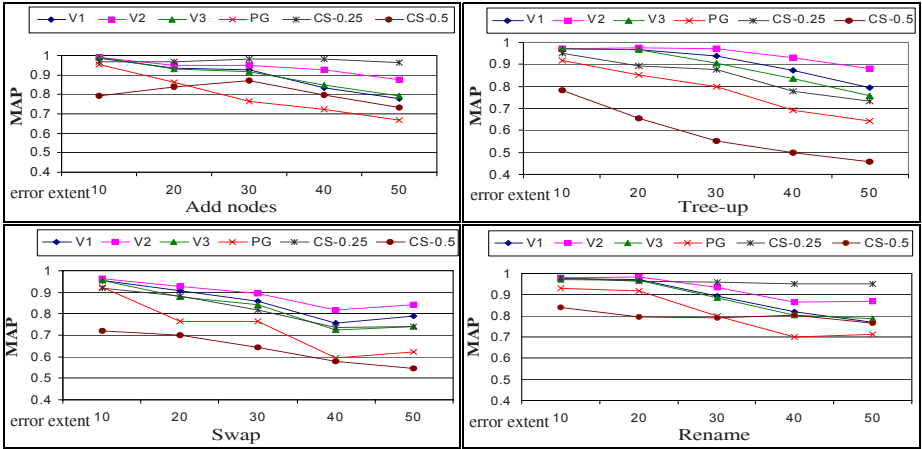


Fig. 11. MAP values for datasets with specific errors

for subtrees containing nodes having different tags. Note that this is similar to the case in our example in Fig. 1. Among the *epq*-gram-based operators, TSJ_{V_2} again presents the best results. Because its underlying tokenization method neither produces *q*-grams of size 1 (in contrast to TSJ_{V_1}) nor decreases the number of text nodes appearing in *epq*-grams (in contrast to TSJ_{V_3}), TSJ_{V_2} can better explore textual similarity to deliver more accurate results.

In our final experiment, we focussed on the *NASA* dataset which has a more document-centric flavour, with much larger string sizes in the content part than the previous datasets. For our experiment, we generated the same classes of datasets described in Table 2. However, we only evaluated the *epq*-grams-based operators, since TSJ_{CS} and TSJ_{PG} do not support path predicates. We used `//author` and `/dataset/title` as path predicates to obtain the textual content of the subtrees. The results shown in Fig. 12 are similar to those of the other datasets: *epq*-v2 has the best accuracy among the various versions of *epq*-grams. We note that, in general, the results of all operators are better than those for *DBLP* and *IMDB*. Besides the textual information obtained by the path predicates, the rich structure of the subtrees provides a large amount of information to be exploited by the similarity evaluation.

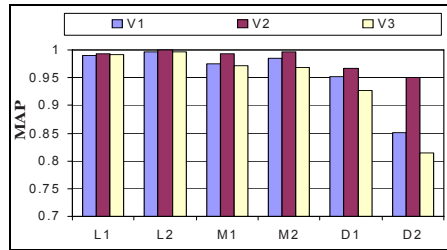


Fig. 12. NASA accuracy results

7 Conclusion

In this paper, we primarily explored an approach to tree-based similarity joins and analyzed its performance and accuracy when embedded in native XDBMSs.

Our results have shown that we achieved a seamless integration of similarity operators into XTC. The internal XDBMS processing, i.e., the specific support of our lower-level suboperators, enabled efficient evaluation of XML documents stored on disk thereby providing scalability in all scenarios considered. Our framework provides multiple instances of similarity joins which support different ways of tree similarity evaluation in a unified manner. We also explored a new concept, the so-called *epq*-grams, combining structural and textual information to improve the quality of similarity joins. Our results have shown that this technique considerably enhances the matching quality, i.e., the accuracy of the similarity join, especially on dirty datasets.

Acknowledgement: Work supported by CAPES/Brazil under grant BEX1129/04-0.

References

1. Amit, C., Hassanzadeth, O., Koudas, N., Sadoghi, M.: Benchmarking Declarative Approximate Selection Predicates. In: Proc. SIGMOD Conf., pp. 353–364 (2007)
2. Arasu, A., Ganti, V., Kaushik, R.: Efficient Set-Similarity Joins. In: Proc. VLDB Conf., pp. 918–929 (2006)
3. Augsten, N., Böhlen, M., Gamper, J.: Approximate Matching of Hierarchical Data using *pq*-Grams. In: Proc. VLDB Conf., pp. 301–312 (2005)
4. Chaudhuri, S., Ganjam, K., Kaushik, R.: A Primitive Operator for Similarity Joins in Data Cleaning. In Proc. ICDE Conf., p. 5 (2006)
5. Gravano, L., Ipeirotis, P., Jagadish, H., Koudas, N., Srivastava, D.: Text Joins in an RDBMS for Web Data Integration. In: Proc. WWW Conf., pp. 90–101 (2003)
6. Gravano, L., Ipeirotis, P., Jagadish, H., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate String Joins in a Database (Almost) for Free. In: Proc. VLDB Conf., pp. 491–500 (2001)
7. Guha, S., Jagadish, H., Koudas, N., Srivastava, D., Yu, T.: Integrating XML Data Sources using Approximate Joins. TODS 31(1), 161–207 (2006)
8. Härder, T., Haustein, M., Mathis, C., Wagner, M.: Node labeling schemes for dynamic XML documents reconsidered. DKE 60(1), 126–149 (2007)
9. Härder, T., Mathis, C., Schmidt, K.: Comparison of Complete and Elementless Native Storage of XML Documents. In: Proc. IDEAS 2007, pp. 102–113 (2007)
10. Haustein, M.P., Härder, T.: An Efficient Infrastructure for Native Transactional XML Processing. DKE 61(3), 500–523 (2007)
11. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. 33(1), 31–88 (2001)
12. Ribeiro, L., Härder, T.: Embedding Similarity Joins into Native XML Databases. In: Proc. 22nd Brazilian Symposium on Databases, pp. 285–299 (2007)
13. Sarawagi, S., Kirpal, A.: Efficient Set Joins on Similarity Predicates. In: Proc. SIGMOD Conf., pp. 743–754 (2004)
14. Ukkonen, E.: Approximate String Matching with *q*-grams and Maximal Matches. Theor. Comput. Science 92(1), 191–211 (1992)
15. Zhang, K., Shasha, D.: Simple Fast Algorithms for the Editing Distance Between Trees and related Problems. SIAM Journal on Computing 18(6), 1245–1262 (1989)

Preserving Functional Dependency in XML Data Transformation*

Md. Sumon Shahriar and Jixue Liu

Data and Web Engineering Lab, School of Computer and Information Science,
University of South Australia, Adelaide, SA-5095, Australia
shamy022@students.unisa.edu.au, jixue.liu@unisa.edu.au

Abstract. With the advent of XML as a data representation and exchange format over the web, a massive amount of data is being stored in XML. The task of data transformation for integration purposes in XML is getting much importance to the research community. In XML data transformation, a source schema and its conforming data are transformed to a target schema. The source schema often has integrity constraints to enforce semantics. One type of constraints is XML functional dependency (XFD). When a source schema is transformed to a target schema, XFDs can also be transformed. Thus, the problems how schema and data transformation should cause XFD transformation becomes important. We study the effects of transformation on XFDs in this research. Towards this problem, we first define the XFDs over the XML Document Type Definition(DTD) and the satisfactions of XFDs. We then show how XFDs are transformed and valid when the DTDs are transformed. We further investigate whether the transformed XFDs are preserved by the transformed data.

1 Introduction

Transformation of data is an important activity in data integration with any data model[1,2]. Historically, the task of data transformation and integration is done mainly in relational data model[1]. In recent years, with the growing use of XML as a data representation and exchange format over the world wide web, much data is stored in XML and hence the task of data transformation and integration in XML [2,3,4] is getting more importance to the database researchers and developers.

In XML data transformation, an XML source schema with data is transformed to the XML target schema. It is quite natural that an XML source schema can often be defined with integrity constraints[9,10] to convey semantics of data. XML functional dependency (XFD) [5,6,7,8] is one type of integrity constraints. When an XML source schema with its conforming data is transformed, XFDs can also be transformed. Thus what are the effects on XFDs in transformation

* This research is supported with Australian Research Council(ARC) Discovery Project Fund, Project No. DP0559202.

becomes important. This motivates our research in this paper. We study how XFDs should be transformed when the schema on which XFDs are defined is transformed, and to what degree the transformed XFDs are satisfied by the transformed data. We illustrate our research problems with an example.

```

<!ELEMENT company(branch+) >
<!ELEMENT branch(bname, employees) >
<!ELEMENT employees(eid, ename)+ >
    
```

Fig. 1. XML DTD D

A motivating example: Consider the DTD D in Fig. 1 that shows information of employees of each branch of a company. Each branch has branch name $bname$ and employees with eid and $ename$. The XML document T in Fig. 2 conforms to the DTD D in Fig. 1. Let $\Phi_a(company/branch, \{employees/eid\} \rightarrow employees/ename)$ be the XFD on D . We call $company/branch$ the **scope**, $employees/eid$ the **determinant**, and $employees/ename$ the **dependent**. Φ_a means that with each branch, eid determines $ename$. This requires that for any two same eid values in a branch, their close $ename$ values must be the same; or equivalently for any two different $ename$ values, their close eid values must be different. When this requirement is satisfied, we say that the XFD is satisfied. If it is not satisfied, the XFD is violated. We now introduce how to check the satisfaction of XFDs. We check the satisfaction of Φ_a branch by branch (the scope). In branch *Sydney*, there are two pairs of $\langle eid, ename \rangle$ values: $\langle (111, John) \rangle$ and $\langle (222, Ron) \rangle$. We note that we do not consider $\langle (111, Ron) \rangle$ as a pair because *Ron* is not close to 111. As the two eid values in the pairs are different, so Φ_a is satisfied by branch *Sydney*. In the same way, Φ_a is satisfied by branch *Perth*. We note that as the scope is branch, so we do not consider $\langle eid, ename \rangle$ pairs from different branches. For example, we do not consider the pairs $\langle (222, Ron) \rangle$ and $\langle (222, Kelly) \rangle$ as they are from two branches.

Now we use a different XFD to show a case of violation. Consider the XFD $\Phi_b(company, \{branch/employees/eid\} \rightarrow branch/employees/ename)$ on D meaning that with the company, eid determines $ename$. In this case, the scope is $company$, the root of the document, and we need to examine all $\langle eid, ename \rangle$

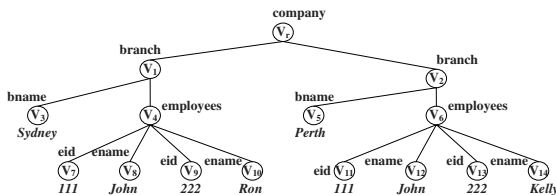


Fig. 2. XML Tree T conforming to D

pairs. We can find two pairs $\langle(222, Ron)\rangle$ and $\langle(222, Kelly)\rangle$ where the two *eid* values are the same but the two *ename* values are different. Thus Φ_b is violated.

Consider another XFD $\Phi_c(company, \{branch/employees\} \rightarrow branch/bname)$ on D meaning that *employees* as a structured value determines *bname* in the *company*. We present structured values by associating the values with their element names and by using brackets to show the levels of trees. Thus the values of $\langle employees, bname \rangle$ pairs are $\langle(employees(eid : 111)(ename : John)(eid : 222)(ename : Ron)), (bname : Sydney)\rangle$ and $\langle(employees(eid : 111)(ename : John)(eid : 222)(ename : Kelly)), (bname : Perth)\rangle$. As the two *employees* structures are different, so Φ_c is satisfied.

Now suppose DTD D and its conforming document T are transformed to DTD \bar{D} in Fig 3 and its conforming document \bar{T} in Fig 4. We use top-bar ($\bar{}$) to mean the transformed result. We assume that the transformation *expands* the structure $(eid, ename)$ using a new element *emp*, may be for the purpose of making the structure clearer.

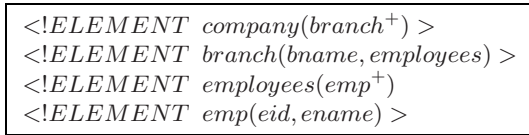


Fig. 3. XML DTD \bar{D} expanding $(eid, ename)$ with new element *emp*

We now consider how Φ_c was affected by the transformation. The determinant path in Φ_c is *branch/employees* and the direct child elements under *employees*, the last element of the path, are *eid* and *ename*. After the transformation, *eid* and *ename* are one level away from *employees* and they are not direct child elements any more. Should we keep *branch/employees* the same to make it simple or should we extend it to *branch/employees/emp* so that *eid* and *ename* are still direct child elements of the determinant? This means when the DTD is transformed, the transformation of XFDs is not automatic. The case here is

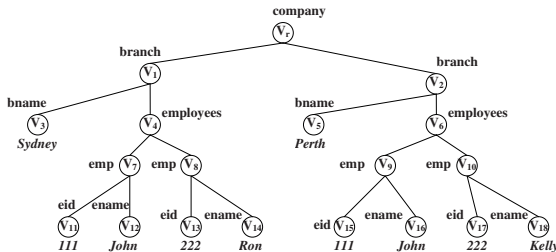


Fig. 4. XML Tree \bar{T} conforming to \bar{D}

only an example. There are cases too requiring the definition on how an XFD should be transformed when a DTD is transformed.

Observation 1. *The transformation of XFDs needs to be defined when the DTD is transformed.*

We next consider how the transformation affects the satisfaction of XFDs. We still use Φ_c as an example. We have shown that Φ_c is satisfied by the tree before the transformation. Now we assume that after the transformation, the XFD becomes $\bar{\Phi}_c(\text{company}, \{\text{branch}/\text{employees}/\text{emp}\} \rightarrow \text{branch}/\text{bname})$, the transformed XFD. We check if the transformed $\bar{\Phi}_c$ is satisfied by the transformed document in Fig. 4. We see that under the root element (the scope), the $\langle \text{emp}, \text{bname} \rangle$ pairs are: $\langle (\text{emp}(\text{eid} : 111)(\text{ename} : \text{John})), (\text{bname} : \text{Sydney}) \rangle$, $\langle (\text{emp}(\text{eid} : 222)(\text{ename} : \text{Ron})), (\text{bname} : \text{Sydney}) \rangle$, $\langle (\text{emp}(\text{eid} : 111)(\text{ename} : \text{John})), (\text{bname} : \text{Perth}) \rangle$, and $\langle (\text{emp}(\text{eid} : 222)(\text{ename} : \text{Kelly})), (\text{bname} : \text{Perth}) \rangle$. Among them, the pairs $\langle (\text{emp}(\text{eid} : 111)(\text{ename} : \text{John})), (\text{bname} : \text{Sydney}) \rangle$ and $\langle (\text{emp}(\text{eid} : 111)(\text{ename} : \text{John})), (\text{bname} : \text{Perth}) \rangle$ violate $\bar{\Phi}_c$. From this example, we see that the transformed document may not satisfy the transformed XFD. This raises the question when a transformed XFD is satisfied and it is not.

Observation 2. *We need to investigate how transformation affects the satisfaction of transformed XFDs.*

While addressing problems, we consider the following contributions.

- First, we define XFDs on the XML DTD [11] and the satisfaction of XFDs by XML documents. A novel concept *tuple* for generating semantically correct values is introduced when the satisfaction is checked.
- Second, we show the transformation of XFD definition using basic transformation operations namely *expand*, *collapse*, *nest* and *unnest* which are considered important in most literatures [2,3,4] for major restructuring of a source DTD with its conforming data to a target DTD. We also check whether the transformation of XFDs makes the transformed XFDs valid. We showed that the transformed XFDs are valid for all the specified operators.
- Last, we study whether the transformed XFDs are preserved by the transformed XML documents. We showed that all operators are XFD preserving with necessary and sufficient conditions.

2 Basic Definitions

In this section, some preliminary definitions are given for the rest of the paper. Our model is the XML DTD [11] with some restrictions. We don't allow the same element names to appear in the same level. We don't allow recursion. We don't consider attributes because there is one-one correspondence between an attribute and an element having the multiplicity '1'.

Our model is denoted by $D = (EN, \beta, \rho)$ where EN contains element names and ρ is the root of the DTD and β is the function defining the types of elements. For example, $\langle !\text{ELEMENT emp1}(\text{eid}, (\text{fname}, \text{lname}), \text{posi}^+) \rangle$ and $\langle !\text{ELEMENT}$

$eid \#PCDATA>$ is represented as $\beta(empl) = [eid \times [fname \times lname] \times posi^+]$ and $\beta(eid) = Str$. Round brackets and commas are not used because they appear frequently in the normal presentation. An element name and a pair of squared brackets '[']' each, with its multiplicity, is called a component. For example, eid , $[fname \times lname]$, and $posi^+$ are three components. A sequence of components, often denoted by g , is called a structure s.t. $g' = eid \times [fname \times lname]$ and $g'' = posi^+$. A structure can be further decomposed into substructures such as g' can be decomposed into $g' = g_1 \times g_2$ where $g_1 = eid$ and $g_2 = [fname \times lname]$. We note that special cases of substructures are components and the multiplicities can only be applied to components as g^c where $c \in [?, 1, +, *]$. We use $mul(e)$ to mean the *multiplicity* of the element $e \in EN$ in D .

We define operations for multiplicities. The meaning of a multiplicity can be represented by an integer interval. Thus the intervals of $?$, 1 , $+$, and $*$ are $[0, 1]$, $[1, 1]$, $[1, m]$, $[0, m]$ respectively. The operators for multiplicities c_1 and c_2 are \oplus , \ominus and \supseteq . $c_1 \oplus c_2$ is the multiplicity whose interval encloses those of c_1 and c_2 . For example, $+\oplus? = *$ and $1\oplus? = ?$. $c_1 \ominus c_2$ is the multiplicity whose interval equals to the interval of c_1 taking that of c_2 and adding '1'. Thus $?\ominus? = 1$ and $*\ominus+ = ?$. $c_1 \supseteq c_2$ means that c_1 's interval contains c_2 's interval.

An XML tree T for an XML document in our notation is represented as $T = (v : e (T_1 T_2 \dots T_f))$ when T has subtrees $T_1 T_2 \dots T_f$; otherwise $T = (v : e : txt)$ if leaf node with the text txt . v is the node identifier which can be omitted when the context is clear and e is the label on the node. $T_1 \dots T_f$ are subtrees.

Example 1. Consider the DTD D in Fig 1. We denote D as $\beta(company) = [branch^+]$, $\beta(branch) = [bname \times employees]$, $\beta(employees) = [eid \times ename]^+$, $\beta(bname) = Str$, $\beta(eid) = Str$, and $\beta(ename) = Str$.

Example 2. The tree T in Fig 2 is represented as $T_{v_r} = (v_r : company(T_{v_1} T_{v_2}))$, $T_{v_1} = (v_1 : branch(T_{v_3} T_{v_4}))$, $T_{v_3} = (v_3 : bname : Sydney)$, $T_{v_4} = (v_4 : employees(T_{v_7} T_{v_8} T_{v_9} T_{v_{10}}))$, $T_{v_7} = (v_7 : eid : 111)$, $T_{v_8} = (v_8 : ename : John)$, $T_{v_9} = (v_9 : eid : 222)$, $T_{v_{10}} = (v_{10} : ename : Ron)$, In the similar way, we represent $T_{v_2} = (v_2 : branch(T_{v_5} T_{v_6}))$, $T_{v_5} = (v_5 : bname : Perth)$, $T_{v_6} = (v_6 : employees(T_{v_{11}} T_{v_{12}} T_{v_{13}} T_{v_{14}}))$, $T_{v_{11}} = (v_{11} : eid : 111)$, $T_{v_{12}} = (v_{12} : ename : John)$, $T_{v_{13}} = (v_{13} : eid : 222)$, $T_{v_{14}} = (v_{14} : ename : Kelly)$.

We now use an example to motivate the next concept, hedge which is critical to the presentation of the paper. Consider $\beta(employees) = [eid \times ename]^+$ for the DTD D in Fig 1. The trees $T_{v_7} T_{v_8} T_{v_9} T_{v_{10}}$ form a sequence conforming to $[eid \times ename]^+$ (note the multiplicity '+'). However, when we consider the structure $g = eid \times ename$ (without '+'), there are two sequences conforming to g : $T_{v_7} T_{v_8}$ and $T_{v_9} T_{v_{10}}$. To reference various structures and their conforming sequences, we introduce the concept **hedge**, denoted by H^g , which is a sequence of trees conforming to the structure g . Thus, $H_1^g = T_{v_7} T_{v_8}$, $H_2^g = T_{v_9} T_{v_{10}}$, and $H^{\beta(employees)} = H^{[eid \times ename]^+} = H_1^g H_2^g = T_{v_7} T_{v_8} T_{v_9} T_{v_{10}}$. Hedges are context dependent. When we consider H_1^g as the context, then $H^{eid} = T_{v_7}$. When context node is v_6 , $H_1^g = T_{v_{11}} T_{v_{12}}$.

Definition 1 (Hedge). A hedge H is a consecutive sequence of primary subtrees $T_1 T_2 \cdots T_n$ of the same node that conforms to the definition of a specific structure g , denoted by $H \in g$ or H^g :

- (1) if $g = e \wedge \beta(e) = \text{Str}, H = T = (v : e : \text{txt})$;
- (2) if $g = e \wedge \beta(e) = g_1, H = T = (v : e : H')$ and $H' \in g_1$;
- (3) if $g = \epsilon, H = T = \phi$;
- (4) if $g = g_1 \times g_2, H = H_1 H_2$ and $H_1 \in g_1$ and $H_2 \in g_2$;
- (5) if $g = g_1^c \wedge g_1 = e, H = (eH_1) \cdots (eH_f)$ and $\forall i = 1, \dots, f (H_i \in \beta(e))$ and f satisfies c ;
- (6) if $g = g_1^c \wedge g_1 = [g], H = H_1 \cdots H_f$ and $\forall i = 1, \dots, f (H_i \in g)$ and f satisfies c . \square

When there are multiple H^g s for a structure g , we use H_j^g to denote one of them and H^{g*} to denote all of them.

Definition 2 (Tree Conformation). Given a DTD $D = (EN, \beta, \rho)$ and XML Tree T , T conforms to D denoted by $T \in D$ if $T = (\rho H^{\beta(\rho)})$. \square

Example 3. Consider the DTD D in Fig 1 and the XML tree T in Fig 2. $T \in D$ because $T_{v_r} = (\text{company} H^{\text{branch}^+})$. Now $H^{\text{branch}^+} = H_1^{\text{branch}} H_2^{\text{branch}} = T_{v_1} T_{v_2}$. $T_{v_1} = (\text{branch} H^{\text{bname} \times \text{employees}})$ where the hedge $H^{\text{bname} \times \text{employees}} = (T_{v_3} T_{v_4})$. $T_{v_4} = (\text{employees} H^{\text{eid} \times \text{ename}^+})$ where the hedges $H^{\text{eid} \times \text{ename}^+} = H_1^{\text{eid} \times \text{ename}^+} H_2^{\text{eid} \times \text{ename}^+}$ and $H_1^{\text{eid} \times \text{ename}^+} = T_{v_7} T_{v_8}$ and $H_2^{\text{eid} \times \text{ename}^+} = T_{v_9} T_{v_{10}}$. In similar way, $T_{v_2} = (\text{branch} H^{\text{bname} \times \text{employees}})$ where $H^{\text{bname} \times \text{employees}} = (T_{v_5} T_{v_6})$. The tree $T_{v_6} = (\text{employees} H^{\text{eid} \times \text{ename}^+})$ where $H^{\text{eid} \times \text{ename}^+} = H_1^{\text{eid} \times \text{ename}^+} H_2^{\text{eid} \times \text{ename}^+}$ and $H_1^{\text{eid} \times \text{ename}^+} = T_{v_{11}} T_{v_{12}}$ and $H_2^{\text{eid} \times \text{ename}^+} = T_{v_{13}} T_{v_{14}}$.

Definition 3 (Hedge Equivalence). Two trees T_a and T_b are value equivalent, denoted by $T_a =_v T_b$, if

- (1) $T_a = (v_1 : e : \text{txt1})$ and $T_b = (v_2 : e : \text{txt1})$, or
- (2) $T_a = (v_1 : e : T_1 \cdots T_m)$ and $T_b = (v_2 : e : T'_1 \cdots T'_n)$ and for $i = 1, \dots, m (T_i =_v T'_i)$.

Two hedges H_x and H_y are value equivalent, denoted as $H_x =_v H_y$, if

- (1) both H_x and H_y are empty, or
- (2) $H_x = T_1 \cdots T_m$ and $H_y = T'_1 \cdots T'_n$ and $m = n$ and for $i = 1, \dots, m (T_i =_v T'_i)$. \square

$T_x \equiv T_y$ if T_x and T_y refer to the same tree. We note that, if $T_x \equiv T_y$, then $T_x =_v T_y$.

Example 4. In Fig 2, $T_{v_7} = (v_7 : \text{eid} : 111)$ and $T_{v_{11}} = (v_{11} : \text{eid} : 111)$. Thus $T_{v_7} =_v T_{v_{11}}$. But $T_{v_7} \neq_v T_{v_9}$ because $T_{v_7} = (v_7 : \text{eid} : 111)$ and $T_{v_9} = (v_9 : \text{eid} : 222)$. Let $H_1 = H^{\text{eid} \times \text{ename}} = T_{v_7} T_{v_8} = (v_7 : \text{eid} : 111)(v_8 : \text{ename} : \text{John})$ of node v_4 and $H_2 = H^{\text{eid} \times \text{ename}} = T_{v_{11}} T_{v_{12}} = (v_{11} : \text{eid} : 111)(v_{12} : \text{ename} : \text{John})$ of node v_6 . Thus $H_1 =_v H_2$ because $T_{v_7} =_v T_{v_{11}}$ and $T_{v_8} =_v T_{v_{12}}$.

Now we introduce the concept of minimal hedge. The need for this concept can be demonstrated by Fig 2. In the figure, there are two *eid* nodes and two *ename* nodes under v_4 . To check the satisfaction of the XFD $\Phi_a(\text{company/branch}, \{\text{employees/eid}\} \rightarrow \text{employees/ename})$ (see Section 1), we need to find *ename* values that are close to an *eid* value. The word 'close' is to restrict the *ename* value *John* to be associated to the *eid* value 111 and to prevent *Ron* to be associated to 111 or *John* to 222. If *eid* and *ename* are two elements, then the minimal structure of the elements are $[\text{eid} \times \text{ename}]^+$, denoted by g , and there are two minimal hedges of g under v_4 : $H_1^g = T_{v_7}T_{v_8}$ and $H_2^g = T_{v_9}T_{v_{10}}$. Thus, when we conduct XFD check, we take values from the same minimal hedge.

Definition 4 (Minimal hedge). Given a DTD definition $\beta(e)$ and two elements e_1 and e_2 in $\beta(e)$, the minimal structure g of e_1 and e_2 in $\beta(e)$ is the pair of brackets that encloses e_1 and e_2 and any other structure in g does not enclose both. Given a hedge H of $\beta(e)$, a minimal hedge of e_1 and e_2 is one of H^g s in H . □

Definition 5 (Simple Path, Complete Path, Empty Path). Given a $D = (EN, \beta, \rho)$, a simple path \wp in D is a sequence $e_1/\dots/e_k/\dots/e_m$, where $e_k \in EN$ and $\forall e_w \in [e_2, \dots, e_m]$ (e_w is a symbol in the alphabet of $\beta(e_{w-1})$). A simple path \wp is a complete path if $e_1 = \rho$. A path \wp is empty if $m = 0$, denoted by $\wp = \epsilon$. We use function $\text{last}(\wp)$ to return e_m , $\text{beg}(\wp) = e_1$, and $\text{par}(e_w) = e_{w-1}$, the parent of e_w . We use $\text{len}(\wp)$ to return m . Let $\wp_1 = e_1/\dots/e_i/\dots/e_m$ and $\wp_2 = e'_1/\dots/e'_i/\dots/e'_n$ are two paths. We define intersected path $\wp_1 \cap \wp_2 = e_1/\dots/e_i$ where $j \in [1, \dots, i](e_j = e'_j)$ and $e_{i+1} \neq e'_{i+1}$. Paths satisfying this definition are said **valid** on D . □

Example 5. Consider the DTD D in Fig 1. In D , *branch/bname* is a simple path and *company/branch/bname* is a complete path. So, $\text{par}(\text{bname}) = \text{branch}$ and $\text{len}(\text{company/branch/bname}) = 3$. Also $\text{last}(\text{company/branch/bname})$ returns *bname*, $\text{beg}(\text{company/branch/bname})$ returns *company*.

Definition 6 (Functional Dependency). An XML functional dependency over the XML DTD can be defined as $\Phi(S, P \rightarrow Q)$ where S is a complete path, P is a set of simple paths as $\{\wp_1, \dots, \wp_i, \dots, \wp_l\}$, and Q is a simple path or empty path. S is called **scope**, P is called **LHS** or **determinant**, and Q is called **RHS** or **dependent**. S/P and S/Q are complete paths. □

If $Q = \epsilon$, then XFD is $\Phi(S, P \rightarrow \epsilon)$. It implies that $P \rightarrow \text{last}(S)$ meaning that P determines S . An XFD following the above definition is valid, denoted as $\Phi \sqsubset D$.

Example 6. Consider $\Phi(\text{company/branch}, \{\text{employees/eid}\} \rightarrow \text{employees/ename})$ on the DTD D in Fig 1. Here, $S = \text{company/branch}$ is a complete path. $P = \{\text{employees/eid}\}$, and $Q = \text{employees/ename}$ are simple paths. We see that the paths *company/branch/employees/eid* and *company/branch/employees/ename* are also complete paths. Consider another XFD $\Phi(\text{company/branch}, \{\text{bname}\} \rightarrow \epsilon)$ that implies *bname* determines the *branch*.

Definition 7 (Prefixed Trees). We use T^e to denote a tree rooted at a node labeled by the element name e . Given path $e_1/\dots/e_m$, we use $(v_1 : e_1)\dots(v_{m-1} : e_{m-1}).T^{e_m}$ to mean the tree T^{e_m} with its ancestor nodes in sequence, called the prefixed tree or the prefixed format of T^{e_m} . Given path $\wp = e_1/\dots/e_m$, T^\wp denotes $(v_1 : e_1)\dots(v_{m-1} : e_{m-1}).T^{e_m}$.

A tree T^\wp is complete if w is a path under \wp and if \wp has a node n_1 , then w must have a node n_2 under n_1 . Also, if $\text{last}(w) = \text{Str}$, then n_2 must have a non-null value.

$\langle T^\wp \rangle$ is the set of all T^\wp and $\langle T^\wp \rangle = \{T_1^\wp, \dots, T_f^\wp\}$. $|\langle T^\wp \rangle|$ returns the number of T^\wp in $\langle T^\wp \rangle$. We use $T^\wp \in T^S$ to mean that T^\wp is a sub tree of T^S where $\wp \in P$ or $\wp = Q$. \square

Example 7. Consider the document T in Fig. 2. T^{company} means the tree T_{v_7} . T^{branch} means T_{v_1} and T_{v_2} . So, $\langle T^{\text{branch}} \rangle = \{T_{v_1}, T_{v_2}\}$. In similar way, T^{name} means the trees T_{v_3} and T_{v_5} . Now consider the path $S = \text{company/branch}$ over the DTD D in Fig. 1. Thus, T^S means T^{branch} , $\langle T^S \rangle = \{T_{v_1}, T_{v_2}\}$, and $|\langle T^S \rangle| = 2$. Now consider the paths $S = \text{company/branch}$ and $P = \{\text{bname}\}$. So, $T^S = T^{\text{branch}}$ and $\langle T^S \rangle = \{T_{v_1}, T_{v_2}\}$. Also, $T^P = T^{\text{name}}$. Now, $\langle T^P \rangle = \{T_{v_3}\}$ in T_{v_1} and $\langle T^P \rangle = \{T_{v_5}\}$ in T_{v_2} .

Definition 8 (Tuple). Consider $\Phi(S, P \rightarrow Q)$ where $P = \{\wp_1, \dots, \wp_i, \dots, \wp_l\}$ and $Q = \wp_{l+1}$. So $\{\wp_1, \dots, \wp_l, \wp_{l+1}\}$ is a set of paths. A tuple is a sequence of subtrees $(T^{\wp_1} \dots T^{\wp_l} T^{\wp_{l+1}})$ that are pair-wise close.

Let $\wp_i = e_1/\dots/e_k/e_{k+1}/\dots/e_m$ and $\wp_j = e'_1/\dots/e'_k/e'_{k+1}/\dots/e'_n$. Let $T^{\wp_i} = (v_1 : e_1)\dots(v_k : e_k).(v_{k+1} : e_{k+1})\dots.T^{e_m}$ and $T^{\wp_j} = (v'_1 : e'_1)\dots(v'_k : e'_k).(v'_{k+1} : e'_{k+1})\dots.T^{e_n}$. Then T^{\wp_i} and T^{\wp_j} are pair-wise close if

(a) If $e_1 \neq e'_1$, then $(v_1 : e_1)$ and $(v'_1 : e'_1)$ are the nodes of the same minimal hedge of e_1 and e'_1 .

(b) If $e_1 = e'_1, \dots, e_k = e'_k, e_{k+1} \neq e'_{k+1}$, then $v_k = v'_k, (v_{k+1} : e_{k+1})$ and $(v'_{k+1} : e'_{k+1})$ are two nodes in the same minimal hedge of e_{k+1} and e'_{k+1} . \square

Let F be the tuple defined above. We denote $F[P] = (T^{\wp_1} \dots T^{\wp_l})$ as P-tuple and $F[Q] = (T^{\wp_{l+1}})$ as Q-tuple. A P-tuple $F[P]$ is complete if $\forall T^{\wp_i} \in (T^{\wp_1} \dots T^{\wp_l})$ (T^{\wp_i} is complete). Similarly, a Q-tuple is complete if $T^{\wp_{l+1}}$ is complete. We use $\langle F[P] \rangle$ to denote all possible P-tuples and $|\langle F[P] \rangle|$ means the number of such P-tuples. Two P-tuples $F_1[P] = (T_1^{\wp_1} \dots T_1^{\wp_l})$ and $F_2[P] = (T_2^{\wp_1} \dots T_2^{\wp_k})$ are value equivalent, denoted by $F_1[P] =_v F_2[P]$ if $l = k$ and for each $i = 1, \dots, k$ ($T_1^{\wp_i} =_v T_2^{\wp_i}$). Similarly, we denote $F_1[Q] =_v F_2[Q]$ to mean $F_1[Q]$ and $F_2[Q]$ are value equivalent where $F_1[Q] = (T_1^{\wp_{l+1}})$ and $F_2[Q] = (T_2^{\wp_{l+1}})$.

Example 8. Let $\Phi(\text{company/branch}, \{\text{employees/eid}\} \rightarrow \text{employees/ename})$ be the XFD on D in Fig. 1. We take $\wp_1 = \text{employees/eid}$ and $\wp_2 = \text{employees/ename}$ to generate the tuples. So the minimal structure for these paths is $[\text{eid} \times \text{ename}]$ because the paths have intersected path $\wp_1 \cap \wp_2 = \text{employees}$ and then $\text{eid} \neq \text{ename}$. Thus, the minimal hedges for the structure $[\text{eid} \times \text{ename}]$ are $H_1 = T_{v_7} T_{v_8}, H_2 = T_{v_9} T_{v_{10}}$ in the T_{v_1} and $H'_1 = T_{v_{11}} T_{v_{12}}, H'_2 = T_{v_{13}} T_{v_{14}}$ in the T_{v_2} .

Now we are ready to generate the tuples from H_1 and H_2 . To generate the P-tuple meaning that tuples for the paths P , we now consider the paths

$\wp_1 = employees/eid$. So in T_{v_1} , $F_1[P] = (T_{v_7}) = ((v_7 : eid : 111))$ for H_1 and $F_2[P] = (T_{v_9}) = ((v_9 : eid : 222))$ for H_2 are the P-tuples. For Q-tuple, meaning that tuples for the paths Q , we take the path $\wp_2 = employees/ename$. So in T_{v_1} , the Q-tuples are $F_1[Q] = (T_{v_8}) = ((v_8 : ename : John))$ for H_1 and $F_2[Q] = (T_{v_{10}}) = ((v_{10} : ename : Ron))$ for H_2 .

In T_{v_2} , the P-tuples are $F'_1[P] = (T_{v_{11}}) = ((v_{11} : eid : 111))$ for H'_1 and $F'_2[P] = (T_{v_{13}}) = ((v_{13} : eid : 222))$ for H'_2 . The Q-tuples are $F'_1[Q] = (T_{v_{12}}) = ((v_{12} : ename : John))$ for H'_1 and $F'_2[Q] = (T_{v_{14}}) = ((v_{14} : ename : Kelly))$ for H'_2 .

Definition 9 (Functional Dependency Satisfaction). *Given a DTD D , an XML document T satisfies the XML functional dependency $\Phi(S, P \rightarrow Q)$, denoted as $T \prec \Phi$ if the followings are held.*

(a) *If $Q = \epsilon$, then $\forall F[P] \in T^S$, $F[P]$ is complete.*

(b) *Else*

(i) *$\exists (F[P], F[Q]) \in T^S$ and $F[P], F[Q]$ are complete.*

(ii) *For every pair of tuples F_1 and F_2 in T^S , if $F_1[P] =_v F_2[P]$, then $F_1[Q] =_v F_2[Q]$. \square*

Example 9. Let $\Phi(company/branch, \{employees/eid\} \rightarrow employees/ename)$ on the DTD D . We want to check whether the document T satisfies the XFD Φ . As the scope is $company/branch$, we see that there are two trees rooted at $branch$. These are T_{v_1} and T_{v_2} . Now we refer to the Example [8](#) for tuples generated by the paths. As $F_1[P] \neq_v F_2[P]$ and then $F_1[Q] \neq_v F_2[Q]$ in T_{v_1} and also $F'_1[P] \neq_v F'_2[P]$ and then $F'_1[Q] \neq_v F'_2[Q]$ in T_{v_2} , so T satisfies Φ .

Consider $\Phi(company, branch/employees/eid \rightarrow branch/employees/ename)$. Here, the scope is $T^{company} = T^{v_r}$. Then $T \not\prec \Phi$ because there exists $F_2[P] =_v F'_2[P]$ but $F_2[Q] \neq_v F'_2[Q]$ in T^{v_r} .

3 Transformation Operations

We now introduce important transformation operations. To understand the effect of these operators on XFD transformation and preservation, we need to know how they work.

Definition 10 (Expand). *The expand operator uses a new element name to push a component one level away from the root. Let e_{new} be the new element and g_d be the component to be pushed. If $g = g_d \neq \epsilon$ where g_d is a component in $\beta(e)$ and $e_{new} \notin \beta(e)$, then $expand(g_d, e_{new}) \rightarrow g = e_{new} \wedge \beta(e_{new}) = g_d$. With documents, $expand(H) \rightarrow \bar{H}$ where $H = H^g$ and $\bar{H} = (e_{new}H^g)$. \square*

Example 10. Let $\beta(\rho) = [A \times [B \times C]^+]^+$. Let the tree be $T = (\rho(A : 1)(B : 2)(C : 3)(B : 4)(C : 5)(A : 6)(B : 7)(C : 8))$. Note that we can expand either $[B \times C]$ (without '+') or $[B \times C]^+$ (with '+'). Then, after $expand([B \times C], E)$, $\beta_1(\rho) = [A \times E^+]^+$, $\beta_1(E) = [B \times C]$ and $\bar{T} = (\rho(A : 1)(E(B : 2)(C : 3))(E(B : 4)(C : 5))(A : 6)(E(B : 7)(C : 8)))$. If we do $expand([B \times C]^+, E)$, $\beta_1(\rho) =$

$[A \times E]^+, \beta_1(E) = [B \times C]^+$ and $\bar{T} = (\rho(A : 1)(E(B : 2)(C : 3)(B : 4)(C : 5))(A : 6)(E(B : 7)(C : 8)))$. We see that whether pushing down the multiplicity in expand operator results in different tree structures which has different effects on XFD transformation and satisfaction.

Definition 11 (Collapse). *The collapse operator uses the definition of an element to replace that element name. Let e_{coll} be the element to be collapsed. If $g = e_{coll}^c \wedge \beta(e_{coll}) = [g_{e_{coll}}]^{c_1}$ and $g_{e_{coll}} \cap \text{par}(e_{coll}) = \phi$, then $\text{collapse}(e_{coll}) \rightarrow g = [g_{e_{coll}}]^{c \oplus c_1}$. With documents, $\text{collapse}(H) \rightarrow \bar{H}$ where $H = H^g = (e_{coll} H_1^{[g_{e_{coll}}]^*}) \dots (e_{coll} H_m^{[g_{e_{coll}}]^*})$, m satisfies c and $\bar{H} = H_1^{[g_{e_{coll}}]^*} \dots H_m^{[g_{e_{coll}}]^*}$. \square*

Example 11. Let $\beta(\rho) = [A \times B]^+, \beta(B) = [C]$. Let the tree be $T = (\rho(A : 1)(B(C : 3)(A : 1)(B(C : 5))))$. Then, after $\text{collapse}(B), \beta_1(\rho) = [A \times C]^+$ and $\bar{T} = (\rho(A : 1)(C : 3)(A : 1)(C : 5))$. Note that we can't $\text{collapse}(C)$ because $\beta(C) = \text{Str}$.

Definition 12 (UnNest). *The unnest operation on g_2 in $[g_1 \times g_2^{c_2}]^c$ is defined as, if $g = [g_1 \times g_2^{c_2}]^c \wedge c_2 = +|*$, then $\text{unnest}(g_2) \rightarrow [g_1 \times g_2^{c_2 \oplus +}]^{c \oplus +}$. Also, $\text{unnest}(H) \rightarrow \bar{H}$ where $H = H^g = H_1^{g_1} H_{11}^{g_2} \dots H_{1n_1}^{g_2} \dots H_m^{g_1} H_{m1}^{g_2} \dots H_{mn_m}^{g_2}$ and the transformed hedge is $\bar{H} = H_1^{g_1} H_{11}^{g_2} \dots H_1^{g_1} H_{1n_1}^{g_2} \dots H_m^{g_1} H_{m1}^{g_2} \dots H_m^{g_1} H_{mn_m}^{g_2}$. \square*

Example 12. Let $\beta(\rho) = [A \times B]^+$ and the tree $T = (\rho(A : 1)(B : 2)(B : 3)(A : 4)(B : 5))$. Then, after $\text{unnest}(B), \beta_1(\rho) = [A \times B]^+$ and $\bar{T} = (\rho(A : 1)(B : 2)(A : 1)(B : 3)(A : 4)(B : 5))$.

Definition 13 (Nest). *The nest operation on g_2 in $[g_1 \times g_2^{c_2}]^c$ is defined as, if $g = [g_1 \times g_2^{c_2}]^c \wedge c \supseteq +$, then $\text{nest}(g_2) \rightarrow [g_1 \times g_2^{c_2 \oplus +}]^c$. Also, $\text{nest}(H) \rightarrow \bar{H}$ where $H = H^g = H_1^{g_1} H_1^{g_2^*} \dots H_n^{g_1} H_n^{g_2^*}, \bar{H} = H_1^{g_1} H_{11}^{g_2^*} \dots H_{1f_1}^{g_2^*} \dots H_h^{g_1} H_{h1}^{g_2^*} \dots H_{hf_h}^{g_2^*}$ where $\forall i = 1, \dots, h(H_{i1}^{g_1} H_{i1}^{g_2^*} \dots H_{if_i}^{g_1} H_{if_i}^{g_2^*})$ in H s.t. $H_i^{g_1} = H_{i1}^{g_1} = \dots = H_{if_i}^{g_1}$ and $\nexists i, j \in [1, \dots, h](i \neq j \Rightarrow H_i^{g_1} \neq H_j^{g_1})$. \square*

Example 13. Let $\beta(\rho) = [A \times B]^*$. Let the tree be $T = (\rho(A : 1)(B : 2)(A : 1)(B : 3)(A : 2)(B : 5))$. Then, after $\text{nest}(B), \beta_1(\rho) = [A \times B]^*$ and $\bar{T} = (\rho(A : 1)(B : 2)(B : 3)(A : 2)(B : 5))$.

4 Transformation of XML Functional Dependency

In this section, we first study the effects of transformation operators on XML functional dependency and we answer the first two questions: how an XFD can be transformed in correspondence to the transformation of the DTD and the document, and whether a transformed definition is valid. In next section, we will study the preservation of XML functional dependencies.

Given a DTD D and a document T such that $T \in D$, a transformation τ transforms D to \bar{D} and T to \bar{T} , denoted by $\tau(D, T) \rightarrow (\bar{D}, \bar{T})$. The problem whether \bar{T} conforms to \bar{D} was investigated in [4]. In this paper, we investigate how the transformation affects the properties of an XFD defined on D . More

formally, given D, T, Φ and a transformation τ such that $\Phi \sqsubseteq D \wedge T \in D \wedge T \prec \Phi$, and $\tau(D, T, \Phi) \rightarrow (\bar{D}, \bar{T}, \bar{\Phi})$, we would like to know what $\bar{\Phi}$ is, whether $\bar{\Phi}$ is valid on \bar{D} , and whether \bar{T} satisfies $\bar{\Phi}, \bar{T} \prec \bar{\Phi}$.

4.1 Transformation of XFD Definition

We now define the transformation of XFDs. We assume τ be a primitive transformation operator because if every τ transforms an XFD to a valid XFD, then a sequence of operators also transforms the XFD valid. In defining the transformation, we need to refer to the DTD type structure g and the paths φ of an XFD. We define the notation for this purpose. To describe the relationship between a type structure g and a path φ on a DTD, we define $g \diamond \varphi \triangleright e$, reading g **crossing** φ **at** e , to mean that element e is in the type structure g and is also a label on path φ , that is $g = [\dots \times e \times \dots] \wedge \varphi = e_1 / \dots / e_k / \dots / e_m$. We say that e is the *cross point* of g and φ . Given D, τ , and $\Phi(S, P \rightarrow Q)$, if a path φ in Φ is not crossed by a type structure $g \in D$, the structure to be transformed by τ , then $\bar{\varphi} = \varphi$.

We now present the transformation of XFDs with regard to the transformation operators that we described in the previous section. We mention here that an XFD Φ is transformed only when path $\varphi = e_1 / \dots / e_k / \dots / e_m$ in Φ is transformed. The operators *nest* and *unnest* do not change an XFD because they manipulate the multiplicities of a type structure and do not change the paths in an XFD. In other words, $\tau(\Phi) = \Phi$. We note that these operators may still affect the satisfaction of an XFD, which will be shown in the next section. The other two operators *expand* and *collapse* change the paths. We now define how they change the XFDs.

Transformation on XFD Using *expand*. We use an example to show how the XFDs should be transformed by the *expand* operator and then we give the definition of XFD transformation.

Example 14. Let $\Phi'(company/branch, \{employees/eid\} \rightarrow employees/ename)$ be an XFD on D in Fig 1. If we *expand*($[eid \times ename], emp$) on D , then the XFD becomes $\bar{\Phi}'(company/branch, \{employees/emp/eid\} \rightarrow employees/emp/ename)$ where the new element *emp* is added in middle of both P and Q as $\bar{P} = employees/emp/eid$ and $\bar{Q} = employees/emp/ename$. This case is straight forward. Now consider another XFD $\Phi''(company/branch, \{employees/eid\} \rightarrow bname)$ on D . If we want to *expand*($[bname \times employees], brinfo$) on D , then there are two options to transform the XFD. As *brinfo* is between *last*(S) and both of *beg*(P), *beg*(Q), we have an option to add *brinfo* to the end of S and an option to add *brinfo* to the front of both P and Q . With option 1, $\bar{\Phi}''(company/branch/brinfo, \{employees/eid\} \rightarrow bname)$ and with option 2, $\bar{\Phi}''(company/branch, \{brinfo/employees/eid\} \rightarrow brinfo/bname)$. We note that *brinfo* needs to be considered in one of two options. Otherwise, the transformed XFD is not valid. Consider the XFD $\Phi'''(company/branch, \{employees\} \rightarrow bname)$ on D . If we want to *expand*($[eid \times ename], emp$) on D , then the new element *emp* is added between *last*(P) and its children. Again there are two

options similar to the motivation in the introduction. One option is to extend the path P to include the new element. The other is to keep the path P unchanged (We note that even if it is unchanged, it is valid on \bar{D} in this case).

We observe that the transformation of Φ' is straightforward. The transformation of Φ'' introduces the choice of two options. For Φ''' , we have options whether we should transform the path or we should keep the paths unchanged. We formalize the three cases.

Let $g_d \in \beta(e_{k-1}) \wedge e_k \in g_d$ and $\tau = \text{expand}(g_d, e_{new})$. Also, consider $w = e_1/\dots/e_{k-1}/e_k/e_{k+1}/\dots/e_m$ where $m \geq 1$ then w is transformed as $\tau(w) = \bar{w} = e_1/\dots/e_{k-1}/e_{new}/e_k/e_{k+1}/\dots/e_m$.

We recall that S/P and S/Q are complete paths in an XFD $\Phi(S, P \rightarrow Q)$. So, if a path $S = w = e_1/\dots/e_{k-1}$, then there is a decision problem whether we should add the new element e_{new} at the end of the path S or at the beginning of P and Q . As there is no unique semantics when the options appear, we define the variable *option* to allow the user to determine semantics. *option* = 1 means to add e_{new} to the end of the path and *option* = 2 means add e_{new} to the beginning of the path starting with e_k .

In the same way, if $\exists \varphi \in P \wedge \varphi = w = e_1/\dots/e_{k-1}$, then we can add the new element e_{new} at the end of path φ or we don't do any change to the path φ . Also, if $Q = w = e_1/\dots/e_{k-1}$, then we can add the new element e_{new} at the end of Q or we don't do any change in the path Q .

We now give the transformation on XFD definition using *expand* operator.

Let $w \in (\{S\} \cup P \cup \{Q\})$.

- (a) If $u = w$ and $\text{last}(u) \neq e_{k-1}$, then $\bar{u} = \bar{w}$.
- (b) Else $//(u = w$ and $\text{last}(u) = e_{k-1})$
 - (i) If $S = u$, then
 - (1) If $\forall \varphi \in (P \cup \{Q\})(\text{beg}(\varphi) \in g)$
 If *option* = 1, then $\bar{S} = u/e_{new}$ and $\bar{\varphi} = \varphi$.
 Else $//$ (*option* = 2), $\bar{S} = u$ and $\bar{\varphi} = e_{new}/\varphi$.
 - (2) Else $\bar{S} = u$ and $\forall \varphi \in (P \cup \{Q\}) \wedge \text{beg}(\varphi) = e_k$, then $\bar{\varphi} = e_{new}/\varphi$.
 - (ii) If $\exists \varphi \in (P \cup \{Q\}) \wedge \varphi = u$, then
 If *option* = 1, then $\bar{\varphi} = u/e_{new}$.
 Else (*option* = 2), then $\bar{\varphi} = u$.

Transformation on XFD Using *collapse*. We give here an example how the *collapse* operator transforms the XFD.

Example 15. Let $\Phi'(\text{company}/\text{branch}, \{\text{employees}/\text{eid}\} \rightarrow \text{employees}/\text{ename})$ be an XFD on D in Fig 1. If we want to *collapse(employees)* on D , then the XFD is transformed as $\bar{\Phi}'(\text{company}/\text{branch}, \{\text{eid}\} \rightarrow \text{ename})$ where the element *employees* is deleted from the determinant as $\bar{P} = \text{eid}$ and the dependent as $\bar{Q} = \text{ename}$. Consider $\Phi''(\text{company}/\text{branch}, \{\text{employees}\} \rightarrow \text{bname})$ on D . If we want to *collapse(employees)* on D , then we can't simple delete the element *employees*; otherwise $P = \text{employees}$ becomes empty. We propose to transform the XFD as $\bar{\Phi}''(\text{company}/\text{branch}, \{\text{eid}, \text{ename}\} \rightarrow \text{bname})$ where the elements *eid* and *ename* are the direct child elements of *employees* and are in the definition

$\beta(\text{employees}) = [\text{eid} \times \text{ename}]^+$. We see that the number of paths in the determinant is increased. Now, consider the XFD $\Phi'''(\text{company}/\text{branch}, \{\text{bname}\} \rightarrow \text{employees})$ on D . If we $\text{collapse}(\text{employees})$ on D , then we propose to transform the XFD Φ''' to new XFDs: $\bar{\Phi}_1'''(\text{company}/\text{branch}, \{\text{bname}\} \rightarrow \text{eid})$ and $\bar{\Phi}_2'''(\text{company}/\text{branch}, \{\text{bname}\} \rightarrow \text{ename})$. This is because we allow only one path in the dependent Q . We see that the number of XFDs are increased in this case. Now we formalize the XFD transformation using collapse operator.

If $e_{k+1} \in \beta(e_k)$ and $\tau = \text{collapse}(e_k)$, then any path $w = e_1/\dots/e_{k-1}/e_k/e_{k+1}/\dots/e_m$ where $k - 1 \geq 1$ is transformed as $\tau(w) \rightarrow \bar{w} = e_1/\dots/e_{k-1}/e_{k+1}/\dots/e_m$. We note that $\text{collapse}(e_k)$ is not applicable if $\beta(e_k) = \text{Str}$ and $S = \rho(\text{root})$.

- (i) If $S = w \wedge e_k \in w$, then $\bar{S} = \bar{w}$.
- (ii) If $\wp \in P \wedge \wp = w \wedge e_k \in w$ then,
 - (1) If $\text{len}(w) > 1$, then
 - (a) If $\text{last}(w) \neq e_k$ then $\bar{\wp} = \bar{w}$.
 - (b) Else($\text{last}(w) = e_k$), $\bar{\wp} = \{\bar{w}_i\}$ where $\bar{w}_i = e_1/\dots/e_{k-1}/e_{k+1}/\dots/e_m$ and $e_{k+1} \in \beta(e_k)$.
 - (2) Else($\text{len}(w) = 1$), $\bar{\wp} = \bar{w} = \{\bar{\wp}_1, \dots, \bar{\wp}_i, \dots, \bar{\wp}_r\}$ where $i \in [1, \dots, r] \wedge \bar{\wp}_i = e_i$ and $\forall i(e_i \in \beta(e_k))$.
- (iii) If $Q = w \wedge e_k \in w$ then
 - (1) If $\text{len}(w) > 1$, then
 - (a) If $\text{last}(w) \neq e_k$, then $\bar{Q} = \bar{w}$.
 - (b) Else($\text{last}(w) = e_k$), $\bar{\Phi}_i(S, P \rightarrow w_i)$ where $w_i = e_1/\dots/e_{k-1}/e_{k+1}/\dots/e_m$ and $e_{k+1} \in \beta(e_k)$.
 - (2) Else($\text{len}(w) = 1$), we need to generate XFDs as $\bar{\Phi}_i(S, P \rightarrow w_i)$ where $w_i = e_i$ and $\forall i(e_i \in \beta(e_k))$.
- (iv) Let $\exists \wp_p \in P \wedge \wp_p = w_p \wedge Q = w_q$. If $e_k \in w_p \wedge e_k \in w_q$, then $\bar{\wp}_p = \bar{w}_p$ using case (ii) and $\bar{Q} = \bar{\wp}_q$ using case (iii) of the collapse operation on XFD.

Observation 3. *Given an XFD $\Phi(S, P \rightarrow Q)$, the transformation operator can transform S , or \wp where $\exists \wp \in P$, or Q or, both \wp, Q together.*

Theorem 1 (Validity of Transformed XFD on \bar{D}). *Let τ be the transformation such that $\tau(D, T, \Phi) \rightarrow (\bar{D}, \bar{T}, \bar{\Phi})$. Then $\bar{\Phi}$ is valid on \bar{D} .*

5 Preservation of XML Functional Dependency

In this section, we study how the transformation of XFDs described in the previous section affects the satisfactions. More specifically, given $\tau(D, T, \Phi) \rightarrow (\bar{D}, \bar{T}, \bar{\Phi})$, $\Phi \sqsubset D$, $T \in D$, $T \prec \Phi$, $\bar{T} \in \bar{D}$, and $\bar{\Phi} \sqsubset \bar{D}$, we investigate whether \bar{T} satisfies $\bar{\Phi}$, denoted as $\bar{T} \prec \bar{\Phi}$.

Definition 14. *Given $\tau(D, T, \Phi) \rightarrow (\bar{D}, \bar{T}, \bar{\Phi})$, $\Phi \sqsubset D$, $T \in D$, $\bar{T} \in \bar{D}$, and $\bar{\Phi} \sqsubset \bar{D}$, if $T \prec \Phi$ and $\bar{T} \prec \bar{\Phi}$, we say that Φ is preserved by the transformations. \square*

5.1 XFD Preserving Properties of the Operators

We recall the satisfaction of the XFD $\Phi(S, P \rightarrow Q)$ where for every pair of complete P, Q tuples in T^S , if $F_1[P] =_v F_2[P]$, then $F_1[Q] =_v F_2[Q]$. The definition of XFD satisfaction requires that when XFD preservation is studied, our focus is to see (a) whether T^S itself and $\langle T^S \rangle$ are changed by the transformation, (b) how the P-tuple $F[P]$ itself is changed by the transformation, (c) how the Q-tuple $F[Q]$ itself is changed by the transformation, (d) whether new $F[P]$, or $F[Q]$ is produced by the transformation, (e) whether $\langle F[P] \rangle$ or $\langle F[Q] \rangle$ is changed by the transformation. We now give a lemma relating to DTD and document transformation and the proof of the lemma is straightforward from the definitions.

Lemma 1. *Let $\tau(D, T, \Phi) \rightarrow (\bar{D}, \bar{T}, \bar{\Phi})$ be the transformation and $T \prec \Phi$. If $\tau(D) = D$, $\tau(T) = T$, and $\tau(\Phi) = \bar{\Phi}$, then $\bar{\Phi}$ is preserved.*

We now show the XFD preservation property of each transformation operators.

Theorem 2. *The expand operator is XFD preserving if any path u in Φ is transformed as $\bar{u} = u/e_{new}$ then the component g_d with its multiplicity c as g_d^c is pushed down one step from the root using the new element e_{new} .*

Proof. We consider the following cases.

Case 1 $[w \in (\{S\} \cup \{Q\}) \cup P \wedge u = w \wedge last(u) \neq e_{k-1}]$. Consider the path $w = e_1/\dots/e_{k-1}/e_k/e_{k+1}/\dots/e_m$. In this case, $\bar{u} = \bar{w} = e_1/\dots/e_{k-1}/e_{new}/e_k/e_{k+1}/\dots/e_m$. So, $last(u) = last(\bar{u}) = e_m$. As a result, the tree $T^u = T^{\bar{u}}$ in the transformed document \bar{T} . Thus if $T \prec \Phi$, then $\bar{T} \prec \bar{\Phi}$.

Case 2 $[S = u \wedge last(u) = e_{k-1}]$. We have the following subcases.

Subcase 1 $(\forall \wp \in (P \cup \{Q\})(beg(\wp) \in g))$. In this subcase, we have two options.

Option1: $\bar{S} = u/e_{new}$ meaning that scope is transformed and $\forall \wp \in (P \cup \{Q\})(\bar{\wp} = \wp)$. Though the P-tuples and Q-tuples are not changed, but the tree rooted at scope $T^u = T^{e_{k-1}}$ is transformed to $T^{\bar{S}} = T^{e_{new}}$ in \bar{T} . But as we push down the whole component g_d^c , $\langle T^S \rangle = \langle T^{\bar{S}} \rangle$ in \bar{T} . So XFD is preserved.

Option2: $\bar{S} = u$ meaning that scope is not transformed, but $\forall \wp \in (P \cup \{Q\})(\bar{\wp} = e_{new}/\wp)$. However, $\forall \wp \in (P \cup \{Q\})(last(\bar{\wp}) = last(\wp))$ and thus $\forall \wp \in (P \cup \{Q\})(T^{\bar{\wp}} = T^{\wp})$. So the P-tuples and Q-tuples are not changed in \bar{T} and XFD is preserved.

Subcase 2 $(\exists \wp \in (P \cup \{Q\})(beg(\wp) \notin g))$. In this subcase, $\bar{S} = u$ meaning that scope is not transformed, but $\forall \wp \in (P \cup \{Q\}) \wedge beg(\wp) = e_k \wedge e_k \in g(\bar{\wp} = e_{new}/\wp)$. However, $\forall \wp \in (P \cup \{Q\})(last(\bar{\wp}) = last(\wp))$ and thus $\forall \wp \in (P \cup \{Q\})(T^{\bar{\wp}} = T^{\wp})$. So, the P-tuples $F[P] = (T^P)$ and Q-tuples $F[Q] = (T^Q)$ are not changed in \bar{T} and XFD is preserved.

Case 3 $[\exists \wp \in (P \cup \{Q\}) \wedge \wp = u \wedge last(u) = e_{k-1}]$. Here, we have two options.

Option1: $\bar{\wp} = u/e_{new}$ meaning that we can transform the path of determinant u adding e_{new} at the end. So $last(u) = e_{k-1}$ and thus $T^u = T^{e_{k-1}}$. But $last(\bar{u}) = e_{new}$ and thus $T^{\bar{u}} = T^{e_{new}}$. So the P-tuple $F[P] = (T^u)$ is transformed to $F[\bar{P}] = (T^{\bar{u}})$ in \bar{T} . But the number of P-tuples is not changed as we expand

the whole component g_d^c . Thus the XFD is preserved.

Option2: $\bar{\varphi} = u$ meaning that we can keep the path of determinant unchanged. Let $\beta(u) = [e_1 \times e_2]^+$. So the P-tuple as $F[P] = (T^u) = (v_1 : last(u)(v_2 : e_1)(v_3 : e_2)(v_4 : e_3)(v_5 : e_4))$ is changed as $F[P] = (T^u) = (v_1 : last(u)(v_2 : e_{new}(v_3 : e_1)(v_4 : e_2))(v_5 : e_{new}(v_6 : e_3)(v_7 : e_4)))$ if we $expand([e_1 \times e_2], e_{new})$ or $F[P] = (T^u) = (v_1 : last(u)(v_2 : e_{new}(v_3 : e_1)(v_4 : e_2)(v_5 : e_3)(v_6 : e_4)))$ if we $expand([e_1 \times e_2]^+, e_{new})$. However, the number of P-tuples is not changed and the transformed P-tuples don't violate the XFD satisfaction in \bar{T} .

Theorem 3. *The collapse operator is XFD preserving if $last(S) \neq e_k$ and $last(P) \neq e_k$.*

Proof. We consider the following cases.

Case 1 [$w = S \wedge e_k \in w$]. Let $w = e_1/\dots/e_{k-1}/e_k/e_{k+1}/\dots/e_m$. In this case, $\bar{w} = e_1/\dots/e_{k-1}/e_{k+1}/\dots/e_m$. So, if $k \neq m$ meaning that $last(S) \neq e_k$ then $last(w) = last(\bar{w}) = e_k$. So, $T^w = T^{\bar{w}} = T^{e_k}$ and the number of T^w is not changed. But if $k = m$ meaning that $last(S) = e_k$, then $last(\bar{w}) = e_{k-1}$ and thus $T^{\bar{w}} = T^{e_{k-1}}$. So the number of T^w can be decreased. In $T^{\bar{w}}$, there may have P-tuples and Q-tuples that may violate the XFD satisfaction.

Case 2 [$\varnothing \in P \wedge w = \varnothing \wedge e_k \in w$]. In this case, we have two subcases.

Subcase 1 ($len(w) > 1$).

(a) Let $w = e_1/\dots/e_{k-1}/e_k/e_{k+1}/\dots/e_m$. Then $\bar{w} = e_1/\dots/e_{k-1}/e_{k+1}/\dots/e_m$. So, if $last(w) \neq e_k$ then $last(w) = last(\bar{w})$. Thus, $T^w = T^{\bar{w}}$. So the P-tuples are not changed in this case.

(b) But If $last(w) = e_k$, then $\bar{w} = \{\bar{w}_i\}$ where $\bar{w}_i = e_1/\dots/e_{k-1}/e_{k+1}/\dots/e_m$ and $e_{k+1} \in \beta(e_k)$. As new paths are generated in the determinant, the P-tuple is changed and the number of P-tuples are also changed.

Subcase 2 ($len(w) = 1$). $\bar{w} = \{\bar{\varphi}_1, \dots, \bar{\varphi}_i \dots, \bar{\varphi}_r\}$ where $i \in [1, \dots, r] \wedge \bar{\varphi}_i = e_i$ and $\forall i (e_i \in \beta(e_k))$. As new paths are generated as determinant, the P-tuple is changed and the number of P-tuples are also changed.

Case 3 [$w = Q \wedge e_k \in w$]. In this case, we have two subcases.

Subcase 1 ($len(w) > 1$).

(a) Let $w = e_1/\dots/e_{k-1}/e_k/e_{k+1}/\dots/e_m$. Then $\bar{w} = e_1/\dots/e_{k-1}/e_{k+1}/\dots/e_m$. So, if $last(w) \neq e_k$ then $last(w) = last(\bar{w})$. Thus, $T^w = T^{\bar{w}}$. So the Q-tuples are not changed.

(b) But If $last(w) = e_k$, then $\Phi_i(S, P \rightarrow w_i)$ where $w_i = e_1/\dots/e_{k-1}/e_{k+1}/\dots/e_m$ and $e_{k+1} \in \beta(e_k)$. The new XFDs are also satisfied by the transformed document.

Subcase 2 ($len(w) = 1$). New XFDs are generated as $\Phi_i(S, P \rightarrow w_i)$ where $w_i = e_i$ and $\forall i (e_i \in \beta(e_k))$. The new XFDs are also satisfied by the transformed document.

Case 4 [$\exists \varphi_p \in P \wedge w_p = \varphi_p \wedge w_q = Q$]. If $e_k \in w_p \wedge e_k \in w_q$. The proof of this case follows the case 2 and case 3.

Example 16. Let $\Phi_d(company, \{branch/employees\} \rightarrow branch/bname)$ be an XFD on D in Fig 1 which is satisfied by T in Fig 2. Using $collapse(employees)$

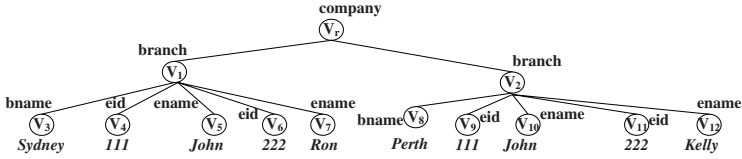


Fig. 5. XML Tree \bar{T} using $collapse(employees)$

on D , the transformed XFD is $\bar{\Phi}_d(company, \{branch/eid, branch/ename\} \rightarrow bname)$. Thus, the P-tuple $F[P] = (T^{employees})$ is transformed to $F[\bar{P}] = (T^{eid}T^{ename})$ in \bar{T} in Fig. 5. We see that in $T^{company}$, $F_1[\bar{P}] = ((eid : 111)(ename : John))$ and $F_2[\bar{P}] = ((eid : 111)(ename : John))$ are value equal but $F_1[Q] = ((bname : Sydney))$ and $F_2[Q] = ((bname : Perth))$ are not value equal in \bar{T} in Fig. 5. So \bar{T} doesn't satisfy $\bar{\Phi}_d$ and thus the XFD $\bar{\Phi}_d$ is not preserved.

Theorem 4. *The nest and unnest operators are XFD preserving.*

Proof sketch: Firstly, The nest and unnest operators don't transform the XFD definition. Secondly, the nest operator merges the duplicated trees (with same values) and hence reduces some P-tuples or Q-tuples without violating XFD satisfaction in the transformed document. Thirdly, the unnest operator makes the nested structure to a flat structure and hence introduces some duplicated P-tuples or Q-tuples without violating XFD satisfaction in the transformed document. The full length of the proofs of the operators can be found in [13].

6 Conclusions

We studied the XFD preserving transformation of XML data. Towards this task, we first defined the XFD over the DTD and its satisfaction by XML document using a novel concept of producing tuples. We then studied the transformation of XFDs using major transformation operators with validity of transformed XFDs. We also studied the XFD preservation of the operators. We found that the important transformation operations are XFD preserving with necessary and sufficient conditions. Our research in this paper is towards the task of XML data integration with constraints preservation.

References

1. Cali, A., Calvanese, D., Giacomo, G.D., Lenzerini, M.: Data Integration under Integrity Constraints. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 262–279. Springer, Heidelberg (2002)
2. Zamboulis, L., Poulouvassilis, A.: Using Automated for XML Data Transformation and Integration. DIWeb, 58–69 (2004)
3. Erwig, M.: Toward the Automatic Derivation of XML Transformations. ER, 342–354 (2003)

4. Liu, J., Park, H., Vincent, M., Liu, C.: A Formalism of XML Restructuring Operations. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 126–132. Springer, Heidelberg (2006)
5. Liu, J., Vincent, M.W.: Functional Dependencies for XML. In: Zhou, X., Zhang, Y., Orłowska, M.E. (eds.) APWeb 2003. LNCS, vol. 2642, pp. 22–34. Springer, Heidelberg (2003)
6. Vincent, M.W., Liu, J., Mohania, M.: On the equivalence between FDs in XML and FDs in relations. *Acta Informatica*, 207–247 (2007)
7. Kolahi, S.: Dependency-preserving normalization of relational and XML data. *Journal of Computer and System Sciences* 73(4), 636–647 (2007)
8. Arenas, M., Libkin, L.: A Normal Form for XML documents. *ACM PODS*, 85–96 (2002)
9. Fan, W.: XML Constraints: Specification, Analysis, and Applications. *DEXA*, 805–809 (2005)
10. Fan, W., Simeon, J.: Integrity constraints for XML. *PODS*, 23–34 (2000)
11. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0., World Wide Web Consortium (W3C) (February 1998)
12. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures, W3C Working Draft (April 2000)
13. Shahriar, M.S., Liu, J.: Towards the Preservation of Functional Dependency in XML Data Transformation, extended version,
<http://www.cis.unisa.edu.au/~cisj1/publications/publ.html>

Enabling XPath Optional Axes Cardinality Estimation Using Path Synopses*

Yury Soldak and Maxim Lukichev

Department of Computer Science, University of Saint-Petersburg, Russian Federation
ysoldak@acm.org, maxim.lukichev@gmail.com

Abstract. The effective support for XML query languages is becoming increasingly important with the emergence of new applications that access large volumes of XML data. The efficient query execution, especially in the distributed case, requires estimating of the path expression cardinalities. In this paper, we propose two novel techniques for the cardinality estimation of the simple path expressions with optional axes (following/preceding): the document order grouping (DG) and the neighborhood grouping (NG). Both techniques summarize the structure of source XML data in compact graph structures (path synopses) and use these summaries for cardinality estimation. We experimentally evaluated accuracy of the techniques, size of the summaries and studied performance of the prototypes. The wide range of source data was used in order to study the behavior of the structures and the area of techniques application.

1 Introduction

The effective support for XML query languages is becoming increasingly important with the emergence of new applications that access large volumes of XML data especially in the case this data is distributed. All of existing proposals for querying XML (e.g. XQuery) rely on the pattern specification languages that allows (1) a path navigation and branching through the label structure of the XML data graph, and (2) predicates on the values of the specific path/branch nodes, in order to reach the desired data elements.

The optimization of such queries requires approximation of the result selectivity of the referenced paths and hence hinges on the existence of concise synopsis structures that enable accurate compile time selectivity and/or cardinality estimations for path expressions over XML data.

As a result, many papers in the area of efficient XML query evaluation are dedicated to approaches of collecting, storing and using statistics about data stored in XML DBMS.

In [1] authors showed that an XML query optimizer, which takes advantage of statistical structures and optimization algorithms enabled by them, can achieve performance of query evaluation comparable to those achieved by state-of-the-art

* Research supported by the Hewlett-Packard Labs.

indexing techniques for XML documents. The advantage is in the reduced cost of creation a path summary (which can be done with essentially one pass over the document), compared with indexing techniques that require a substantially larger amount of computation to generate the indexes.

Enabling the cost estimation for XML queries is more complex than for the relational case, due to the semantic differences and the complexity of the operators, as well as the versatility and the complexity of XML data; for example, the hierarchical data model, flexibility of the document schema and predicate constraints on the structure and values.

In this paper, we propose two novel techniques based on path-trees for the cardinality estimation of the simple path expressions with optional axes (following/preceding): the document order grouping (DG) and the neighborhood grouping (NG). Both techniques summarize the structure of source XML data in compact graph structures (path synopses) and use these summaries for the cardinality estimation.

The paper is organized as follows. In Section 2 we describe the problem area and set the goals. Then Section 3 outlines some general observations about optional axes. The estimation techniques are described in Section 4 and Section 5 discusses their experimental evaluation. Section 6 surveys and classifies related works. Finally, Section 7 contains conclusions.

2 Problem Definition

The XPath specification defines thirteen axes to use in path expressions. The "ancestor", "descendant", "following", "preceding" and "self" axes partition a document: they do not overlap and together they cover all document elements. The first two axes usually referenced as *base* and the second two as *optional*.

The base axes, together with particular cases like *child* and *parent*, were actively studied in papers, while the optional axes received little attention from the community. Particularly, cardinality estimation of path expressions with optional axes has not been previously studied and XML query optimizers, therefore, have no good methods for estimation of such path expressions.

We believe that little attention to these axes is due to two reasons. First, optional axes are originally intended to enable usage of the document order in queries, but XML is used today mainly as data-oriented format, so document order is not significant there. An example is the DBLP data set where order in which articles appear in a file is not significant. Second, the field of XML query optimization is quite new and community just doesn't cover all the gaps yet, investing efforts to cover the base cases at first.

Here we concentrate on the case of optional axes. The primary goal is cardinality estimation techniques for simple (i.e. without predicates) path expressions with optional axes. This includes the development of statistical structures to store all necessary information and actual estimation algorithms which take advantage of these structures. The secondary goal is to also support base axes using our developed statistical structures while keeping average estimation errors for these axes comparable with existing techniques (see Section 6).

3 General Observations about Optional Axes

There are four optional axes defined in the XPath specification: two major cases and two minor cases. We refer to the "preceding" and "following" cases as *major optional axes*, "preceding-sibling" and "following-sibling" as *minor optional axes* in this paper. Also we define the axis "preceding-sibling" ("following-sibling") as *corresponding* to "preceding" ("following"). Without loss of generality and for simplicity reasons, we are using "following" and "following-sibling" axes exclusively in examples, assuming that "preceding" and "preceding-sibling" cases will be similar.

A useful observation about major optional axes is that the result of evaluating a path expression with such an axis is equal to the united result obtained from specially constructed set of path expressions without any major optional axis. These generated path expressions exploit the minor optional axes instead and are constructed in such a way that together they cover all of document elements matched by the original path expression.

For example, path expression $A/B/C/following::D$ reduces to two paths:

1. $A/B/following-sibling::* / descendant-or-self::D$
2. $A/B/C/following-sibling::* / descendant-or-self::D$

Thus, the cardinality estimation of a path expression with major optional axes reduces to the estimation of a set of path expressions with the minor optional axes. The cardinality of the original path expression equals the sum of cardinalities of the path expressions from the set.

The following is the formal description of cardinality estimation algorithm.

Having a *path* = $[s_1, \dots, s_M, \dots, s_N]$, where s_i is i 'th step and M is an index of the step with major axis to reduce. Let $s_i = a_i :: n_i$, where a_i – axis of i 'th step, n_i – name test of i 'th step. Thus, $s_M = a_M :: n_M$.

Then,

$$cardinality(path) = \sum_{i=2}^{M-1} cardinality([s_1, \dots, s_i, a'_M :: *, d-o-s :: n_M, s_{M+1}, \dots, s_n])$$

where a'_M – the minor optional axis corresponding to a_M , $d-o-s$ is the shortcut to *descendant-or-self*.

As a result, it is sufficient to support only the minor optional axes in the structures to be developed in order to cover all the cases.

4 Cardinality Estimation Techniques

The idea of representation of statistics about semistructured data as a graph structures (also called "summaries" or "path synopses") is well-known [2]. For instance, the Path-tree is the summary which stores an information about simple path expressions for XML data (it is a tree of all possible "top-down" paths in a document). Each path-tree node corresponds to a set of document elements

reachable by particular path, and vice versa each document element corresponds to exact one Path-tree node. Thus, the document elements are *grouped* to the equivalence sets according to the following equivalence rule: any two elements belong to the same set (group) if they (1) have the same name and (2) their parents belong to some other single set (group).

Both structures proposed in this paper are built using the Path-tree as a prototype. They differ from the Path-tree (and from each other) mainly in the way they group the document elements. Particularly, each structure defines its own third property the document elements should satisfy to be considered equivalent and contribute to the same group. Naturally, this leads to larger structures in the terms of number of nodes. Another feature to be discussed is additional edges proposed to store statistics about the optional axes.

Further in this section we introduce two statistical structures, based on Neighborhood (NG) and Docorder (DG) grouping strategies as well as two algorithms to estimate cardinality of path expressions using the structures.

4.1 Neighborhood Grouping Technique

The structure for this technique is constructed using specially designed *neighborhood* grouping method. The idea behind this method is that elements with the same set of child element names expected to have the same number of siblings (in either direction) with different names. In other words, the variety of the child elements implicitly defines a type and possibly semantics of their parent element. And the elements with the same type/semantics tend to share the same properties, mainly in respect of the relative position to their siblings, number of the siblings in either direction, etc. And this is important for the optional axes estimation.

Definition 1. *The document elements contribute to the same **neighborhood group** if they share the following properties:*

1. *have the same node name*
2. *their parent elements contribute to a single neighborhood group (distinct from current though)*
3. *have the same set of distinct child node names*

The structure should be constructed in the top-down fashion, starting from the root document element.

All of the "C" elements in Figure [□](#) do not contribute to the same group (structure node) since the third property is violated. The leftmost "C" element contributes to the leftmost group in resulting structure while other two "C" elements contribute to the rightmost one. The set of child nodes for the each node is unordered in the Neighborhood structure, in contrast to the Docorder structure (discussed later) where node order is preserved.

The child nodes are connected with their parent nodes using the *child* edges (solid at the figure). This is the same as in the Path-tree structure. The Neighborhood structure has extra edges to maintain the document order information

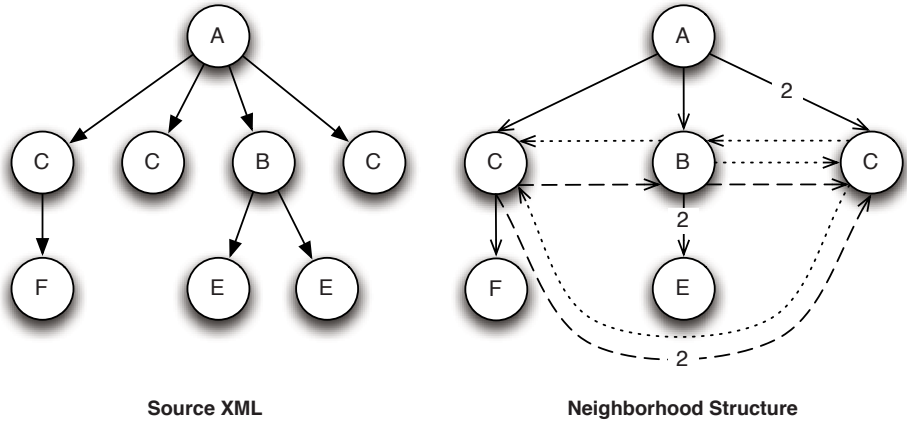


Fig. 1. Neighborhood Structure (Grouping Method, Minor Edges and Edge Labeling)

which gets lost during the grouping. We reference to these edges as *minor edges*, since they define the same relation among the structure nodes as the minor optional axes among the document elements. So, Neighborhood structure has the following and the preceding minor edges which can be used to estimate the cardinality of the "following-sibling" and the "preceding-sibling" XPath axes respectively.

In Figure 1 the preceding minor edges are shown using short dashes, the following ones using long dashes and child edges are solid. Every edge has the cardinality. The method of assigning the cardinalities to the structure edges is exactly the same as in the Path-tree case. The edge cardinality equals the exact number of the document elements contributing to the edge's destination node and being in the respective (to the king of the edge) relation with the elements contributing to the edge's source node. The minor edges with zero cardinality are simply omitted in Figure 1. In case the cardinality equals "1" then the cardinality is not shown at the figure for that edge. The single minor edge with the cardinality greater than "1" connects two "C" nodes and has cardinality "2".

The cardinality estimation algorithm is simple and resembles the standard XPath evaluation algorithm. The first difference is that we apply the standard algorithm to the Neighborhood structure instead of an XML document. The second difference is that we consider the edges instead of the nodes as being the items to push between the algorithm iterations. So, having the set of edges at the last iteration we just summarize their cardinalities to get the estimation for the target path expression.

This estimation algorithm suffers from an issue though. The issue can be named *cardinality duplication* and discussed further.

The source of the issue is that we actually can't simply summarize cardinalities of the edges if any of minor optional edges was examined during evaluation.

Figure 2 presents three XML fragments which demonstrate three different situations of this duplication issue. The elements with the same name and shape contribute to the same structure node. (Please note, figure shows the XML document fragments only, so the document elements with the same name but different shape just have different children not shown at the figure and will not contribute to the same group.)

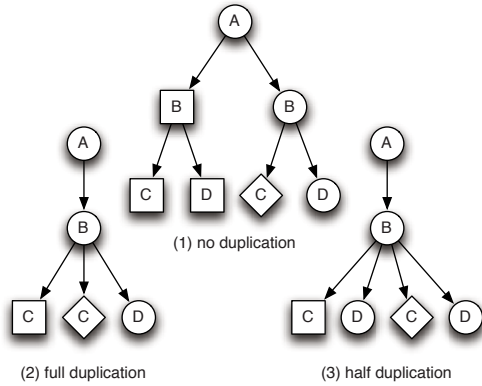


Fig. 2. Cardinality Duplication

Assume that we have the following path to estimate cardinality: $A/B/C/$ following-sibling:: D . It will be no duplication if the source document resembles the first case, while in the second case the single element "D" will be counted twice, since it has same relation with each "C" element. In the third case the last "D" element is counted twice, but not the first one.

We worked this issue around by means of tweaking the structure a little. Namely, the minor edge cardinalities should be computed a little differently.

Having set of sibling neighborhood nodes with the same name ($a_i \in A$) and any other sibling neighborhood node not in this set ($b \notin A$). Let e_b be any document element contributing to b ; e_L is the nearest left document element to the e_b (among all document elements contributing to $a_i \forall i$); e_R is the nearest right element respectively. Obviously e_L contributes to a single a_i , the same is for e_R .

The single following minor edge from the nodes in A to the b which counts e_b is the edge for which e_L contributes to the source node. All other following minor edges from the nodes in A to the b should simply avoid considering e_b while computing cardinality. The same is for preceding minor edges and e_R .

This edge cardinality computing trick completely eliminates the cardinality duplication, while preserving cardinality estimation algorithm untouched.

4.2 Docorder Grouping Technique

This technique is heavily based on the document order of a source document, hence the name. The structure tries to resemble source XML as close as possible

in order to enable extremely accurate estimation. The grouping method puts document element into one group if they form continuous chunk. More formally:

Definition 2. *The document elements contribute to the same **docorder group** if they share the following properties:*

1. *have the same node name*
2. *their parent elements contribute to a single docorder group (distinct from current though)*
3. *there are no sibling elements with different name between them*

The groups (structure nodes) are ordered according to the document order of a source document. This enables the minor optional axes cardinality estimation without extra minor edges (as in case of NG). Instead, two special boolean flags are used to mark sibling boundaries and to help the minor optional axes cardinality estimation. A node has the *first* flag set if it contains an element which is the first among all its siblings, similarly the *last* flag is set to the nodes with the last sibling elements. The Path-tree edge labeling approach is used to label edges in Docorder (DG) structure.

Figure 3 shows an example of the grouping method, sibling flags and the edge labeling used in DG structure.

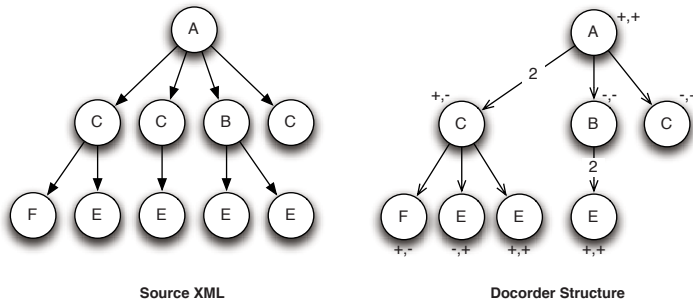


Fig. 3. Docorder Structure (Grouping Method, Sibling Flags and Edge Labeling)

The first two "C" elements of the example contribute to the leftmost group and the the last "C" contributes to the separate rightmost group. The element "B" is placed between the second and the third "C" elements preventing all "C" elements contribute to a single group.

The cardinality estimation algorithm resembles the standard XPath evaluation algorithm. The single difference is that it takes into account the sibling flags set at the structure nodes.

The sibling flags are represented as +/- pairs at the figure. This feature works like following: `cardinality(A/C/F/following-sibling::E)` will be estimated as 1, since the leftmost "E" node has *last* flag set, so evaluation stops here and the next "E" node is not considered.

4.3 Wildcard Path Expressions

This section discusses the method of estimating wildcard path expressions. The wildcard path expression is the path expression with at least one wildcard step like "child:*" or "following-sibling:*". The wildcard steps with the base axes are estimated without any problem using techniques discussed above while the optional axes case (*optional wildcard step*) is tricky.

The wildcard expressions can come directly from the user query or indirectly from the major optional axes reduction algorithm discussed in Section 3.

The source of the issue is that the minor optional wildcard step matches all-except-one elements contributing to a context node. This forces the cardinality estimation algorithm to "dive" into the subgraph of the context node in order to produce accurate estimation. Unfortunately, both proposed structures do not provide any good way to "dive" into node subgraph if not all elements are matched in the node.

We exploit unified distribution assumption to evaluate such path expressions. The justification for this is that we expect the elements contributed to the structure node to be very similar in terms of size and shape of their subtrees.

The formal algorithm is following. Let $fullpath = [path, tailpath]$; $path = [s_1, \dots, s_M]$; $tailpath = [following-sibling::n_M, \dots, s_N]$; $parentpath = [s_1, \dots, s_{M-1}]$; $a_i = "child" \forall i$ where a_i is the axis for s_i step.

Then

$$multiplier(path) = \frac{cardinality(path) - cardinality(parentpath)}{cardinality(path)}$$

$$estimation(fullpath) = cardinality(fullpath) * multiplier(path)$$

The multiplier adjusts the cardinality harvested from the subgraph (reached by the $path$). Obviously, the estimation is not always accurate and is the source of errors as shown in the experiments section.

5 Experiments

This section discusses the experimental evaluation of the described techniques. Both techniques were prototyped in order to study their behavior in general, catch flaws not visible from the theoretical point of view at first and conduct experiments to check applicability of the techniques to the real data. The datasets used in the experiments are described in the next subsection, followed by discussion of three experiments made.

5.1 Datasets

Three types of XML data were chosen to evaluate the cardinality estimation techniques represented by three datasets:

1. XHTML pages (W3C recommendations) [34], as an example of document-oriented, irregular XML data

2. University courses description (UW XML Repository) [5], as an example of super-regular XML data
3. Shakespeare plays in XML by Jon Bosak [6], as kind of compromise between first two

Table 1 shows dataset document sizes statistics.

Table 1. Absolute Sizes of Documents in the Datasets (in KB)

	min	avg	max
xhtml	72	132.00	192
shakespeare	136	204.65	276
courses	277	1389.66	2283

It is expected from the techniques to show bad results on document-oriented data (like XHTML). The point to consider this data set is to understand how bad they do. In turn, super-regular data is not the target for the developed techniques, we anticipate, however, best results in terms of accuracy on such data. Most interesting case is the Shakespeare plays – these documents are regular enough to let the structures compress information effectively, while some irregularity exists and the document order is significant in this data to make usage of the optional axes sensible.

5.2 Size

This experiment was conducted in order to understand the rates at which different grouping strategies compress information for three types of source data.

The Docorder and Neighborhood structures were constructed for each document in each data set. The number of structure nodes then was divided by the number of elements in a source document in order to get the relative size of the structure. Table 2 shows relative average size of structures per grouping method for each dataset.

Table 2. Relative Size of Structures per Data Set in Terms of Nodes

	DG, %	NG, %
xhtml	66.88	20.00
shakespeare	41.33	0.75
courses	93.75	0.08

The very first observation: The DG structure compresses the information a lot worse than the NG structure. Indeed, the number of nodes in DG is comparable with the number of elements in a source document. The DG approach ignores CDATA and text values, of course, still it expected to require a lot of room to store.

Obvious trend is that the NG structure needs less room as regularity of a source document increases. Big size of the NG on the XHTML dataset can be easily explained as the result of the very irregular structure of "ordinary" text, which clearly prevents document elements from contribute to the same group since the number and the kind of the child elements differ heavily in such document. At the same time, the NG structure quickly shrinks as fast as any regularity appears, since the grouping rule start working.

The regularity is not so important for the DG approach. The main factor which decreases the structure size is presence of contiguous element sequences with the same name, which is demonstrated by the Shakespeare dataset there every SPEECH element contains tens of LINE elements in contiguous sequences.

5.3 Accuracy

The point of this experiment is to define (average) expected deviation of approximated value from the real one achieved by techniques on different types of XML data.

We studied four types of path expressions: (1) minor-at-tail ($Minor_t$), (2) minor-at-middle ($Minor_m$), (3) major-at-tail ($Major_t$), (4) major-at-middle ($Major_m$). Here *minor* and *major* references to the type of optional axis used in a path expression, *tail* and *middle* are about place of this axis in a path expression. For each of four path types all possible (for document under evaluation) path expressions were constructed using "child" axis for all the steps except the target step and their relative errors were calculated. All these errors then were grouped by document type and average error is calculated for each dataset and for each path type. The base for the relative error is the real cardinality, so paths with zero real cardinality do not contribute to the result. The results are presented in Table 3.

Table 3. Average Relative Error of Path Estimation (base is real value $\neq 0$)

	xhtml		shakespeare		courses	
	DG, %	NG, %	DG, %	NG, %	DG, %	NG, %
$Minor_t$	0	0	0	0	0	0
$Minor_m$	0	4.94	0	0.39	0	0
$Major_t$	0.21	77.08	0.03	3.25	0	0.02
$Major_m$	0.28	30.56	0	1.08	0	0.06

Extended study was also made for all path expressions, including ones with zero cardinality and the relative error base was the count of elements in source XML document. The results are order of magnitude better, none of the techniques show error $> 0.3\%$ in this case. Please, see Table 4 for the details.

It is easy to see the DG technique is extremely accurate in cardinality estimation, even on fairly irregular data such as XHTML. Small deviations from the real values in the *Major* cases is the result of usage of the uniform distribution assumption.

Table 4. Average Relative Error of Path Estimation (base is number of document elements)

	xhtml		shakespeare		courses	
	DG, %	NG, %	DG, %	NG, %	DG, %	NG, %
$Minor_t$	0	0	0	0	0	0
$Minor_m$	0	0.017	0	0	0	0
$Major_t$	0.003	0.217	0	0.127	0	0
$Major_m$	0.002	0.061	0	0.059	0	0.001

The experimental evaluation of the NG technique approach shows theoretically expected behavior. It achieves almost same accuracy as the DG does on the regular documents, estimates XHTML bad and shows acceptable accuracy at the middle (shakespeare plays). Excellent property of the NG is that it returns exact value for the $Minor_t$ (preceding-sibling/following-sibling at tail) path expressions no matter how irregular source document is. The $Minor_m$ case can be very accurately estimated using the NG also. Both $Major_t$ and $Major_m$ path expression types suffer from the uniform distribution assumption used to produce cardinality estimation for them. The error on the irregular documents is significant, nevertheless $Major_t$ and $Major_m$ path expressions on regular and semi-regular documents can be estimated with the very good accuracy.

And the last observation about the estimation accuracy is that both techniques process the conventional simple path expressions (without predicates and without optional axes) exactly the same way as the Path-tree does due to the fact they both based on the Path-tree, so there is no degradation in accuracy of cardinality estimation for the conventional simple path expressions.

5.4 Performance

This experiment studies the performance of our prototypes. The source data was a shakespeare play which was gradually doubled in size in such a special way both structures double their size. Hence, the statistical structure size to the document size relation remains almost constant. (Actual structure sizes can be estimated using Table 2 in "size" experiment.)

Both prototypes were implemented in Java as in-memory graph structures and not industrial-strong, of course. The results obtained may rather serve as a lower bound to know what performance we can expect from the techniques for sure.

Figure 4 shows the DG and the NG prototypes performance in terms of the milliseconds spent in average for a single path expression cardinality estimation. Please notice the graph has both axes logarithmic. The $Major_t$ and $Major_m$ cases reached near the same performance, so they are presented as the single $Major$ case at the graph in order to make it more readable. The same is for the $Minor$ case.

The results were obtained on a laptop with 2GHz Intel processor, 2Gb RAM.

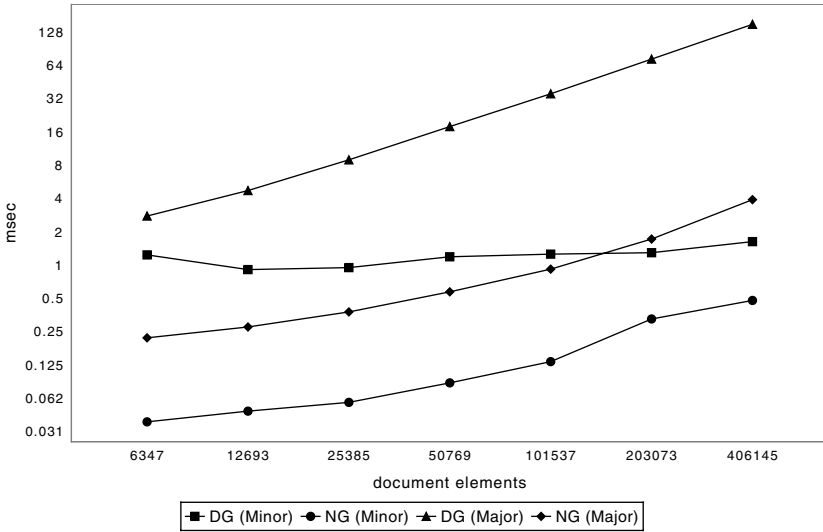


Fig. 4. DG and NG Performance

6 Related Work

Cardinality estimation techniques differ in construction methods, in queries supported and in structure (whatever it is path synopsis or histogram and so on).

6.1 Construction

There are two approaches to construct and maintain statistical structures: online and offline. The offline approach explicitly demands building of a whole structure from scratch for every change in source data implying unavoidable and almost always costly operations. As a result this approach can be used only if source data changes rarely. The online approach is reasonable in cases when data changes quickly or full data scan is too costly or impossible. The Online update process can be triggered by a storage or using a lazy (feedback) approach after user query evaluation. Most of researches in the field of cardinality estimation proposes offline techniques, and only few of them - online. There are two of them in our references list: XPathLearner [7] and CXHist [8]. XPathLearner was the first attempt to use the online method to collect XML statistics. Three years later by the same authors CXHist was proposed as further development of the ideas.

6.2 Expressions Supported

The major part of early works is dedicated to simple path expressions (SPE) only. Then in later works predicates grab the focus, while the following/preceding axes are still untouched. Completely different type of expressions are so-called

twig queries. These expressions describe a complex traversal of the document graph and retrieve document elements through a joint evaluation of multiple path expressions usually highly correlated.

Simple Path Expressions. While Simple path expressions (SPE) notion is heavily used, a lot of meanings exist on that path expressions are simple. In general SPEs do not contain steps with predicates, preceding and following axes. Many early works study SPEs, so for example, the approaches proposed in [9] are quite simple and effective: the Markov table technique, in particular, delivers a high level of accuracy. Unfortunately, they are limited to the simple path expressions (simplest case), and it is not clear how to extend them to twigs or predicates [10].

Twig Queries. A correlation problem arise while dealing with twig queries. It refers to the need of correlating estimations coming from distinct branches of the same twig. In [11] twig queries are estimated using suffix tree constructed for all existing paths. This was the one of the first papers dedicated to the problem of elements correlation.

Originally XSketch [12,13] was not able to estimate twig queries, but in the paper [14] published two years later twig queries support was added based on histograms.

Paths with Predicates. Developments dealing with the predicates are rare. Several cardinality estimation approaches that supports value predicates were proposed in [15] and [8].

6.3 Representation

Researchers and XDBMS developers prefer to store statistics in form of path synopses or histograms, also hybrid approaches in the form of path synopses, enhanced with histograms in leaf nodes, are used sometimes.

Path Synopses. The Path-tree is the classical path synopsis structure, it was introduced in [9]. The Path-tree was defined as a tree of all possible "top-down" paths in a document. Most of subsequent developments utilizing graphs to store statistics about XML documents are direct deviations of the Path-tree approach.

Tagged Region Graph (TRG) was proposed in [10]. It is more powerful and flexible structure than Path-tree, actually Path-tree is the special case of TRG. The regions are denoted by (1) node name, (2) node level and (3) parent node name. There are many possibilities to group nodes into the regions and only some of them are studied (dealing with graphs similar to the Path-trees) for now. TRGs seems to be promising basis for statistical structures for semistructured data.

XSketch [12,13,14] is a graph-based synopsis that tries to capture both the structural and the value characteristics of an XML document. Developers of XSketch consider recursive XML documents where recursion is created with ID-IDREFs.

Histograms. The most interesting approaches of histograms application in XML are XPathLearner [7], Position Histograms [15], XSketch [12][13][16][14], StatiX [17] and Bloom Histograms [18].

In XPathLearner the Markov Histograms [9] are used, where the frequencies of results of traversing all paths of length 2 are stored in two 2-dimensional histograms.

The Position Histograms approach was developed for the cardinality estimation of a single path or an expression with predicates using two dimensional histograms. Such histograms are built for each concrete predicate. So using these histograms for the predicates P1 and P2 we can estimate the cardinality of the expression satisfying the template P1/descendant-or-self:P2.

In XSketch the histograms are used to capture statistical correlations of elements and values in particular the neighborhoods of its graph.

In StatiX the XML Schema is used to identify potential sources of structural skew and then 1-dimensional histograms are built for the most problematic places in the schema, approximating the distributions of parent ids for different elements.

The Bloom Histograms use buckets for paths with similar cardinalities. The corresponding bucket for a path is chosen using Bloom filter.

7 Conclusion

Increasing use of applications that access large volumes of XML data leads to a situation when effective support for XML querying becoming extremely important. It results in evolving of cost-based optimization techniques in XML query evaluation process, that require advanced methods oriented on statistic gathering and estimation of a query result size. The major drawback of currently existing approaches in this field is inability to estimate cardinality of the path expressions with optional axes (e.g. following/preceding).

In order to cover this gap we proposed two novel techniques based on the Path-tree for the cardinality estimation of the simple path expressions with the optional axes: the document-order grouping (DG) and the neighborhood grouping (NG). Both the techniques summarize the structure of source XML data in compact graph structures and use these summaries for cardinality estimation.

The techniques were developed to be applicable for both document-oriented or data-oriented cases. However, usage of them expected to be reasonable mainly in the document-oriented case, since it enables usage of the optional axes in queries.

We experimentally demonstrated accuracy of our techniques, studied size of the structures and evaluated performance of the prototypes. The wide range of source data was used in order to model different use cases. The NG technique shows acceptable accuracy, good performance and structure is very compact. We expect this technique to be basis for the future developments in the area and/or used in query optimization engines. The DG technique is extremely accurate, but suffers from huge structure size. We do not believe this technique to be

used as-is due to the structure size, but it rather may be combined with some synopsis-based indexing techniques to support optional axes.

References

1. Barta, A., Consens, M.P., Mendelzon, A.O.: Benefits of path summaries in an XML query optimizer supporting multiple access methods. In: VLDB 2005: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, pp. 133–144 (2005)
2. Goldman, R., Widom, J.: Enabling query formulation and optimization in semistructured databases. In: VLDB 1997: Proceedings of the 23rd International Conference on Very Large Data Bases, pp. 436–445. Morgan Kaufmann Publishers Inc., San Francisco (1997)
3. W3C HTML Working Group: XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), W3C Recommendation (August 1, 2002), <http://www.w3.org/TR/xhtml1/>
4. W3C XML Working Group: Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation (6 October 2000), <http://www.w3.org/TR/2000/REC-xml-20001006.html>
5. Doan, A.: University of Washington XML Data Repository, University Courses, <http://www.cs.washington.edu/research/xmldatasets/www/repository.html>
6. Bosak, J.: The Plays of Shakespeare, <http://metalab.unc.edu/bosak/xml/eg/shaks200.zip>
7. Lim, L., Wang, M., Padmanabhan, S., Vitter, J., Parr, R.: XPathLearner: An On-line Self-Tuning Markov Histogram for XML Path Selectivity Estimation. In: VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases. Morgan Kaufmann, San Francisco (2002)
8. Lim, L., Wang, M., Vitter, J.S.: Cxhist: an on-line classification-based histogram for Xml string selectivity estimation. In: VLDB 2005: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, pp. 1187–1198 (2005)
9. Aboulnaga, A., Alameldeen, A.R., Naughton, J.F.: Estimating the selectivity of XML path expressions for internet scale applications. The VLDB Journal, 591–600 (2001)
10. Sartiani, C.: A framework for estimating Xml query cardinality. In: Christophides, V., Freire, J. (eds.) International Workshop on Web and Databases, pp. 43–48 (2003)
11. Chen, Z., Jagadish, H.V., Korn, F., Koudas, N., Muthukrishnan, S., Ng, R.T., Srivastava, D.: Counting twig matches in a tree. In: Proceedings of the 17th International Conference on Data Engineering, pp. 595–604. IEEE Computer Society, Washington (2001)
12. Polyzotis, N., Garofalakis, M.: Structure and Value Synopses for XML Data Graphs. In: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 466–477. Morgan Kaufmann, San Francisco (2002)
13. Polyzotis, N., Garofalakis, M.: Statistical synopses for graph-structured Xml databases. In: SIGMOD 2002: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, pp. 358–369. ACM Press, New York (2002)
14. Polyzotis, N., Garofalakis, M., Ioannidis, Y.: Selectivity estimation for Xml twigs. In: ICDE 2004: Proceedings of the 20th International Conference on Data Engineering, p. 264. IEEE Computer Society, Washington (2004)

15. Wu, Y., Patel, J.M., Jagadish, H.V.: Estimating answer sizes for Xml queries. In: Jensen, C.S., Jeffery, K.G., Pokorný, J., Šaltenis, S., Bertino, E., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 590–608. Springer, Heidelberg (2002)
16. Polyzotis, N., Garofalakis, M., Ioannidis, Y.: Approximate Xml query answers. In: SIGMOD 2004: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 263–274. ACM Press, New York (2004)
17. Freire, J., Haritsa, J.R., Ramanath, M., Roy, P., Simeon, J.: Statix: making Xml count. In: SIGMOD 2002: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, pp. 181–191. ACM Press, New York (2002)
18. Wang, W., Jiang, H., Lu, H., Yu, J.X.: Bloom histogram: Path selectivity estimation for Xml data with updates. In: Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B. (eds.) VLDB 2004: Proceedings of the Thirtieth International Conference on Very Large Data Bases, pp. 240–251. Morgan Kaufmann, San Francisco (2004)

Improvement of Data Warehouse Optimization Process by Workflow Gridification

Goran Velinov, Boro Jakimovski, Darko Cerepnalkoski,
and Margita Kon-Popovska

Institute of Informatics, Faculty of Science and Mathematics
Ss. Cyril and Methodius University, Skopje, Macedonia
{goranv, boroj, darko, margita}@ii.edu.mk

Abstract. Generalized problem optimization of the relational data warehouses ,i.e., selection of the optimal set of views, their optimal fragmentation and their optimal set of indexes is very complex and still a challenging problem. Therefore, choice of optimization method and improvements of optimization process are essential. Our previous research was focused on utilization of Genetic Algorithms for problem optimization. In this paper we further optimize our solution by applying our novel Java Gid framework for Genetic Algorithms (GGA) in the process of relational data warehouses optimization. Obtained experimental results have shown, that for different input parameters, GGA dramatically improves efficiency of the optimization process.

1 Introduction

The main elements of a system for data warehouse optimization are: definition of solution space, objective function and choice of the optimization method.

The solution space includes factors relevant for data warehouse performance as view and index selection and view fragmentation ,i.e., partitioning. In some existing approaches the solution space for problem of optimization of data warehouses is simplified to great extent, and the selection of views or indexes is studied without considering other factors [1,2,17]. In [2] the problem of optimization of horizontal schema partitioning was defined. Optimization problem of data warehouses as combination of materialized views, indexes and horizontal data partitioning was introduced in [3] and the approach of vertical fragmentation was introduced in [8].

The objective function evaluates the quality of a solution. The optimization is usually considered with a certain constraints which divide whole solution space into two groups: feasible and infeasible solution. Constraints are well studied in existing research prototypes [1,8,17]. They can be disk space or maintenance cost constraint. In [1,8] the linear cost model was used under disk space constraint and formally system constraints were embedded in the penalty function. In [16,17] the objective function involved query processing cost under views maintenance (refreshment) cost constraint.

Some types of evolutionary (genetic) algorithms as optimization method were used in [2,12,16], and greedy algorithm with its variants and some heuristic

searching techniques were used in [5]. There it was shown experimentally that for large solution space other optimization methods have poor performances compared to genetic algorithms.

Usually the search for a solution using GA is a long and computationally intensive process that makes hard changes of a certain parameters of the problem if they need to be changed. Fortunately the GA is easily parallelized using data partitioning of the population among different processes. Fortunately the GA is easily parallelized using data partitioning of the population among different processes. This kind of parallelism ensures close to linear speedup, and sometimes super-linear speedup. Over the past years many variants of parallelization techniques are exploited for parallelization of GA [4,13] were exploited. Computational Grids [7] represent a technology that enables ultimate computing power at the fingertips of users. Today, Grids are evolving in their usability and diversity. New technologies and standards are used for improving their capabilities. Gridified Genetic Algorithms (GGA) have been efficiently used in the past years for solving different problems [9,10]. The Grid architecture resources are very suitable for GA since the parallel GA algorithms use highly independent data parallelism. Gridification and effective utilization of the powerful Grid resources represent a great challenge. New programming models need to be adopted for implementation of such parallel algorithms.

In this paper we present our approach for data warehouse optimization process improvement, based of novel Java Grid Framework for Genetic Algorithms, introduced in [11]. The GGA are applied to generalized optimization problem of data warehouse performance, based on our multi-dimensional model that includes complete vertical fragmentation. The model provides definition of all possible aggregate views and their data dependencies. Further, the solution space of the optimization problem includes different types of indexes. The system is tested for complex workload ,i.e., queries with projection, selection, join and grouping operations and with complex selection predicates. Furthermore, the algorithms are successfully applied to large solution spaces, up to 3544 objects (views and indexes). In [17] the algorithm is successfully applied to solution space that consists of 16 to 256 views. To show the efficiency of GGA we have compared them with standard GA. We have made many experiments which verified dramatic improvements (up to 920%) of the optimization process.

The paper is organized as follows: in Section 2 we describe definition of the solution space, in Section 3 we define objective function of the optimization system, in Section 4 we present the novel framework for GGA. In Section 5 we show an experimental evidence of the efficiency of the gridification, applied to generalized solution space and finally, in Section 6 we give conclusions and discuss our future work.

2 Definition of Solution Space

In this section we will define generalized solution space of the optimization process. The solution space is based on relational data warehouse schema which

includes definition of dimension relations, all possible aggregate views, their different states of normalization, named variants of views and relational dimensions as well as all possible indexes.

Let us define a data cube $DC = (DC_D, M)$ as a pair of dimension set $DC_D = \{D_1, \dots, D_n\}$ and measure attributes set M . Each dimension consists of set of relations $R_D = \{R_1, \dots, R_k\} \neq \emptyset$, where each dimension relation R_i is characterized by basic set of attributes $BA_i = PA_i \cup DA_i \cup FA_i$, where $PA_i \neq \emptyset$ is the set of primary key attributes (identifying attributes), $DA_i \neq \emptyset$ is the set of descriptive attributes and FA_i is the set of foreign key attributes. $HD = \{RD_1, \dots, RD_m\}$ is the set of dependencies between dimension relations of dimension D , named dimension hierarchy. Each RD_i is characterized by two dimension relations R_j^i, R_p^i , and it is presented by $RD_i : R_j^i \rightarrow R_p^i$, i.e., $RD_i(R_j^i, R_p^i)$, and also $PA_j \subseteq FA_p$. AA_i is the set of additional attributes of relation R_i , defined as union of basic sets of its dependent relations.

An implementation schema SC of data cube DC is defined as pair $SC = (DC, AV)$, where $AV = \{V_1, \dots, V_p\}$ is set of aggregate views. Each V_l is characterized by set of measure attributes $M_l \subseteq M$ and by basic set of group by attributes BGA_l . Aggregate views can contain different subsets of the set of measure attributes, which provides vertical fragmentation of measure attributes. Between the appropriate dimension relations and the aggregate views there exist 1:M relationships, i.e., primary key attributes of the dimension relation at the same time are foreign key attributes of the aggregate views. Group by attributes of any aggregate view consists of primary key attributes of at most one relation of each dimension. An extended set of group by attributes EGA_l of aggregate view V_l is defined as union of basic sets attributes of its dimension relations and additional set of group by attributes AGA_l is defined as $AGA_l = EGA_l \setminus BGA_l$. The set RSC is set of all dimension relations of implementation schema SC .

The aggregate view V^i derived from aggregate view V by adding (possible empty) subset of union of basic sets attributes of its dimension relations is variant of original view V . Intuitively, the view variants are created by denormalization of original schema. Our idea is to consider all possible states of normalization of data cube schema, from fully normalized state in $3^{rd}NF$, through the process of denormalization, to fully denormalized state (whole schema in one view). The variant V^0 of view V , which is characterized by basic set of group by attributes $BGA^0 = BGA \cup \emptyset$ is named zero variant of view V .

The important issue is how to select a variant of each dimensional relation, a set of views for materialization (each view presented with its proper variant) and a set of their appropriate indexes in order to minimize the total query processing time of the queries with a certain constraint.

The next step in definition of solution space is determining the indexing strategy. According to the theory of indexing, novel bitmapped indexes are very suitable for processing of data cube based queries. Thus, in this work for aggregate views, one-attribute bitmap indexes are considered. For each attribute of the extended set of group by attributes, one index can be created. Note that for each view V_l , the number of elements of the set EGA_l and sequence of all

indexes are disposed in advance. The set of all possible indexes in implementation schema SC is SI . The view length L_{V_i} of aggregate view V_i is defined as $L_{V_i} = 1 + k + m + n$, where k is the number of elements of additional set of group by attributes of the view AGA_i , m is the number of elements of set of measure attributes and n is the number of indexes of the view ,i.e., number of elements of extended set of group by attributes EGA_i . The parameters L_V are necessary to calculate number of bits (genes) for presentation of the solutions in our algorithms.

A data warehouse responses to large number of aggregated, data cube based, ad hock queries (workload) ,i.e., to dynamic and in principle unpredictable queries. However, lot of them can be determined a priori and can be formalized. To evaluate the real performances of the system we define the workload ,i.e., the set of predefined generalized project-select-join queries based on aggregation views of the data cube schema. The query definition is extended by using all possible subsets of grouping and selection attributes and by using complex selection expressions. Data operators ,i.e., predicates in selection expressions are also important for query processing time ,i.e., different data operators returns different numbers of tuples. Therefore, in this paper, in our query definition a set of different data operators is included. Query Q in schema $SC = (DC, AV)$ is tuple $Q = (M_Q, GA_Q, P_Q, F_Q)$, with $M_Q \subseteq M$ as set of measure attributes, $GA_Q \subseteq EGA_p$ as set of group by attributes, P_Q as selection expression and F_Q as expected frequency of the query. P_Q as selection expression of query Q is in form: $p_1 AND p_2 AND \dots AND p_m$, where $p_i : A_i \theta Value_i$, $\theta \in \{=, <, >, \leq, \geq\}$ is data operator, $Value_i \in Dom(A_i)$. The set SA_Q defined as $SA_Q = \{A_i, \dots, A_m\}$ is set of selection attributes (where clause attributes) and $SA_Q \subseteq EGA_p$ (EGA_p is extended set of attributes of the primary view). Data operators are divided in two groups: equality and inequality data operators. Thus, inequality operators usually return more then one tuple and after the selection by a certain set of attributes it is reasonable to group the data by the same set of attributes. Consequently, the two sets obtained by "group by" and "selection attributes" are not disjoint. Set of all possible queries in implementation schema SC is SQ .

Ratio of equality operator ER in selection expression of query Q is defined as k/n , where $n > 0$ is number of data operators ,i.e., number of elements of the set of selection attributes SA_Q and k is number of equality operators. The parameter ER is necessary to estimate number of tuples returned in each step of query execution i.e to calculate query execution cost. This is important to develop realistic query evaluator. Query Q is computable from (can be answered by) aggregate view V in schema $SC = (DC, AV)$ if: 1. $M_Q \subseteq M_V$ and 2. $GA_Q \cup SA_Q \subseteq EGA_V$, where M_V, EGA_V is set of measures and extended set of group by attributes of V , respectively. The previous definition is necessary to determinate the set of views from which each query Q can be answered. Thus, we define ratio of usability UR of aggregate view V as k/n , where $n > 0$ is number of queries ,i.e., number of elements of the set SQ and k is number of queries which can be answered by V . The UR parameter is necessary to estimate the frequency of usability ,i.e., "importance" of each view.

3 Definition of Objective Function

In this section we will propose a suitable evaluation function of the optimization process. Let SC_M is the state of the data cube schema SC with the set $AV_M \subseteq AV$ of candidate views for materialization where each of them is presented by its variant and the set $SI_M \subseteq SI$ of its candidate indexes. Let also all dimensional relations are presented by their appropriate variants. Then maintenance-cost constrained optimization problem is the following one:

Select a state SC_M of data cube schema SC that minimizes

$$\tau(SC, SC_M, SQ) = \sum_{Q \in SQ} F_Q * P(Q, SC_M),$$

under the constraint $U(SC, SC_M) \leq S$, where SQ is the set of predefined queries, F_Q is query frequency and $P(Q, SC_M)$ denotes the minimum processing cost of the query Q in the SC_M state of SC .

Let $U(SC, SC_M)$ is total maintenance cost defined as: $U(SC, SC_M) = \sum_{R \in RSC} G_R * m(R, SC_M) + \sum_{V \in AV_M} G_V * m(V, SC_M) + \sum_{I \in SI_M} G_I * m(I, SC_M)$, where G_R , G_V and G_I is update frequency of relations, views and indexes, respectively. Let $m(R, SC_M)$, $m(V, SC_M)$ and $m(I, SC_M)$ is the minimum cost of maintaining relations, views and indexes, respectively in presence of state SC_M .

We note that $P(Q, SC_M)$ is objective function of the problem. To calculate values of the functions $P(Q, SC_M)$, $m(R, SC_M)$, $m(V, SC_M)$ and $m(I, SC_M)$ we developed algorithms based on common query execution (processing) theory, presented in [6] and also on some ideas from [12], as well as [15].

4 Grid Framework for Genetic Algorithms

In this section we will present the architecture of the framework used for Gridification of Data Warehouse Optimization called Java Grid Framework for Genetic Algorithms (JGFGA). Various GA frameworks have been developed in the past years, implemented in different programming languages. The reason for developing new GA framework was not for introducing new model for parallel GA, but a framework that enables easier implementation of parallel GA capable of Grid execution. The framework is implemented in Java because of its platform independence and good OO properties. The framework design is founded on the following concepts: robustness, modularity, extensibility, flexibility and adaptability. The framework is organized in three main packages: *gridapp.grid*, *gridapp.ga* and *gridapp.util*.

The focus in this section will be the *gridapp.ga* package containing the core GA classes. The package *gridadd.util* contains utility classes used by many different classes in the framework. The last framework package *gridapp.grid* contains gridification classes used for the workflow implementation. There is a fourth package *gridapp.dw* which is not part of the framework, but holds the gridified data warehouse application.

Figure 1 presents the design of the core classes that implement or allow the implementation of GA in the framework. We will briefly describe and justify this design. The abstract class *Gene* $\langle T \rangle$ is used for representation of one gene. Real implementation of a gene inherits this class and some standard gene representations are available in the *gridapp.ga.impl* package. The abstract class *Chromosome* $\langle T \text{ extends } Gene \rangle$ represents one chromosome derived from a *Gene* type, designed to act as a container of *Genes* and implement a structure for gene organization. The only subclass included in the framework that implements the *Chromosome* class is the *ArrayChromosome* $\langle T \rangle$ class that organizes the genes into an array. The *Chromosome* class has several methods, mainly for accessing the genes, manipulation of the structure such as cloning or copying parts of the gene which later are used for recombination or mutation.

The class *Population* $\langle T \text{ extends } Chromosome \rangle$ represents single population of chromosomes, implemented as a container of chromosomes with standard methods for chromosome access and manipulation. The class *Domain* $\langle T \text{ extends } Chromosome \rangle$ plays key role in GA execution. This class represents a structure that holds additional information on the chromosomes and genes needed for their interpretation. As shown previously, genes and chromosomes carry only raw data organized in certain data structure. The *Domain* class together with *FitnessFunction* $\langle T \text{ extends } Chromosome, S \text{ extends } Domain \langle T \rangle \rangle$ adds order and meaning to this data, usually by storing additional data specific for the problem and objective function for a certain chromosome type.

The *Domain* is one of the key ingredients to the class *Evolver* that implements the actual process of evolution of the population. Another class that is used by the *Evolver* is the *Policy* $\langle T \text{ extends } Chromosome \rangle$ class. The *Policy* class specifies the rules the *Evolver* will use to implement the optimization. In another words the *Policy* is a program that *Evolver* will follow. The policy uses the subclasses of the abstract classes *NaturalSelector* and *GeneticOperator* to specify which operators will be used to create new chromosomes, and which selectors will be used to select the new chromosomes. The classes *NaturalSelector* and *GeneticOperator* are inherited in many classes from the *gridapp.ga.impl* package. Some of them are: *RouletteWheelSelector*, *TournamentSelector*, *LinearRankSelector*, *EliteSelector*, *RandomSelector*, *CrossoverOperator* and *MutationOperator*.

We continue with presentation of the Grid components of the JGFGA. Gridification of GA optimizations can be divided in two aspects. The first aspect is the choice of parallel GA technique. The second aspect is concerned with the underlying Grid connectivity with the Glite grid middleware.

The Parallel Genetic Algorithms (PGA) are extensions of the single population GA. The well-known advantage of PGA is their ability to perform speciation, a process by which different subpopulations evolve in diverse directions simultaneously. They have been shown to speed up the search process and to attain higher quality solutions on complex design problems [14]. There are three major classes of PGA: Master-slave, Cellular and Island. The nature of the Grid makes it best suited for Island PGA for achieving high performance parallelism. Another positive aspects of using several clusters in parallel is having bigger

population and separation into islands might increase diversity and speed up convergence.

The JGFGA implements the PGA by using workflow execution. A grid workflow is a directed acyclic graph (DAG), where nodes are individual jobs, and the vertices are inter-job communication and dependences. The grid workflow inter-job communication is implemented by input and output files per job. More precisely the first job outputs data into files, and after the job terminates the files are transferred as input files to the second job. The JGFGA enables implementation of PGA by means of four classes (jobs): *Breeders*, *Migrator*, *Creator* and *Collector*, all members of the *gridapp.grid* package. Breeders take as input a domain file and policy file. The domain file is simply Java serialized Domain object, while policy file is a java serialized Policy object. Having a domain and a policy the Breeder calls the Evolver and iterates several generations. The resulting domain is again serialized into a domain file. Migrators on the other hand take as input several domain files, execute the inter-population migrations and output one resulting domain. The Creator and Collector classes enable easy creation of new random populations and collect several populations into one.

Having this four classes, currently we have implemented the class *JobGraph < T extends Chromosome >* that enables automatic workflow generation. Generated workflows contain several iterations of Breeder and Migrator jobs. One iteration of breeding is called an epoch. An example of a sample workflow is shown on Figure 2.

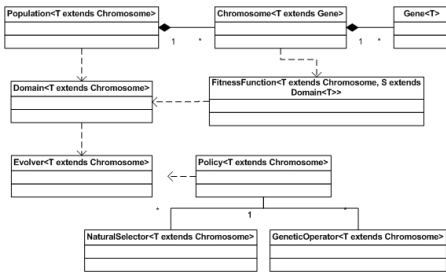


Fig. 1. Core Genetic Algorithm class design and connection

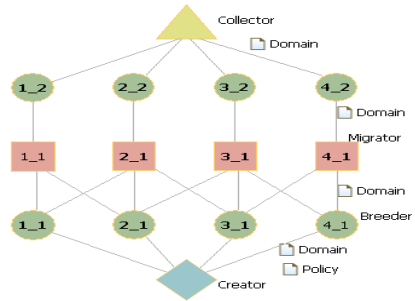


Fig. 2. Sample Grid Workflow Genetic Algorithm

The same class enables users to generate Job Description Language (JDL) files specifying the workflow for gLite grid middleware. These files are later submitted using the Grid submission tools. The parameters that can be given to JobGraph in order to model the Grid execution are: number of islands, number of epochs and migration width. Additional parameters that specify the population and policy are number of iteration per epoch and size of a single population. Further parameters that define Grid execution characteristics are the Retry counts which make the workflow more resilient when some job hits problematic Grid site and fails to execute.

The framework also contains Grid components that enable usage of the gLite Grid testbeds. The components are based on the Workload Management System (WMS) and Logging and Bookkeeping (LB) Web Services (WS). Additional functionality of the framework is Grid authentication by using VOMS-proxy-init. This makes the framework completely independent from the Glite UI installation and enable users to use it from any available platform.

5 Experimental Results

In this section we present some of the experimental results obtained by JGFGA system and implemented Grid Workflow Optimization. We compare performances of the SGA (Standard Genetic Algorithm) and GGA(Gridified Genetic Algorithm).

For the efficiency of genetic algorithm several input parameters are important. In this work we fixed the parameters to: population size PS=1000 divided into 10 islands, probability of mutation to PC=0.02. We also include elitism selector to retain the best chromosome. For all experiments termination condition of optimization process was 150 generations. The workflow was designed to divide the generations into 3 epochs, each with 50 iterations. Between the epochs we submitted Migration jobs to mix chromosomes from 2 adjacent islands. Both algorithms were applied to generalized solution space based of our model.

Most important parameter of optimization process is the size of the solution space ,i.e., the length of the chromosome for genetic algorithms. A comparison of algorithms according to the solution space size is shown in Figure 3. The numbers adjacent to each experiment label corresponds to number of objects in each optimization. For better presentation of the performances of the algorithms on the same chart, we scaled values of optimization process execution time. On y-axis the benefit of optimization process execution time relatively to the worse algorithm is shown. Evidently, in all cases GGA algorithm has the significant improvement of the optimization time. Note that improvements of GGA rise by increasing the solution space size, which is another important feature: the scalability of our approach and its appropriateness for practical implementation.

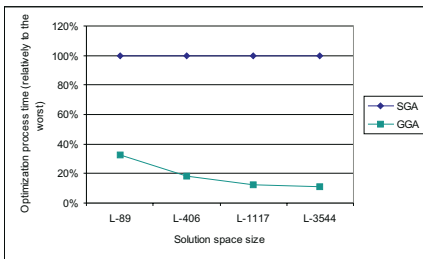


Fig. 3. Optimization process time according solution space size

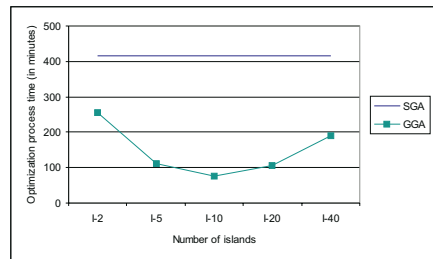


Fig. 4. Optimization process time according number of islands

We have also analyzed the effect of workflow parameters on the performance of the GGA. Our tests were done over a fixed solution space (L-406) and fixed population of 1000 chromosomes, divided into 2, 5, 10, 20 and 40 islands. The overall performance analysis is shown in Figure 4. The aim of this analysis was to see the performance effect depending on the different amount of jobs submitted on the grid, while at the same time analyzing the resulting solution quality. As it can be seen from Figure 4 the number of islands should be chosen carefully depending on the problem size. We can conclude that in our case making many islands can make the optimization time worse due to unpredictable behavior of availability of the Grid resources. Finally, we note that values of given solutions by using GGA are in the range of 99%-101% of solutions given by using SGA.

6 Conclusion and Future Work

The performance of the system of relational data warehouses depends of several factors and the problem of its optimization is very complex and making a optimal system is still a challenge. Thus, in this paper we have focused on improvement of the efficiency of generalized optimization problem of relational data warehouses. We fully analyzed the problem by including lot of factors relevant for optimization of the system ,i.e., view selection, vertical view fragmentation and index selection. We have achieved significant performance improvements of the optimization process compared to the standard GA and we have verified those improvements by performing large set of experiments. Also we have focused to testing different kinds of approaches for Grid parallelizations of GA, mainly focused on what kind of models of workflow parallelization (based on previously defined parameters) can be most effective when using the Grid infrastructure.

In the future we plan to extend our multidimensional model by including horizontal partitioning and definition of a clustering strategy and to define an optimization process by applying the algorithms to the extended space. Further development of the JGFGA is oriented towards implementation of grid data management and job wrapping per island in order to overcome the problem with job failure. The job wrapping can be easily facilitated since the input and output files of each job are in the same format (Domain files). Hence in failed nodes we can skip the epoch iteration and state that output files are the same as input files without any modification. This approach will decrease the convergence and might produce worse final results, but will increase job success rate.

References

1. Aouiche, K., Jouve, P., Darmont, J.: Clustering-Based Materialized View Selection in Data Warehouses. In: Manolopoulos, Y., Pokorný, J., Sellis, T.K. (eds.) ADBIS 2006. LNCS, vol. 4152, pp. 81–95. Springer, Heidelberg (2006)
2. Bellatreche, L., Boukhalfa, K.: An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2005. LNCS, vol. 3589, pp. 115–125. Springer, Heidelberg (2005)

3. Bellatreche, L., Schneider, M., Lorinquer, H., Mohania, M.: Bringing Together Partitioning, Materialized Views and Indexes to Optimize Performance of Relational Data Warehouses. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) DaWaK 2004. LNCS, vol. 3181, pp. 15–25. Springer, Heidelberg (2004)
4. Cantu-Paz, E.: A Survey of Parallel Genetic Algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis* 10(2), 141–171 (1998)
5. Chan, G.K.Y., Li, Q., Feng, L.: Optimized Design of Materialized Views in a Real-Life Data Warehousing Environment. *International Journal of Information Technology* 7(1), 30–54 (2001)
6. Elmasri, R., Navathe, S.B.: *Fundamentals of Database Systems*, 4th edn. Addison-Wesley Publishing Company Inc., Reading (2003)
7. Foster, I., Kesselman, C.: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco (1999)
8. Golfarelli, M., Maniezzi, V., Rizzi, S.: Materialization of fragmented views in multidimensional databases. *Data & Knowledge Engineering* 49(3), 325–351 (2004)
9. Herrera, J., Huedo, E., Montero, R.S., Llorente, I.M.: A Grid-Oriented Genetic Algorithm. In: Sloat, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 315–322. Springer, Heidelberg (2005)
10. Imade, H., Morishita, R., Ono, I., Ono, N., Okamoto, M.: A Grid-Oriented Genetic Algorithm Framework for Bioinformatics. *New Generation Computing* 22(2), 177–186 (2004)
11. Jakimovski, B., Cerepnalkoski, D., Velinov, G.: Framework for Workflow Gridification of Genetic Algorithms in Java. In: Proc. of the ICCS 2008: Advancing Science through Computation, Krakow, Poland (June 2008)
12. Ljubić, I., Kratica, J., Tosić, D.: A Genetic Algorithm for the Index Selection Problem. In: Raidl, G.R., Cagnoni, S., Cardalda, J.J.R., Corne, D.W., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E., Meyer, J.-A., Middendorf, M. (eds.) EvoIASP 2003, EvoWorkshops 2003, EvoSTIM 2003, EvoROB/EvoRobot 2003, EvoCOP 2003, EvoBIO 2003, and EvoMUSART 2003. LNCS, vol. 2611, pp. 280–290. Springer, Heidelberg (2003)
13. Nowostawski, M., Poli, R.: Parallel Genetic Algorithm Taxonomy. In: Proc. of the Third International conference on knowledge-based intelligent information engineering systems, KES 1999, Adelaide, pp. 88–92 (1999)
14. Sena, G.A., Megherbi, D., Isern, G.: Implementation of a parallel genetic algorithm on a cluster of workstations: travelling salesman problem, a case study. *Future Generation Computer Systems* 17(4), 477–488 (2001)
15. Tsois, A., Karayannidis, N., Sellis, T., Theodoratos, D.: Cost-based optimization of aggregation star queries on hierarchically clustered data warehouses. In: Proc. International Workshop on Design and Management of Data Warehouses DMDW 2002, Toronto, Canada, pp. 62–71 (2002)
16. Velinov, G., Gligoroski, D., Kon-Popovska, M.: Hybrid Greedy and Genetic Algorithms for Optimization of Relational Data Warehouses. In: Proc. of the 25th IASTED International Multi-Conference: Artificial intelligence and applications, Innsbruck, Austria, February 2007, pp. 470–475 (2007)
17. Yu, J.X., Yao, X., Choi, C., Gou, G.: Materialized Views Selection as Constrained Evolutionary Optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews* 33(4), 458–468 (2003)

Towards the Design of a Scalable Email Archiving and Discovery Solution

Frank Wagner¹, Kathleen Krebs², Cataldo Mega³, Bernhard Mitschang¹,
and Norbert Ritter²

¹University of Stuttgart, IPVS

{fkwagner, mitsch}@ipvs.uni-stuttgart.de

²University of Hamburg, VSIS

{kkrebs, ritter}@informatik.uni-hamburg.de

³IBM Deutschland Entwicklung GmbH

cataldo_mega@de.ibm.com

Abstract. In this paper we propose a novel approach to specialize a general purpose Enterprise Content Management (ECM) System into an Email Archiving and Discovery (EAD) System. The magnitude and range of compliance risks associated with the management of EAD is driving investment in the development of more effective and efficient approaches to support regulatory compliance, legal discovery and content life-cycle needs. Companies must recognize and address requirements like legal compliance, electronic discovery, and document retention management. What is needed today are EAD systems capable to process very high message ingest rates, support distributed full text indexing, and allow forensic search such to support litigation cases. All this must be provided at lowest cost with respect to archive management and administration. In our approach we introduce a virtualized ECM repository interface where the key content repository components are wrapped into a set of tightly coupled Grid service entities, such to achieve scale-out on a cluster of commodity blade hardware that is automatically configured and dynamically provisioned. By doing so we believe, we can leverage the strength of Relational Database Management Systems and Full Text Indexes in a managed clustered environment with minimal operational overhead.

Keywords: Enterprise Content Management, Email Archiving and Discovery, Legal Compliance, Virtualization, Scale-out, Engineering for Scalability.

1 Introduction

Recently there has been considerable interest in the corporate governance practices of modern corporations, particularly since the high-profile collapses of a number of large U.S. firms such as Enron Corporation and WorldCom. As a consequence due to corporate governance and legislative regulations, companies around the world are forced to archive all relevant electronic information for compliance reasons. It turns out, that the most important type of business relevant electronic information these days consists of: Email, Instant Messaging (IM), and recently also transcripts of audio recording (from

telephone and audio chat conversations) [3,7,19]. As a consequence, more and more companies are forced to deploy Email Archiving and Discovery (EAD) systems that help find and retrieve all necessary information when needed. Usually EAD systems are built upon Enterprise Content Management (ECM) systems.

From a technical point of view, these new compliance requirements pose some non-trivial challenges to the underlying Enterprise Content Management System for three reasons: 1) the rate at which emails and instant messages arrive can reach the millions per day and therefore requires an explicitly tailored ingest, index and archive framework, 2) over the years the number of archived emails and instant messages can become prohibitive, i.e. in the range of hundreds of billions and with growing sizes in the range of petabyte of data, 3) finally, the support for e-Discovery. In litigation cases, the task to produce the “relevant” information within an acceptable time window is called e-Discovery. E-Discovery can be described as a form of forensic information retrieval in very large collections. For more detailed description see [1].

Globalization is forcing enterprises to remodel and restructure their business processes in order to allow merging, splitting and outsourcing of certain business units and services as it best fits. Business relevant information needed by the implied business processes must be available at all times at the right location for consumption and use. To IT departments this means that they must accommodate constant change in a flexible and cost efficient way. These requirements are passed on to the business solutions in charge of archiving and managing all electronic content produced. In this paper the term “content” stands for any kind of electronically stored information [7] relevant to business processes, i.e. messages, documents, rich media content, web content, records, data, metadata, etc. The classes of solutions we want to analyze and later enhance are specifically Email Archive and Discovery systems, a specialized form of the more general purpose Enterprise Content Management systems which are used in the context of corporate and legal governance. The major reasons for using an EAD system are laws and regulations enforcing regulatory compliance when archiving email messages [20,21] demanding safekeeping of electronic documents (cf. Sarbanes-Oxley Act, 2002). Besides the necessity to comply with legal requirements concerned with sensitive and classified information or involving discretionary authority, today archiving email is a general obligation.

In order to determine, which information qualifies to be archived, email messages need to undergo several pre-processing steps. This makes an EAD workload so unique; it is the mix and sequence of primitive archive operations that differ so much from the typical workload found in a general purpose ECM system. With the new needs, new technical challenges came up that ECM systems must face. Scalability in multiple dimensions has to be considered the most important one. Investigations in these areas have shown that the individual EAD components must effectively process and manage huge amounts of continuously incoming data [18], in some case 10 million email messages per week for a single large multi-national company. The unsteady inbound and outbound flow of business information must be parsed, interpreted, tagged, indexed and archived. Later also searched, retrieved and disposed. Given the type of workload and the amount of data this is a very ambitious task. Past experience shows that generic ECM systems must evolve and adapt. They must be enhanced in new, innovative ways, combining old and new technology in order to cope with the given challenges.

In this paper we will emphasize on non-functional requirements, more explicitly, performance, scalability, and efficiency based on reduction in storage need, processing and maintenance costs. The market analysts believe that in the compliance space, Email Archiving and Discovery is becoming an important facet of ECM and we know by experience that this is pushing the edge on throughput, scale and cost. We therefore consider EAD a challenging ECM field of interest worth to be researched and better understood.

The remainder of the paper is organized as follows: Section 2 gives an overview of related work. Section 3 introduces the concept of “Email Archiving and Discovery”. Section 4 first defines scalability in the EAD context, and then describes our approach. In section 5 we describe our test environment and discuss the measured results. Our final section 6 presents our conclusion and an outlook of the work ahead.

2 Related Work

Today one can choose from many commercial email archiving solutions that exist on the market [9]. In addition to the commercial systems we have also considered open source technology that is en-vogue. P2P file sharing systems like BitTorrent¹ and Gnutella in a sense can also be seen as specialized content management systems. They focus on one specific job, i.e. file sharing, which they perform very well. What primarily caught our attention was their scale-out characteristics and automatic topology reconfiguration ability. Earlier P2P systems had scaling issues because processing a query involved flooding it to all peers. The demand on each peer would increase in proportion to the total number of peers, quickly overrunning the peer’s limited capacity. More recent P2P systems like BitTorrent scale well because of the lessons learned. Here demand on each peer is independent of the total number of peers. In addition, there is no centralized bottleneck, so the system may expand indefinitely (scale-out) without the addition of supporting resources other than the peers themselves. In the event of peers joining or leaving the configuration the system is also able to re-configure itself in an automatic fashion (automatic configuration).

Now, despite of all valuable features available, P2P lack some important capabilities that are mandatory for ECM systems, for example:

1. There is virtually no content management framework with an underlying repository infrastructure and application integration layers. Content management is reduced mostly to file system semantics.
2. Support for content retrieval is purely via hash key value lookup. The ability to allow content retrieval via metadata search is missing. The latter is a service that implies the existence of a document data model. If wanted, this service must be added by some other means, like a catalog database or a full text engine. For this purpose, we considered the Google MapReduce [8] programming model and the Hadoop platform² of the Apache Lucene project. Both are approaches striving for distributed data processing around the construction and management of clusters of full text indexes.

¹ <http://www.bittorrent.com>

² <http://lucene.apache.org/hadoop/>

3. P2P lacks important storage management functions, a capability that is pivotal to ECM systems fundamental to satisfy hierarchical storage management needs. Nevertheless we want to have a closer look at the P2P concepts in order to evaluate their adoption in our EAD design approach.

The most common approach adopted in ECM systems w.r.t. scalability was scale-up on the basis of large multi-processor systems. Measurements [15,16,23] performed on cluster and grid systems indicate that scale-out might be more cost effective and affordable if administration and maintenance overhead can be kept to a minimum.

3 Email Archiving and Discovery (EAD)

EAD systems are meant to complement and support email servers with their work in managing mailboxes. They are concerned with the routing of in-bound and out-bound emails and managing the user’s inboxes. In this context emails are scanned for spam, viruses and when necessary also classified for later post-processing. Therefore, email servers provide interfaces to plug in filters for accomplishing the different tasks including compliance scanning.

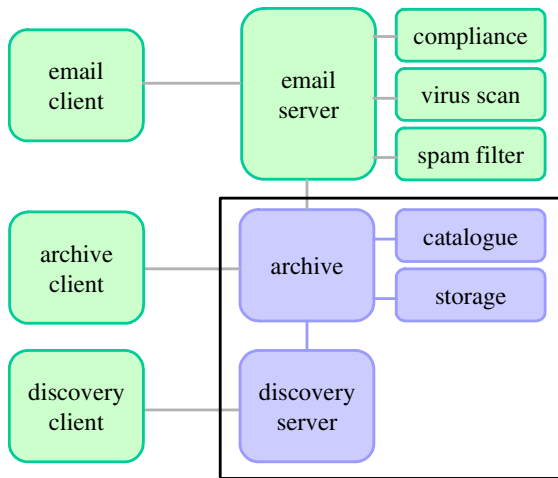


Fig.1. Major EAD system components within an email server environment

Figure 1 shows a typical email server environment including an EAD system (inside the black border). The central archive component constitutes the core part of the EAD system assisted by additional specialized archive tasks or services. Its main task is to manage the content lifecycle for automatic email archiving. As can be seen, the archive component relies on three other components:

1. The system catalogue that stores and manages the metadata in a searchable form based on an optimized EAD data model.
2. The storage system that is responsible of storing and managing the actual electronic content in its original and other derived formats, so called renditions.

3. Finally, the information discovery services that provide e-Discovery functions via the means of additional discovery servers. The latter are needed for supporting information discovery performed by compliance officers and case workers. Given the size of the collections archived, e-Discovery must be tailored and tuned towards the ability of handling very large result lists produced by compliance investigations and its forensic queries.

3.1 EAD Functional Requirements

Let us summarize the key functional requirements of EAD. The three major functional areas are:

1. Email archiving,
2. Retentions and Records Management.
3. E-Discovery.

Email archiving is tailored towards storage optimization and management of email server farms. Archiving is done automatically on a regular basis according to a set of predefined corporate rules.

The next step toward true email management is to treat emails like business records, in order to achieve corporate and government compliance. Thus, the information saved in email messages is now made accessible throughout the enterprises itself. Legislative regulations mandate also the preservation of the chain of custody, i.e. businesses must prove that requested information has been preserved and can be produced in case of litigation within a finite time. All these regulations are translated into the set of archive rules which then dictate the kind of long-term archiving adopted.

E-Discovery is the framework targeted to support compliance officers and workers to find and retrieve information when ordered to do so. Producing information in the compliance context must be understood as producing “all” relevant information, not just portions.

3.2 EAD Non-functional Requirements

When designing an EAD system, major focus must be put on the aspect of efficiently handling variable email message ingest rates, and coping with sometimes very high spikes and fluctuations of these rates. Therefore, the three key non-functional requirements for EAD systems are:

1. High throughput and low response times under normal production conditions.
2. The ability to dynamically adapt to the current load situation by acquiring additional resources or releasing unused ones according to the goals of matching the load variations and minimizing system power consumption.
3. Reliability, authenticity and security are mandatory for an archive in a real-world environment.

3.3 EAD Activity Flows

A major task of an EAD system is to monitor emails entering and leaving the enterprise as well as the emails exchanged within. Whenever necessary, emails are

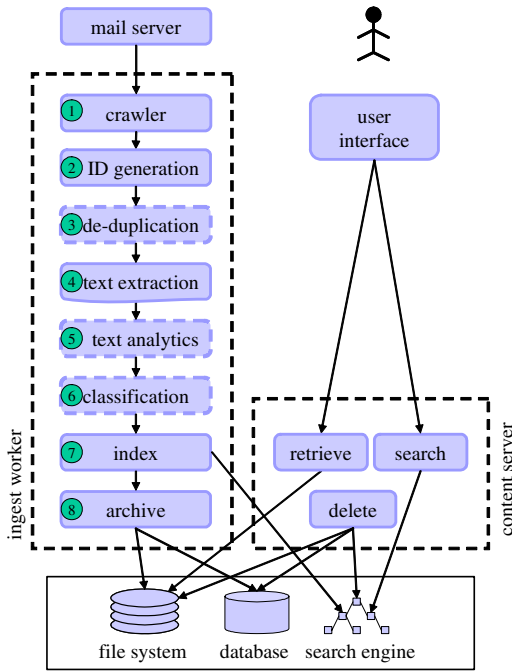


Fig. 2. Ingestion process, search and retrieve

captured, parsed, interpreted, and, if found relevant, finally archived. From an archive workload perspective, message ingest processing is the predominant macroscopic operation. At microscopic levels the necessary processing steps during the ingestion are as follows:

1. A mail server as message or document source has to be determined. Then an email crawler is scheduled to connect to the chosen source server using appropriate credentials. Email extraction is then started according to set of predefined archive rules.
2. From within the ingest work queue, unique hash IDs are generated based on the email using a hash algorithm like SHA-1 or SHA-256.
3. The IDs are used to check, whether the email itself or a part of it has already been stored in the system. If it is not a duplicate, the email is archived.
4. In the case that an email contains rich media attachments, the plain text is extracted by means of a text extraction step. For example, plain text would be extracted from PDF or from Microsoft® Office documents.
5. The extracted plain text can now be further processed by linguistic methods used to augment the text information, before it is full text indexed and ready for e-Discovery.
6. Text classification is an important step in a compliance scenario. Emails with content that does not comply with the rules have to be intercepted. Typically a supervisor then decides if the mail can be forwarded to its destination or needs to be sent back to its originator for further actions.

7. If an email is eligible for archiving, then it is also full text indexed. For this purpose the extracted plain text from headers, body and attachments is passed to the Lucene full text index engine.
8. Finally the email is stored in the archive in a format well suited to provide 100% format fidelity with its original. In some scenarios other renditions of the email are created additionally.

4 Scalable System Design for EAD

Scalability is a non-functional system characteristic that reflects a number of intriguing issues. As is known from literature, scalability guarantees that the system behaves similarly under varying usage characteristics like spikes in workload and managed data volumes. It is clear that a scalability analysis has to target system components and execution traces in order to uncover existing bottlenecks mostly in communication overhead, data sizes, and algorithmic performance.

An EAD system is said to be scalable if its workload throughput improves proportionally to the capacity of available system resources. More practically it is said to scale if it is suitably **efficient**, sufficiently **easy** to administer and lastly also **affordable** when applied to complex cases, e.g. a large input data set or large number of participating nodes in the case of a distributed clustered systems.

4.1 EAD System Scalability Factors

Scaling a system can be done in two directions: vertically (scale-up) by using larger machines, and horizontally (scale-out) by adding more machines. Scale up is in our approach supported to some extent by the multi-threaded ingestion. Our major objective is scale-out with respect to the total system size, i.e. the total amount of aggregated electronic content, an EAD system can manage and the overall throughput performance characteristics with respect to ingest, index and query response times. Next we will focus and discuss design factors that limit or enhance the EAD system scalability. As listed before, we found that there were few but key factors that limit system scalability in an EAD system. Due to these factors we have designed special enhancements for email archiving and e-discovery systems. Here is what we think will boost scalability:

1. In our approach we **separate immutable from mutable data**. Immutable data is best treated by Full Text Index (FTI) technology, mutable data is most effectively managed by Relational Database Management Systems (RDBMS) technology (cf. section 4.2).
2. **Avoid a single central physical catalog**. In our approach we aim at scale-out using a cluster of commodity machines instead of highly optimized large SMP systems. The central physical catalogue is replaced by a central logical but physically distributed catalogue, mapped onto a cluster of RDBMS and FTI.
3. **Avoid costly join operations**. In our approach joins among RDBMS and FTI data are avoided due to a specific data organization as described in section 4.2.

4. **Avoid processing queue serialization.** In our approach document IDs are generated from the document content itself using a SHA-1 or SHA-256 algorithm. By adopting this approach every ingest node can generate the document IDs independently, allowing scale-out to happen.
5. **Avoid wrong processing order.** In our approach email messages are processed right after they are extracted from their source system. They are loaded into memory and archived immediately after the corresponding decision. All other processing steps are done in parallel. Redundant data copy operations are avoided or minimized.

Further scalability factors are:

- **Avoid static deployment topology.** We exploit an automatic re-configurable deployment topology using DHT based P2P technologies [9,11,13,24].
- **Avoid static resource provisioning.** We introduce dynamic resource provisioning based on a specific EAD resource model and implementing a SLA cost model [12].

These two issues refer to on demand technologies that are used to dynamically configure the system infrastructure. These technologies are currently realized in our system prototype and will be evaluated later.

4.2 Scalable Data Organization

The core functionality of an archive is the management of data and metadata. Let us therefore look at the underlying data model of the EAD catalogue. The metadata necessary to manage the EAD archive can be classified into two different categories: a) mutable and b) immutable. Now by its very nature most of the archive metadata is immutable because it is derived from the email messages to be archived. Therefore we decided to store it in a distributed full text index. A cluster of FTI engines provides efficient and effective mechanisms for searching on unstructured and semi-structured data and is therefore well suited. In our current implementation we are using a cluster of Apache Lucene³ search engines. Note, that storing only immutable metadata in the full text index helps avoid paying the index update and re-org penalty. All the mutable metadata crucial to the operations of the EAD system is kept in a cluster of database systems distributed over a variable number of nodes. Using a cluster of RDBMS ensures consistency and transactional integrity of the create, retrieve, update and delete (CRUD) operations. In our case we use a cluster of IBM® DB2® databases for this purpose.

By adopting this strategy, it is absolutely necessary to avoid join operations during searches between the database and the full text index. We achieve this goal by restricting email searches to the full text index which contains all the relevant information. Experiences with email archive solutions in the market have shown that for EAD systems this is possible. The biggest benefit from this design approach is that the amount of information stored in the database is so kept several orders of magnitude lower than the information stored in the full text index. The full text index is therefore used for search and discovery, the database for all other archive operations. Typical

³ <http://lucene.apache.org>

archive operations would for example be in the areas of: security, administration, management, workflow, auditing, and the like.

4.3 System Engineering Solution

A data model with one central catalog is not sufficient in large EAD scenarios. One problem is performance, due to the catalog becoming the bottleneck. The other problem is the size of the managed data. Most systems impose some limits, like the maximum size of a file system or the maximum number of files, or the maximum number of documents in an index.

The approach we are using to tackle these problems is to break up the catalog into multiple catalogs which together yield one virtual catalog. The catalogs can then be placed on different nodes in a cluster. This allows us to 1) grow the system as necessary, and 2) to arrange the catalogs in a way that the expected performance requirements are met.

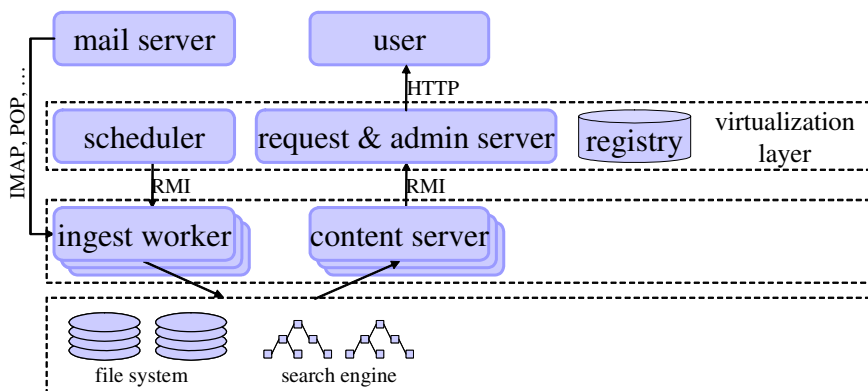


Fig. 3. Architecture of the EAD system

Figure 3 shows our approach. At the top are the two most important external components that are driving the EAD system: The mail server is the data source and manages the emails before they are archived. The users are searching for emails in the archive and are retrieving selected emails.

Next, the virtualization layer integrates the distributed nodes into one virtual system. The implementation that was used to produce the results given in the next section uses a central database as registry. This database keeps track of the current state of the system and the location of existing indexes. However, this is only an initial approach. Future versions of the prototype will realize the virtualization layer with more dynamic and scalable techniques from peer-to-peer computing like distributed hash tables.

Two further components are part of the virtualization layer and are driving the system: the scheduler and the request server. The scheduler coordinates the work that has to be performed by the system. The most important jobs it schedules, and currently the only ones implemented, are jobs to process new emails. The scheduler therefore maintains a list of mailbox locations that have to be processed together with the

respective archiving policies. These jobs are passed on to the ingest workers. Other jobs manage the expiration of old emails, and re-arrange the catalog, e.g. optimize or merge the Lucene indexes.

The request server provides the user interface and is mainly used to receive and process search and retrieve requests from the users. It forwards these requests to the relevant content servers and integrates their results which are then presented to the user. Besides this, the request server also provides interfaces for administrative tasks.

The components of the next lower layer are distributed over a cluster of nodes. The ingest workers retrieve the emails from the email servers and archive them. Thereby they are extracting metadata that can be used to search for emails later on. The content servers provide interfaces to search for emails and to retrieve them.

At the bottom the actual storage is depicted. This can be file systems, full text indexes, databases or more elaborate ECM systems. They are typically assigned to individual nodes, but they could also be shared resources as indicated in the figure.

5 Measurements and Evaluation

In order to verify our EAD design enhancements and to measure the scalability enhancements we have implemented an EAD prototype which allows us to perform scalability tests and to measure actual system performance in the key design areas. The prototype will be enhanced in an iterative fashion and our tests refined to use more elaborated workloads as we go. At each measurement step we then verify and document to what extent the overall system scalability is improved.

5.1 Test Workload

For the tests conducted in this paper we used the Enron email dataset⁴ as the corpus of test data. It consists of 517,431 real emails that were made public during the investigation of the Enron scandal. When the Enron data were published all attachments were removed. Since attachment processing is a major and the most costly step during ingest, we randomly added documents crawled from the internet in order to circumvent this deficiency. The logic to add attachments to our corpus of test data was derived by analyzing the attachment distribution of real mailboxes on the web. The appended document set contains PDF and HTML documents as well as Microsoft and OpenOffice documents, but also images and binary data. The average size of an email thereby grew from 2.7 KB to 46 KB and thus the overall data volume grew to approximately 23 GB.

5.2 Test Infrastructure

We evaluated our EAD approach on an IBM BladeCenter® with HS20 blades with two 3.2 GHz Intel® Xeon® CPUs each, and HS21 blades with two dual-core 2.3 GHz each. All blades have 4 GB of main memory and are connected with GBit-Ethernet.

In this experimental setup, the emails to be processed by the EAD system are retrieved from zip files stored in a shared file system using a custom JavaMail™

⁴ <http://www.cs.cmu.edu/~enron/>

provider. Thereby we achieve an adequate throughput and avoid the struggle to set up a sufficiently powerful email server infrastructure. Particularly when we add steps like deleting or stubbing the emails on the email server we will have to switch to real email servers.

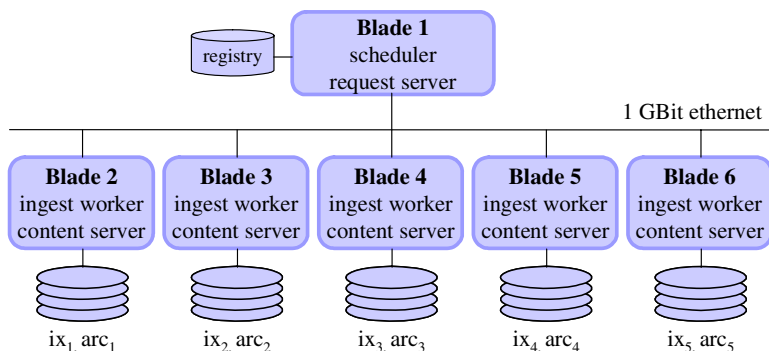


Fig. 4. Layout of the test infrastructure

Figure 4 shows the layout of the system as it was used for the measurements shown in this section. Blade 1 is one of the HS20 blades and is the scheduler during ingest benchmarks and the request server for search benchmarks. The database with the registry is also located on this blade. The other blades 2-6 are either ingest workers or content servers depending on the tests performed. They are storing the index and the archived emails on local hard disks. The shared disk with the test data set is not shown in this figure.

5.3 Ingest Measurements

To evaluate the scalability of the ingestion processing of the approach we set up a series of tests where the dataset described in the previous section was processed by a varying number of worker nodes. On each node four threads were processing the emails in parallel. As the prototype is still work in progress, it did not perform all the steps mentioned in section 3.3. The ingest workers retrieve the emails from ZIP-files, where each ZIP-file contains all the emails from one mailbox. For each email hash keys are calculated for all parts, the complete email is stored in the local file system, and a document is added to the Lucene index containing the header-fields, the body and plain text extracted from the attachments.

Figure 5 shows the ingest rate for one to five ingest workers, each on its own blade. The separation of the system into multiple, nearly independent workers leads to a linear scalability of the ingest processing in the number of nodes. There is one system-internal bottleneck, the scheduler, which assigns the jobs to the workers. But unless the mailboxes are very small, the time spent inside the scheduler is negligible compared to the time to process the mailboxes.

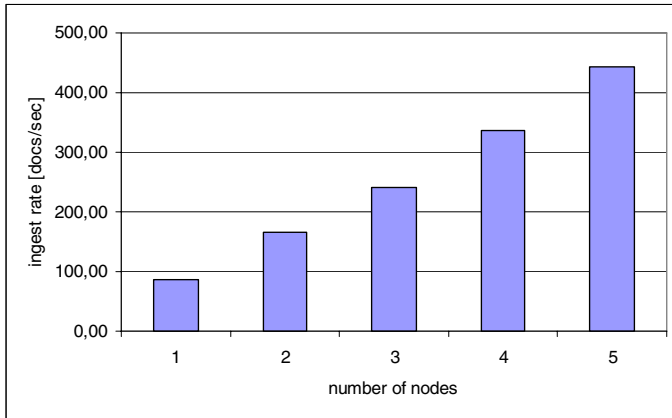


Fig. 5. Dependency of ingest throughput from number of nodes

We also evaluated the behavior of the ingest with larger indexes. Therefore the test data set was ingested up to 10 times into one index. The variation in the time for the iterations was below 4% and no tendency in one direction was observable. Consequently, the time for adding a single document can be considered to be basically independent of the index size. The jitter is likely due to internal optimization of the index by Lucene.

5.4 Search Measurements

For the search measurements we built a benchmark of 45 queries based on information from a domain expert. The queries comprise single-word queries, queries with multiple words, queries with phrases and queries with Boolean predicates. Wildcard queries and fuzzy queries were not included in the benchmark. Some of the queries are also restricting the hits to a certain date range. Additionally, the queries are further restricted to a few mailboxes the user is allowed to search on.

To execute this benchmark we are using JMeter⁵ from Apache. At the beginning of each test a set of threads is started. Each thread simulates one concurrent user. A user sends a random query within an HTTP-request to our web application. After receiving the page with the hits, the user waits between one and five seconds (think time). An average think time of three seconds is relatively small and imposes more load as a regular user would generate. When the think time is over the user starts from the beginning and issues another query.

On receiving a request, our request server first retrieves the set of available indexes from the registry. In the measurements shown here, the query is always sent to all indexes. This is the worst case, given if the indexes were not partitioned by some exploitable criteria. In other cases the registry could return only those indexes that are known to contain emails for the user that issues the query. With many distributed indexes this pruning considerably reduces the load on the search nodes.

⁵ <http://jakarta.apache.org/jmeter/>

We have implemented a quite simplistic approach to process the queries, where a `ParallelMultiSearcher` from Lucene running on the request server is driving the search, and `RemoteSearchables`, also from Lucene, are used to remotely access indexes over Java™ RMI. With this approach, the ranking of the hits is very expensive. For each query, first the global document frequencies for terms in the query are calculated sequentially. Only the actual search is performed in parallel.

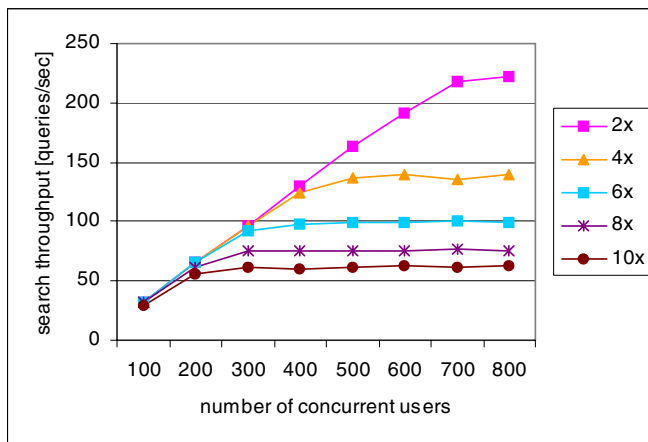


Fig. 6. Search throughput on one node with different index sizes

Additionally, even for a simple query with four terms executed on one search server over 100 packets are sent over the network. For the measurements in this paper we did not change this implementation as it up to now served its purpose.

Figure 6 shows the throughput in queries per second when using only one search server. The different lines represent different index sizes. For this test we loaded the test data set up to ten times into the system. The figure shows that the achievable throughput clearly depends on the index size. With a small index, where the data set was ingested once and the final index has a size of around 914 MB, the system can handle the load. With ten times the documents in the index, and a final index size of 8818 MB, the system with one search server saturates at around 62 queries per second.

This shows that the time needed to search in the index, in contrast to adding documents, is dependent on the index size. In this test, this might be due to the growing result size as the same data set was added multiple times. But separate measurements showed that this is also the case when there is only one hit in all result sets. This has to be taken into account when generating the index.

In the following test we therefore distributed the documents from the largest test into the indexes on a varying number of nodes. Figure 7 shows the throughput in queries per second with different numbers of search servers and concurrent users. For this test we loaded our test data set ten times into the system. For a small number of concurrent users the throughput is independent of the number of nodes. However, with more and more users the systems profits from distributing the data over more machines.

The measurements primarily showed the performance aspects of the scalability. With respect to size we are only at the beginning and far from the sizes required in medium and large environments.

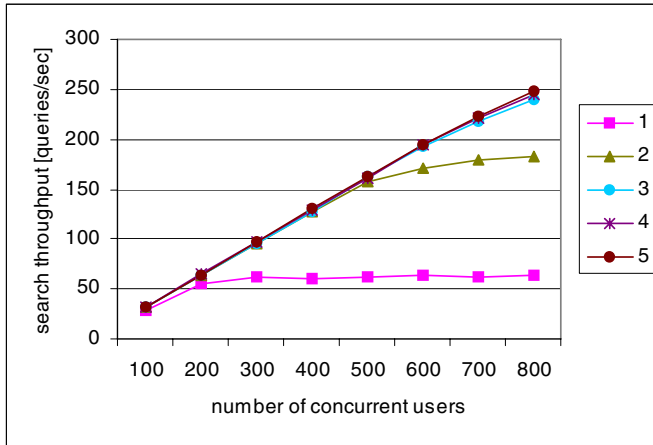


Fig. 7. Search throughput with different number of search servers

6 Conclusion

In this paper we introduced and discussed the concept and design of an EAD architecture that is targeted to enable high-end scalable email archiving and discovery at affordable levels. Our prototype approach builds upon commercial and open source software packages exploiting a cluster of relational DBMS and Full Text Search Engines. The tests described in this paper performed a proof-of-concept and an initial scalability analysis in order to investigate the basic interplay among the system components and the overall system scalability. Albeit the usage of un-optimized open source software, most notably Lucene for document indexing and search as well as available text extraction components, we could observe a linear scalability behavior for simple ingest and search functionality. We identified various areas to enhance the current distributed processing. This provides opportunities to further enhance system performance up to a next order of magnitude. The next iteration of our prototype will introduce peer-to-peer technology (e.g. a distributed hash table) to further enhance scalability and performance. As a backend to store and manage the archived artifacts we are integrating a commercial ECM system.

Trademarks. The following terms are trademarks or registered trademarks of the International Business Machines Corporation in the United States, other countries or both: IBM, DB2, BladeCenter

Java and JavaMail are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft is trademark of Microsoft Corporation in the United States, other countries, or both.

Intel Xeon is trademark or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

References

1. Bace, J., Logan, D.: The Costs and Risks of E-discovery in Litigation. Gartner (December 1, 2005)
2. Base One: Database Scalability - Dispelling myths about the limits of database-centric architecture (Retrieved 2008-02-20), <http://www.boic.com/scalability.htm>
3. Barlas, D., Vahidy, T.: The Email Management Crisis. White paper, Iron Mountain Inc. (January 24, 2006)
4. Brewer, E.: Combining systems and databases: A search engine retrospective. In: Stonebraker, M., Hellerstein, J. (eds.) Readings in Database Systems, 4th edn. MIT Press, Cambridge (2004)
5. Chaudhuri, S., Dayal, U., Yan, T.W.: Join queries with external text sources: Execution and optimization techniques. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, pp. 410–422 (1995)
6. Chen, K.: IBM DB2 content manager v8 implementation on DB2 universal database: A primer. Technical report, IBM (2003)
7. Churchill, B., Clark, L., Rosenoer, J., von Bulow, F.: The impact of electronically stored information on corporate legal and compliance management: An IBM point of view. White paper, IBM Corporation (October 2006)
8. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI 2004: 6th Symposium on Operating Systems Design and Implementation (2004)
9. DiCenzo, C., Chin, K.: Magic Quadrant for E-Mail Active Archiving. Gartner (2007)
10. Ghodsi: Distributed k-ary System: Algorithms for Distributed Hash Tables. Doctoral thesis, KTH - Royal Institute of Technology (2006)
11. Hausheer, D., Stiller, B.: Design of a distributed P2P-based content management middleware. In: Proceedings of the 29th Euromicro Conference, pp. 173–180 (2003)
12. Manoel, E., Horkan, P., Parziale, L.: Dynamic Provisioning of SAP Environments using IBM Dynamic Infrastructure for MySAP and IBM Tivoli Provisioning Manager. IBM Redbooks Paper (October 2005)
13. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429. Springer, Heidelberg (2002)
14. Mega, C., Wagner, F., Mitschang, B.: From Content Management to Enterprise Content Management. In: Datenbanksysteme in Business, Technologie und Web (BTW) (2005)
15. Michael, M., Moreira, J.E., Shiloach, D., Wisniewski, R.W.: Scale-up x Scale-out: A Case Study using Nutch/Lucene. In: Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium (March 2007)
16. Moreira, J.E., Michael, M.M., Da Silva, D., Shiloach, D., Dube, P.: Scalability of the Nutch search engine. In: Proceedings of the 21st annual international conference on Supercomputing (ICS 2007) (June 2007)
17. Plotkin, J.: E-mail discovery in civil litigation: Worst case scenario vs. best practices. White paper by KVault Software Plc., (April 2004)

18. The Radicati Group: Taming the Growth of Email – An ROI Analysis. White Paper by The Radicati Group, Inc. (2005)
19. The Radicati Group: An Overview of the Archiving Market and Jatheon Technologies by The Radicati Group, Inc. (September 2006)
20. Thickins, G.: Compliance: Do no evil – critical implications and opportunities for storage. *Byte and Switch Insider* 2(5) (2004)
21. U. S. Department of the Interior: It's in the mail: Common questions about electronic mail and official records (2006)
22. Werelius, T.: Trends in Email Archiving., *Computer World Storage Networking World Online* (August 21, 2006)
23. Yu, H., Moreira, J.E., Dube, P., I-hsin, C., Zhang, L.: Performance Studies of a Web-Sphere Application, Trade, in Scale-out and Scale-up Environments. In: *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, pp. 1–8 (2007)
24. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiawicz, J.D.: Tapesstry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications* 22 (2004)

Author Index

- Alfred, Rayner 2
- Bauer, Péter 14
- Bednárek, David 30
- Ben Hassine, Mohamed Ali 112
- Böhlen, Michael H. 168
- Cerepnalkoski, Darko 295
- Corral, Antonio 46
- Cuzzocrea, Alfredo 62
- Dzemyda, Gintautas 143
- Grossmann, Matthias 81
- Gurský, Peter 97
- Härder, Theo 246
- Hernáth, Zsolt 14
- Holze, Marc 127
- Hönle, Nicola 81
- Horváth, Zoltán 14
- Huang, Wei 215
- Hvasshovd, Svein-Olaf 153
- Ivanikovas, Sergėjus 143
- Jakimovski, Boro 295
- Kolltveit, Heine 153
- Kon-Popovska, Margita 295
- Krebs, Kathleen 305
- Liu, Jixue 262
- Lukichev, Maxim 279
- Mannila, Heikki 1
- Manolopoulos, Yannis 46
- Mayer, Gyula 14
- Mazeika, Arturas 168
- Medvedev, Viktor 143
- Mega, Cataldo 305
- Mitschang, Bernhard 81, 305
- Mokhtar, Hoda 184
- Nicklas, Daniela 81
- Ondreička, Matúš 199
- Ounelli, Habib 112
- Parragi, Zsolt 14
- Pokorný, Jaroslav 199
- Porkoláb, Zoltán 14
- Ray, Indrakshi 215
- Revesz, Peter 231
- Ribeiro, Leonardo 246
- Ritter, Norbert 127, 305
- Shahriar, Md. Sumon 262
- Soldak, Yury 279
- Su, Jianwen 184
- Sztupák, Zsolt 14
- Torres, Manuel 46
- Triplet, Thomas 231
- Trivellato, Daniel 168
- Vassilakopoulos, Michael 46
- Velinov, Goran 295
- Vojtáš, Peter 97
- Wagner, Frank 305