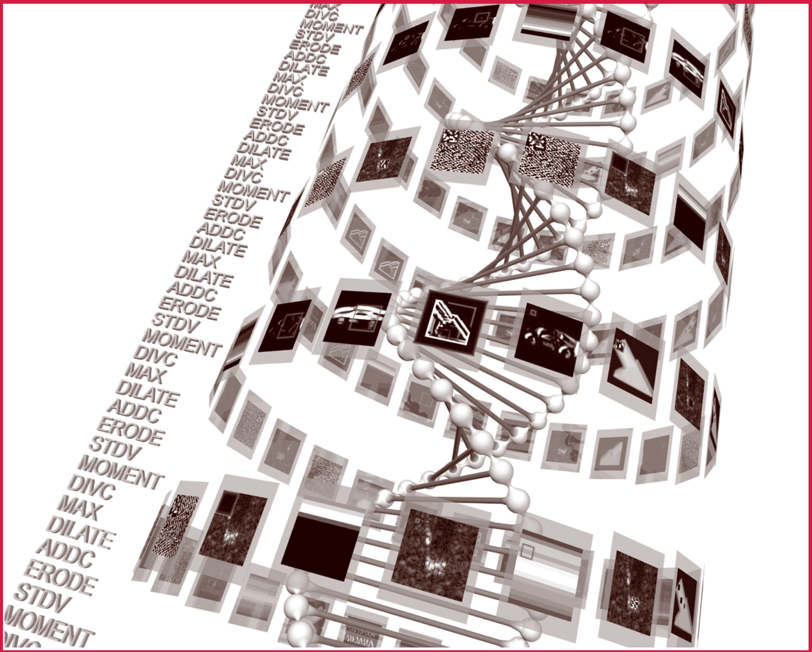
MONOGRAPHS IN COMPUTER SCIENCE

EVOLUTIONARY SYNTHESIS OF PATTERN RECOGNITION SYSTEMS

Bir Bhanu
Yingqiang Lin
Krzysztof Krawiec



Springer

Evolutionary Synthesis of Pattern Recognition Systems

Bir Bhanu
Yingqiang Lin
Krzysztof Krawiec

Evolutionary Synthesis of Pattern Recognition Systems

Springer

Bir Bhanu
Center for Research in
Intelligent Systems
University of California
at Riverside
Bourns Hall RM B232
Riverside, CA 92521

Yingqiang Lin
Center for Research in
Intelligent Systems
University of California
at Riverside
Bourns Hall RM B232
Riverside CA 92521

Krzysztof Krawiec
Center for Research in
Intelligent Systems
University of California
at Riverside
Bourns Hall RM B232
Riverside CA 92521

Series Editors
David Gries
Dept. of Computer Science
Cornell University
Upson Hall
Ithaca NY 14853-7501

Fred B. Schneider
Dept. Computer Science
Cornell University
Upson Hall
Ithaca NY 14853-7501

Library of Congress Cataloging-in-Publication Data
Bhanu, Bir.
Evolutionary Synthesis of Pattern Recognition Systems /Bir Bhanu, Yingqiang Lin, and Krzysztof Krawiec.
p. cm. –(Monographs in Computer Science)
Includes bibliographic references and index.

ISBN 0-387-21295-7 e-ISBN 0-387-24452-2 Printed on acid-free paper.

Printed in the United States of America. (BS/DH)

9 8 7 6 5 4 3 2 1 SPIN (HC) 10984741 / SPIN (eBK) 11381136

springeronline.com

Contents

List of Figures

Chapter 7

List of Tables

Chapter 4

Preface

Designing object detection and recognition systems that work in the real world is a challenging task due to various factors including the high complexity of the systems, the dynamically changing environment of the real world and factors such as occlusion, clutter, articulation, and various noise contributions that make the extraction of reliable features quite difficult. Furthermore, features useful to the detection and recognition of one kind of object or in the processing of one kind of imagery may not be effective in the detection and recognition of another kind of object or in the processing of another kind of imagery. Thus, the detection and recognition system often needs thorough overhaul when applied to other types of images different from the one for which the system was designed. This is very uneconomical and requires highly trained experts. The purpose of incorporating learning into the system design is to avoid the time consuming process of feature generation and selection and to lower the cost of building object detection and recognition systems.

Evolutionary computation is becoming increasingly important for computer vision and pattern recognition fields. It provides a systematic way of synthesis and analysis of object detection and recognition systems. With learning incorporated, the resulting recognition systems will be able to automatically generate new features on the fly and cleverly select a good subset of features according to the type of objects and images to which they are applied. The system will be flexible and can be applied to a variety of objects and images.

This book investigates evolutionary computational techniques such as genetic programming (GP), linear genetic programming (LGP), coevolutionary genetic programming (CGP) and genetic algorithms (GA) to automate the synthesis and analysis of object detection and recognition systems. The ultimate goal of the learning approaches presented in this book is to lower the cost of designing object detection and recognition systems and build more robust and flexible systems with human-competitive performance.

The book presents four important ideas.

First, this book shows the efficacy of GP and CGP in synthesizing effective composite operators and composite features from domain-independent primitive image processing operations and primitive features (both elementary and complex) for object detection and recognition. It explores the role of domain knowledge in evolutionary computational techniques for object recognition. Based on GP and CGP's ability to synthesize effective features from simple features not specifically designed for a particular kind of imagery, the cost of building object detection and recognition systems is lowered and the flexibility of the systems is increased. More importantly, a large amount of unconventional features are explored by GP and CGP and these unconventional features yield exceptionally good detection and recognition performance in some cases, overcoming the human experts' limitation of considering only a small number of conventional features.

Second, smart crossover, smart mutation and a new fitness function based on the minimum description length (MDL) principle are designed to improve the efficiency of genetic programming. Smart crossover and smart mutation are designed to identify and keep the effective components of composite operators from being disrupted and a MDL-based fitness function is proposed to address the well-known code bloat problem of GP without imposing severe restriction on the GP search. Compared to normal GP, smart GP algorithm with smart crossover, smart mutation and a MDL-based fitness function finds effective composite operators more quickly and the composite operators learned by smart GP algorithm have smaller size, greatly reducing both the computational expense during testing and the possibility of overfitting during training.

Third, a new MDL-based fitness function is proposed to improve the genetic algorithm's performance on feature selection for object detection and recognition. The MDL-based fitness function incorporates the number of features selected into the fitness evaluation process and prevents GA from selecting a large number of features to overfit the training data. The goal is to select a small set of features with good discrimination performance on both training and unseen testing data to reduce the possibility of overfitting the training data during training and the computational burden during testing.

Fourth, adaptive coevolutionary linear genetic programming (LGP) in conjunction with general image processing, computer vision and pattern recognition operators is proposed to synthesize recognition systems. The basic two-class approach is extended for scalability to multiple classes and various architectures and strategies are considered.

The book consists of eight chapters dealing with various evolutionary approaches for automatic synthesis and analysis of object detection and recognition systems. Many real world imagery examples are given in all the chapters and a comparison of the results with standard techniques is provided.

The book will be of interest to scientists, engineers and students working in computer vision, pattern recognition, object recognition, machine learning, evolutionary learning, image processing, knowledge discovery, data mining, cybernetics, robotics, automation and psychology.

Authors would like to thank Ken Grier, Dale Nelson, Lou Tamburino, and Bob Herklotz for their guidance and support. Many discussions held with Ed Zelnio, Tim Ross, Vince Velten, Gregory Power, Devert Wicker, Grinnell Jones, and Sohail Nadimi were very helpful.

The work covered in this book was performed at the University of California at Riverside. It was partly supported by funding from Air Force Research Laboratory during the last four years. Krzysztof Krawiec was at the University of California at Riverside on a temporary leave from Poznan University of Technology, Poznan, Poland. He would like to acknowledge the support from the Scientific Research Committee, Poland (KBN). Authors would like to thank Julie Vu and Lynne Cochran for their secretarial support.

Riverside, California
November 2004

Bir Bhanu
Yingqiang Lin
Krzysztof Krawiec

Chapter 1

INTRODUCTION

In recent years, with the advent of newer, much improved and inexpensive imaging technologies and the rapid expanding of the Internet, more and more images are becoming available. Recent developments in image collection platforms produce far more imagery than the declining ranks of image analysts are capable of handling due to human work load limitations. Relying on human image experts to perform image analysis, processing and classification becomes more and more unrealistic. Building object detection and recognition systems to take advantage of the speed of computer is a viable and important solution to the increasing need of processing a large quantity of images efficiently.

1.1 Object Detection and Recognition Problem

The object detection and recognition problem is one of the most important research areas in pattern recognition and computer vision [7], [18]. It has wide range of applications in surveillance, reconnaissance, object and target recognition, autonomous navigation, remote sensing, manufacturing automation, etc. The major task of object detection is to locate and extract regions that may contain objects in an image. It is an important intermediate step to object recognition. The extracted regions are called regions-of-interest (ROIs) or object chips. ROI extraction is very important to object recognition, since the size of an image is usually large, leading to the heavy computational burden of processing the whole image. By extracting ROIs, the computational cost of object recognition is greatly reduced, thus improving the recognition efficiency. This advantage is particularly useful to real-time applications, where the recognition speed is of prime importance. Also, by extracting ROIs, the recognition system can focus on the extracted regions that may contain potential objects and this can be very helpful in improving the recognition accuracy. Generally, the extracted ROIs are identical to their corresponding regions in the original image, but sometimes, they may be images that result from applying some image processing operations to the corresponding regions in the original image. No matter what ROIs are, they are passed to an object recognition module for further processing. Usually, in order to increase the probability of object detection, some false alarm ROIs, which do not contain an object, but some natural or man-made clutter, are allowed to pass object detection phase.

The task of object recognition is first to reject the false alarm ROIs and then recognize the kinds of objects in the ROIs containing them. It is actually a signal-to-symbol problem of labeling perceived signals with one or more symbols. A solution to this problem takes images or the features extracted from images as input and outputs one or more symbols which are the labels of the objects in the images. Sometimes, the symbols may further represent the pose of the objects or the relations between different objects. These symbols are intended to capture some useful aspects of the input and in turn, permit some high level reasoning on the perceived signals.

It is well known that automatic object detection and recognition is really not an easy task. The quality of detection and recognition is heavily dependent on the kind and quality of features extracted from the image, and it also highly relies on the representation of an object based on the extracted features. The features used to represent an object are the key to object detection and recognition. If useful features with good quality are unavailable to build an efficient representation of an object, good detection and recognition results cannot be achieved no matter what detection and recognition algorithms are used. However, in most real images, there is always some noise, making the extraction of features difficult. More importantly, since there are many kinds of features that can be extracted, so what are the appropriate features for the current detection and recognition task or how to synthesize composite features particularly useful to the detection and recognition from the primitive features extracted from an image? There is no easy answer to these questions and the solutions are largely dependent on the intuitive instinct, knowledge, previous experience and even the bias of human image experts. Object detection and recognition in many real-world applications is still a challenging problem and needs further research.

1.2 Motivations for Evolutionary Computation

In the past, object detection and recognition systems are manually developed and maintained by human experts. The traditional approach requires a human expert to select or synthesize a set of features to be used in detection and recognition. However, handcrafting a set of features requires human ingenuity and insight into the objects to be detected and recognized since it is very difficult to identify a set of features that characterize a complex set of objects. Typically, many features are explored before object detection and recognition systems can be built. There are a lot of features available and these features may be correlated. To select a set of features which, when acting cooperatively, can give good performance is very time consuming and expensive. Sometimes, simple features (also called primitive features) directly extracted from images may not be effective in detecting and recognizing objects. At this point, synthesizing composite features useful for the current detection and recognition task from those simple ones becomes imperative.

Traditionally, it is the human experts who synthesize features to be used. However, based on their knowledge, previous experience and limited by their bias and speed, human experts only consider a small number of conventional features and many unconventional features are totally ignored. Sometimes it is those unconventional features that yield very good detection and recognition performance. Furthermore, after the features are selected or designed by human experts and incorporated into a system, they are fixed. The features used by the system are pre-determined and the system cannot generate new features useful to the current detection and recognition task on the fly based on the already available features, leading to inflexibility of the system. Features useful to the detection and recognition of one kind of object or in the processing of one kind of imagery may not be effective in the detection and recognition of another kind of object or in the processing of another kind of imagery. Thus, the detection and recognition system often needs thorough overhaul when applied to other types of images that are different from the one when the system was devised. This is very uneconomical.

Synthesizing effective new features from primitive features is equivalent to finding good points in the feature combination space where each point represents a combination of primitive features. Similarly, selecting an effective subset of features is equivalent to finding good points in the feature subset space where each point represents a subset of features. The feature combination space and feature subset space are huge and complicated and it is very difficult to find good points in such vast spaces unless one has an efficient search algorithm.

Hill climbing, gradient descent and simulated annealing (also called stochastic hill climbing) are widely used search algorithms. Hill climbing and gradient descent are efficient in exploring a unimodal space, but they are not suitable for finding global optimal points in a multi-modal space due to their high probability of being trapped in local optima. Thus, if the search space is a complicated and multi-modal space, they are unlikely to yield good search results. Simulated annealing has the ability to jump out of local optimal points, but it is heavily dependent on the starting point. If the starting point is not appropriately placed, it takes a long time, or even could be impossible, for simulated annealing to reach good points. Furthermore, in order to apply a simulated annealing algorithm, the neighborhood of a point must be defined and the neighboring points should be somewhat similar. This requires some knowledge about the search space and it also requires some smoothness of the search space.

It is very difficult, if not impossible, to define the neighborhood of a point in the huge and complicated feature combination and feature subset spaces, since similar feature combinations and similar feature subsets may have very different object detection and recognition performance. Due to the lack of knowledge about these search spaces, a variety of genetic programming techniques and genetic algorithms [6], [36], [57], [58], [66] are employed in this book. In order to apply GP and GA, all that needs to be known are how to define individuals, how to define crossover and mutation operations on the individuals and how to evaluate individuals. GP and GA are very much capable of exploring huge complicated multi-modal spaces with unknown structures. Maintaining a large population of individuals as multiple searching points, GP and GA explore the search spaces along different directions concurrently. With multiple searching points and the crossover and mutation operations' ability to immediately move a searching point from one portion of the search space to another faraway portion, GP and GA are less likely to be trapped at local optimal points. All these characteristics greatly enhance the probability of finding global optimal points, although they cannot guarantee the finding of global optima. It is to be noted that GP and GA are not random search algorithms, they are guided by the fitness of the individuals in the population. As search proceeds, the population is gradually adapted to the portion of the search space containing good points.

1.3 Evolutionary Approaches for Synthesis and Analysis

In this book, the techniques necessary for automatic design of object detection and recognition systems are investigated. Here, the object detection and recognition system itself is the theme and the efficacy of evolutionary learning algorithms such as genetic programming and genetic algorithm in the feature generation and selection is studied. The advantage of incorporating learning is to avoid the time consuming process of feature selection and generation and to automatically explore many unconventional features. The system resulting from the learning is able to automatically generate features on the fly and cleverly select a good subset of features according to the type of object and image to which it is applied. The system should be somewhat flexible and can be applied to a variety of objects and images. The goal is to lower the cost of designing object detection and recognition systems and build more robust and flexible systems with human-competitive performance.

This book investigates evolutionary computational techniques such as genetic programming (GP), coevolutionary genetic programming (CGP), linear genetic programming (LCP) and genetic algorithm (GA) to automate the synthesis and analysis of object detection and recognition systems.

First, this book shows the efficacy of GP and CGP in synthesizing effective composite operators and composite features from domain-independent primitive image processing operations and primitive features for object detection and recognition. It explores the role of domain knowledge in evolutionary computation. Based on GP and CGP's ability to synthesize effective features from simple features not specifically designed for a particular kind of imagery, the cost of building object detection and recognition systems is lowered and the flexibility of the systems is increased. More importantly, it shows that a large amount of unconventional features are explored by GP and CGP and these unconventional features yield exceptionally good detection and recognition performance in some cases, overcoming the human experts' limitation of considering only a small number of conventional features.

Second, smart crossover, smart mutation and a new fitness function based on minimum description length (MDL) principle are designed to improve the efficiency of genetic programming. Smart crossover and smart mutation are designed to identify and keep the effective components of composite operators from being disrupted and a MDL-based fitness function is proposed to address the well-known code bloat problem of GP without imposing severe restriction on the GP search. Compared to normal GP, a smart GP algorithm with smart crossover, smart mutation and a MDL-based fitness function finds effective composite operators more quickly and the composite operators learned by a smart GP algorithm have smaller size, greatly reducing both the computational expense during testing and the possibility of overfitting during training.

Third, a new MDL-based fitness function is proposed to improve the genetic algorithm's performance on feature selection for object detection and recognition. The MDL-based fitness function incorporates the number of features selected into the fitness evaluation process and prevents GA from selecting a large number of features to overfit the training data. The goal is to select a small set of features with good discrimination performance on both training and unseen testing data to reduce both the possibility of overfitting the training data during training and the computational burden during testing.

Fourth, linear genetic programming (LGP) and coevolutionary genetic programming (CGP) techniques are used to synthesize a feature extraction procedure (FEP) to generate features for object recognition. FEP consists of a sequence of instructions, which are primitive image processing operators that are executed sequentially one after another. Each instruction in a FEP is composed of an opcode determining the operator to be used and arguments referring to registers from which to fetch the input data and to which to store the result of the instruction. LGP is a variety of GP with simplified, linear representation of individuals and it is a hybrid of GA and GP and combines their advantages. LGP is similar to GP in the sense that each individual actually contains a sequence of interrelated operators. On the other hand, a FEP has a fixed number of instructions and an instruction is encoded into a fixed-length binary string at the genome level, which is essentially equivalent to GA representation. LGP encoding is, therefore, more positional and more resistant to destructive crossovers. When CGP is applied, the problem of feature construction can be decomposed at different levels. We explore decomposition at the instruction, feature, class and decision levels. Our experiments show the superiority of decomposition at the instruction level. With different segments of a FEP evolved by sub-populations of CGP, a better FEP can be synthesized by concatenating the segments from sub-populations. The benefits we expect from the decomposition of feature construction by CGP include faster convergence of the learning process, better scalability of the learning with respect to the problem size and better understanding of the obtained solutions.

1.4 Outline of the Book

The outline of the book is as follows:

Chapter 1 is the introduction. It describes object detection and recognition problems, provides motivation and advantages of incorporating evolutionary computation in the design of object detection and recognition systems.

Chapter 2 discusses synthesizing composite features for object detection. Genetic programming (GP) is applied to the learning of composite features based on primitive features and primitive image processing operations. The primitive features and primitive image processing operations are domain-independent, not specific to any kind of imagery so that the proposed feature synthesis approach can be applied to a wide variety of images.

Chapter 3 concentrates on improving the efficiency of genetic programming. A fitness function based on the minimum description length (MDL) principle is proposed to address the well-known code bloat problem of GP while at the same time avoiding severe restriction on the GP search. The MDL fitness function incorporates the size of a composite operator into the fitness evaluation process to prevent it from growing too large, reducing possibility of overfitting during training and the computational expenses during testing. The smart crossover and smart mutation are proposed to identify the effective components of a composite operator and keep them from being disrupted by subsequent crossover and mutation operations to further improve the efficiency of GP.

In Chapter 4, genetic algorithms (GA) are used for feature selection for distinguishing objects from natural clutter. Usually, GA is driven by a fitness function based on the performance of selected features. To achieve excellent performance during training, GA may select a large number of features. However, a large number features with excellent performance on training data may not perform well on unseen testing data due to the overfitting. Also, selecting more features means heavier computational burden during testing. In order to overcome this problem, an MDL-based fitness function is designed to drive GA. With MDL-based function incorporating the number of features selected into the fitness evaluation process, a small set of features is selected to achieve satisfactory performance during both training and testing.

Chapter 5 presents a method of learning composite feature vectors for object recognition. Coevolutionary genetic programming (CGP) is used to synthesize composite feature vectors based on the primitive features (simple or relatively complex) directly extracted from images. The experimental results using real SAR images show that CGP can evolve composite features that are more effective than the primitive features upon which they are built.

Chapter 6 presents a coevolutionary approach for synthesizing recognition systems using linear genetic programming (LGP). It provides a rationale for the design of the method and outlines main differences in comparison to standard genetic programming. The basic characteristic of LGP approach is the linear (sequential) encoding of elementary operations and passing of intermediate arguments through temporary variables (registers). Two variants of of the approach are presented. The first approach called, evolutionary feature programming (EFP), engages standard single-population evolutionary computation. The second approach called, coevolutionary feature programming (CFP), decomposes feature synthesis problem using cooperative coevolution. Various decomposition strategies for breaking up the feature synthesis process are discussed.

Chapter 7 presents experimental results of applying the methodology described in chapter 7 to real-world computer vision/pattern recognition problems. It includes experiments using single-population evolutionary feature programming (EFP), and selected variants of coevolutionary feature programming (CFP) cooperating at different decomposition levels. To provide experimental evidence for the generality of the proposed approach, it is verified on two different real-world tasks. First of them is the recognition of common household objects in controlled lighting conditions, using the widely known COIL-20 benchmark database. The second application is much more difficult and concerns the recognition of different types of vehicles in synthetic aperture radar (SAR) images.

Finally, Chapter 8 provides the conclusions and future research directions.

Chapter 2

FEATURE SYNTHESIS FOR OBJECT DETECTION

2.1 Introduction

Designing automatic object detection and recognition systems is one of the important research areas in computer vision and pattern recognition [7], [35]. The major task of object detection is to locate and extract regions of an image that may contain potential objects so that the other parts of the image can be ignored. It is an intermediate step to object recognition. The regions extracted during detection are called regions-of-interest (ROIs). ROI extraction is very important in object recognition, since the size of an image is usually large, leading to the heavy computational burden of processing the whole image. By extracting ROIs, the recognition system can focus on the extracted regions that may contain potential objects and this can be very helpful in improving the recognition rate. Also by extracting ROIs, the computational cost of object recognition is greatly reduced, thus improving the recognition speed. This advantage is particularly important for real-time applications, where the recognition accuracy and speed are of prime importance.

However, the quality of object detection is dependent on the type and quality of features extracted from an image. There are many features that can be extracted. The question is what are the appropriate features or how to synthesize features, particularly useful for detection, from the primitive features extracted from images. The answer to these questions is largely dependent on the intuitive instinct, knowledge, previous experience and even the bias of algorithm designers and experts in object recognition.

In this chapter, we use genetic programming (GP) to synthesize composite features which are the output of composite operators, to perform object detection. A composite operator consists of primitive operators and it can be viewed as a way of combining primitive operations on images. The basic approach is to apply a composite operator on the original image or primitive feature images generated from the original one; then the output image of the composite operator, called composite feature image, is segmented to obtain a binary image or mask; finally, the binary mask is used to extract the region containing the object from the original image. The individuals in our GP based learning are composite operators represented by binary trees whose internal nodes represent the pre-specified primitive operators and the leaf nodes represent the original image or the primitive feature images. The primitive feature images are pre-defined, and they are not the output of the pre-specified primitive operators.

This chapter is organized as follows: chapter 2.2 provides motivation, related research and contribution of this chapter; chapter 2.3 provides the details of genetic programming for feature synthesis; chapter 2.4 presents experimental results using synthetic aperture radar (SAR), infrared (IR) and color images. Various comparisons are given in this section to demonstrate the effectiveness of the approach, including examples of two-class and multi-class imagery; finally, chapter 2.5 provides the conclusions of this chapter.

2.2 Motivation and Related Research

2.2.1 Motivation

In most imaging applications, human experts design an approach to detect potential objects in images. The approach can often be divided into some primitive operations on the original image or a set of related feature images obtained from the original one. It is the expert who, relying on his/her experience, figures out a smart way to combine these primitive operations to achieve good detection results. The task of synthesizing a good approach is equivalent to finding a good point in the space of *composite operators* formed by the combination of primitive operators.

Unfortunately, the ways of combining primitive operators are infinite. The human expert can only try a very limited number of conventional combinations. However, a GP may try many unconventional ways of combining primitive operations that may never be imagined by a human expert. Although these unconventional combinations are very difficult, if not impossible, to be explained by domain experts, in some cases, it is these unconventional combinations that yield exceptionally good results. The unlikeliness, and even incomprehensibility of some effective solutions learned by GP demonstrates the value of GP in the generation of new features for object detection. The inherent parallelism of GP and the high speed of current computers allow the portion of the search space explored by GP to be much larger than that by human experts. The search performed by GP is not a random search. It is guided by the fitness of composite operators in the population. As the search proceeds, GP gradually shifts the population to the portion of the space containing good composite operators.

2.2.2 Related research

Genetic programming, an extension of genetic algorithm, was first proposed by Koza [55], [56], [57], [58] and has been used in image processing, object detection and object recognition. Harris and Buxton [39] applied GP to the production of high performance edge detectors for 1-D signals and image profiles. The method is also extended to the development of practical edge detectors for use in image processing and machine vision. Poli [92] used GP to develop effective image filters to enhance and detect features of interest and to build pixel-classification-based segmentation algorithms. Bhanu and Lin [14], [17], [21], [69] used GP to learn composite operators for object detection. Their experimental results showed that GP is a viable way of synthesizing composite operators from primitive operations for object detection. Stanhope and Daida [114] used GP to generate rules for target/clutter classification and rules for the identification of objects. To perform these tasks, previously defined feature sets are generated on various images and GP is used to select relevant features and methods for analyzing these features. Howard et al. [44] applied GP to automatic detection of ships in low-resolution SAR imagery by evolving detectors. Roberts and Howard [103] used GP to develop automatic object detectors in infrared images. Tackett [115] applied GP to the development of a processing tree for the classification of features extracted from images.

Belpaeme [5] investigated the possibility of evolving feature detectors under selective pressure. His experimental results showed that it is possible for GP to construct visual functionality based on primitive image processing functions inspired by visual behavior observed in mammals. The inputs for the feature detectors are images. Koppen and Nickolay [54] presented a special 2-D texture filtering framework, based on the so-called 2-D-Lookup with its configuration evolved by GP that allowed representing and searching a very large number of texture filters. Their experimental results demonstrated that although the framework may never find the globally optimal texture filters, it evolves the initialized solutions toward better ones. Johnson et al. [50] described a way of automatically evolving visual routines for simple tasks by using genetic programming. The visual routine models used in their work were initially proposed by Ullman [121] to describe a set of primitive routines that can be applied to find spatial relations between objects in an input image. Ullman proposed, that given a specific task, the visual routine processor compiled and organized an appropriate set of visual routines and applied it to a base representation. But as Johnson et al. [50] pointed out, Ullman did not explain how routines were developed, stored, chosen and applied. In their work, Johnson et al. [50] applied typed genetic programming to the problem of creating visual routines for the simple task of locating the left and right hands in a silhouette image of a person. In their GP, crossover was performed by exchanging between two parents the subtrees of the same root return type. To avoid the code bloat problem of GP, they simply canceled a particular crossover if it would produce an offspring deeper than the maximum allowable depth. Rizki et al. [102] use hybrid evolutionary computation (genetic programming and neural networks) for target recognition using 1-D radar signals.

Unlike the prior work of Stanhope and Daida [114], Howard et al. [44] and Roberts and Howard [103], the input and output of each node of a tree in the system described in this chapter are images, not real numbers. When the data from node to node is an image, the node can contain any primitive operation on images. Such image operations do not make sense when the data is a real number. In our system, the data to be processed are images, and image operations can be applied to primitive feature images and any other intermediate images to achieve object detection results. In [114], [44], [103], image operations can only be applied to the original image to generate primitive feature images. Also, the primitive features defined in this chapter are more general and easier to compute than those used in [114], [44]. Unlike our previous work [17], in this chapter the hard limit of composite operator size is removed and a soft size limit is used to let GP search more freely while at the same time preventing the code-bloat problem. The training in this chapter is not performed on a whole image, but on the selected regions of an image and this is very helpful in reducing the training time. Of course, training regions must be carefully selected and represent the characteristics of training images [11]. Also, two types of mutation are added to further increase the diversity of the population. Finally, more primitive feature images are employed. The primitive operators and primitive features designed in this chapter are very basic and domain-independent, not specific to a kind of imagery. Thus, this system and methodology can be applied to a wide variety of images. For example, results are shown here using synthetic aperture radar (SAR), infrared (IR) and color video images.

2.3 Genetic Programming for Feature Synthesis

In our GP based approach, individuals are composite operators represented by binary trees. The search space of GP is huge and it is the space of all possible composite operators. Note that there could be equivalent composite operators in terms of their output images. In the computer system, a pixel of an image can assume only finite values, the number of possible images is finite, but this number is huge and astronomical. Also, if we set a maximum composite operator size, the number of composite operators is also finite, but again this number is also huge and astronomical. To illustrate this, consider only a special kind of binary tree, where each tree has exactly one leaf node and 30 internal nodes and each internal node has only one child. For 17 primitive operators and only one primitive feature image, the total number of such trees is 17^{30}. It is extremely difficult to find good composite operators from this vast space unless one has a smart search strategy.

2.3.1 Design considerations

There are five major design considerations, which involve: determining the set of terminals; the set of primitive operators; the fitness measure; the parameters for controlling the evolutionary run; and the criterion for terminating a run.

- **The set of terminals:** The set of terminals used in this chapter are sixteen primitive feature images generated from the original image: the first one is the original image; the others are mean, deviation, maximum, minimum and median images obtained by applying templates of sizes 3×3, 5×5 and 7×7, as shown in Table 2.1. These images are the input to composite operators. GP determines which operations are applied on them and how to combine the results. To get the mean image, we translate a template across the original image and use the average pixel value of the pixels covered by the template to replace the pixel value of the pixel covered by the central cell of the template. To get the deviation image, we just compute the pixel value difference between the pixel in the original image and its corresponding pixel in the mean image. To get maximum, minimum and median images, we translate the template across the original image and use the maximum, minimum and median pixel values of the pixels covered by the template to replace the pixel value of the pixel covered by the central cell of the template, respectively.

Table 2.1. Sixteen primitive feature images used as the set of terminals.

No.	Primitive feature image	Description	No.	Primitive feature image	Description
0	PFIM0	Original image	8	PFIM8	5×5 maximum image
1	PFIM1	3×3 mean image	9	PFIM9	7×7 maximum image
2	PFIM2	5×5 mean image	10	PFIM10	3×3 minimum image
3	PFIM3	7×7 mean image	11	PFIM11	5×5 minimum image
4	PFIM4	3×3 deviation image	12	PFIM12	7×7 minimum image
5	PFIM5	5×5 deviation image	13	PFIM13	3×3 median image
6	PFIM6	7×7 deviation image	14	PFIM14	5×5 median image
7	PFIM7	3×3 maximum image	15	PFIM15	7×7 median image

- **The set of primitive operators:** A primitive operator takes one or two input images, performs a primitive operation on them and stores the result in a resultant image. Currently, 17 primitive operators are used by GP to form composite operators, as shown in Table 2.2, where A and B are input images of the same size and c is a constant (ranging from –20 to 20) stored in the primitive operator. For operators such as ADD, SUB, MUL, etc., that take two images as input, the operations are performed on the pixel-by-pixel basis. In the operators MAX, MIN, MED, MEAN and STDV, a 3×3, 5×5 or 7×7 neighborhood is used with equal probability. Operator 16 (MEAN) can be considered as a kind of convolution for low pass filtering and operator 17 (STDV) is a kind of convolution for high pass filtering. Operators 13 (MAX), 14 (MIN) and 15 (MED) can also be considered as convolution operators. We do not include edge operators for several reasons. *First*, these operators are not primitive and we want to investigate if GP can synthesize effective composite operators or features from simple and domain-independent operations. This is important since without relying on domain knowledge, we can examine the power of a learning algorithm when applied to a variety of images. *Second*, edge detection operators can be dissected into the above primitive operators and it is possible for GP to synthesize edge operators or composite operators approximating them if they are very useful to the current object detection task. *Finally*, the primitive operator library is decoupled from the GP learning system. Edge detection operators can be added in the primitive operator library if they are absolutely needed by the current object detection task.

Some operations used to generate feature images are the same as some primitive operators (see Table 2.1 and Table 2.2), but there are some differences. Primitive feature images are generated from original images, so the operations generating primitive feature images are applied to an original image. A primitive operator is applied to a primitive feature image or to an intermediate image output that is generated by the child node of the node containing this primitive operator. In short, the input image of a primitive operator varies.

Table 2.2. Seventeen primitive operators.

No.	Operator	Description
1	ADD (A, B)	Add images A and B.
2	SUB (A, B)	Subtract image B from A.
3	MUL (A, B)	Multiply images A and B.
4	DIV (A, B)	Divide image A by image B (If the pixel in B has value 0, the corresponding pixel in the resultant image takes the maximum pixel value in A).
5	MAX2 (A, B)	The pixel in the resultant image takes the larger pixel value of images A and B.
6	MIN2 (A, B)	The pixel in the resultant image takes the smaller pixel value of images A and B.
7	ADDC (A)	Increase each pixel value by c.
8	SUBC (A)	Decrease each pixel value by c.
9	MULC (A)	Multiply each pixel value by c.
10	DIVC (A)	Divide each pixel value by c.
11	SQRT (A)	For each pixel with value v, if v ≥ 0, change its value to $\sqrt{v}$. Otherwise, to $-\sqrt{-v}$.
12	LOG (A)	For each pixel with value v, if v ≥ 0, change its value to ln(v). Otherwise, to –ln(-v).
13	MAX (A)	Replace the pixel value by the maximum pixel value in a 3×3, 5×5 or 7×7 neighborhood.
14	MIN (A)	Replace the pixel value by the minimum pixel value in a 3×3, 5×5 or 7×7 neighborhood.
15	MED (A)	Replace the pixel value by the median pixel value in a 3×3, 5×5 or 7×7 neighborhood.
16	MEAN (A)	Replace the pixel value by the average pixel value of a 3×3, 5×5 or 7×7 neighborhood.
17	STDV (A)	Replace the pixel value by the standard deviation of pixels in a 3×3, 5×5 or 7×7 neighborhood.

• **The fitness measure:** It measures the extent to which the ground-truth and the extracted ROI overlap. The fitness value of a composite operator is computed in the following way. Suppose G and G' are foregrounds in the ground-truth image and the resultant image of the composite operator respectively. Let $n(X)$ denote the number of pixels within region X, then $Fitness = n(G \cap G') / n(G \cup G')$. The fitness value is between 0 and 1. If G and G' are completely separated, the value is 0; if G and G' are completely overlapped, the value is 1.

• **Parameters and termination:** The key parameters are: the population size M; the number of generations N; the crossover rate; the mutation rate; and the fitness threshold. The GP stops whenever it finishes the pre-specified number of generations or whenever the best composite operator in the population has fitness value greater than the fitness threshold.

2.3.2 Selection, crossover and mutation

GP searches through the space of composite operators to generate new composite operators, which may be better than the previous ones. By searching through the composite operator space, GP gradually adapts the population of composite operators from generation to generation and improves the overall fitness of the whole population. More importantly, GP may find an exceptionally good composite operator during the search. The search is done by performing selection, crossover and mutation operations [2], [71], [118]. The initial population is randomly generated and the fitness of each individual is evaluated.

• **Selection:** The selection operation involves selecting composite operators from the current population. In this chapter, we use tournament selection, where a number of individuals (in this case five) are randomly selected from the current population and the one with the highest fitness value is copied into the new population.

• **Crossover:** To perform crossover, two composite operators are selected on the basis of their fitness values. The higher the fitness value, the more likely the composite operator is selected for crossover. These two composite operators are called parents. One internal node in each of these two parents is randomly selected, and the two subtrees rooted at these two nodes are exchanged between the parents to generate two new composite operators, called offspring. The offspring are composed of subtrees from their parents. If two composite operators are somewhat effective in detection, then some of their parts probably have some merit. The reason that an offspring may be better than the parents is that recombining randomly chosen parts of somewhat effective composite operators may yield a new composite operator that is even more effective in detection.

It is easy to see that the size of one offspring (i.e., the number of nodes in the binary tree representing the offspring), may be greater than both parents. So if we do not control the size of composite operators when implementing crossover in this simple way, the sizes of composite operators will become larger and larger as GP proceeds. This is the well-known code bloat problem of GP. It is a very serious problem, since when the size becomes too large, it will take a long time to execute a composite operator, thus, greatly reducing the search speed of GP. Further, large-size composite operators may overfit the training data by approximating various noisy components of an image. Although the results on the training image may be very good, the performance on unseen testing images may be bad. Also, large composite operators take up a lot of computer memory. Due to the finite computer resources and the desire to achieve a good running speed (efficiency) of GP, we must limit the size of composite operator by specifying its maximum size. In our previous work [17], if the size of one offspring exceeds the *maximum size* allowed, the crossover operation is performed again until the sizes of both offspring are within the limit. Although this simple method guarantees that the size of composite operators does not exceed the size limit, it is a brutal method since it sets a *hard size limit*. The hard size limit may restrict the search performed by GP, since after randomly selecting a crossover point in one composite operator, GP cannot select some nodes of the other composite operator as a crossover point in order to guarantee that both offspring do not exceed the size limit. However, restricting the search may greatly reduce the efficiency of GP, making it less likely to find good composite operators.

One may suggest that after two composite operators are selected, GP may perform crossover twice and may each time keep the offspring of smaller size. This method can enforce the size limit and will prevent the sizes of offspring composite operators from growing large. However, GP will now only search the space of these smaller composite operators. With a small number of nodes, a composite operator may not capture the characteristics of objects to be detected. How to avoid restricting the GP search while at the same time prevent code-bloat is the key to the success of GP and it is still a subject of intensive research. The key is to find a balance between these two conflicting factors.

In this chapter, we set a composite operator size limit to prevent code-bloating, but unlike our previous work, the size limit is a *soft size limit*, so it restricts the GP search less severely than the hard size limit. With a soft size limit, GP can select any node in both composite operators as crossover points. If the size of an offspring exceeds the size limit, GP still keeps it and evaluates it later. If the fitness of this large composite operator is the best or very close to the fitness of the best composite operator in the population, it is kept by GP; otherwise, GP randomly selects one of its sub-trees of size smaller than the size limit to replace it in the population. In this chapter, GP discards any composite operator beyond the size limit unless it is the best one in the population. By keeping the effective composite operators exceeding the size limit, GP enhances the possibility of finding good composite operators, since good composite operators usually contain effective components (sub-trees) and these effective components are kept by the soft size limit and they may transfer to other composite operators during crossover. Also, by keeping some large composite operators, the size difference between composite operators in the population is widened and this is helpful in reducing the possibility of fitness bloat (in which an increasing number of redundant composite operators in the population evaluate to the same fitness value), although it cannot get rid of it. With a hard size limit, many composite operators in the population have size equal or very close to the hard size limit in the later generations of GP. This increases the possibility of fitness bloat. However, large composite operators kept by the soft size limit take a long time to execute and many of them have redundant branches. By getting rid of the redundant branches, we can reduce the size and running time of composite operators without degrading their performance. But, in order to identify the redundant branches, the fitness of each internal node has to be evaluated and this is a time-consuming process. Moreover, some redundant branches are effective components. They are redundant just because they are in an inhospitable context and their effect is cancelled by other nodes. Eliminating them does no good to the GP search since these effective components may go into other friendly composite operators via crossover operation. Also, composite operators with redundant branches are more resistant to destructive crossover and mutation. Without redundant branches, each part of a composite operator is important to its performance and breaking any component may have a major impact on the performance of the composite operator.

- **Mutation:** In order to avoid premature convergence, mutation is introduced to randomly change the structure of some individuals to maintain the diversity of the population. Composite operators are randomly selected for mutation. In this system, there are three types of mutation invoked with equal probability:

1. Randomly select a node of the binary tree representing the composite operator and replace the subtree rooted at this node, including the node selected, by a new randomly generated binary tree
2. Randomly select a node of the binary tree representing the composite operator and replace the primitive operator stored in the node with another primitive operator of the same arity as the replaced one. The replacing primitive operator is selected at random from all the primitive operators with the same arity as the replaced one.
3. Randomly select two subtrees within the composite operator and swap these two subtrees. Of course, neither of the two sub-trees can be the sub-tree of the other.

2.3.3 Steady-state and generational genetic programming

Both steady-state and generational genetic programming are used in this chapter. In *steady-state GP*, two parent composite operators are selected on the basis of their fitness for crossover. The children of this crossover replace a pair of composite operators with the smallest fitness values. The two children are executed immediately and their fitness values are recorded. Then another two parent composite operators are selected for crossover. This process is repeated until the crossover rate is satisfied. Finally, mutation is applied to the resulting population and the mutated composite operators are executed and evaluated. The above cycle is repeated from generation to generation. In *generational GP*, two composite operators are selected on the basis of their fitness values for crossover and generate two offspring. The two offspring are not put into the current population and do not participate in the following crossover operations on the current population. The above process is repeated until the crossover rate is satisfied. Then, mutation is applied to the composite operators in the current population and the offspring from crossover. After mutation is done, selection is applied to the current population to select some composite operators. The number of composite operators selected must meet the condition that after combining with the composite operators from crossover, we get a new population of the same size as the old one. Finally, combine the composite operators from crossover with those selected from the old population to get a new population and the next generation begins. In addition, we adopt an *elitism* replacement method that keeps the best composite operator from generation to generation. Figure 2.1 and Figure 2.2 show the pseudo code for steady-state and generational genetic programming algorithms, respectively.

Steady-state Genetic Programming Algorithm:

```
1.  randomly generate population P of size M and evaluate each composite
      operator in P.
2.    for gen = 1 to N do loop 1  // N is the number of generation.
3.      keep the best composite operator in P.
        repeat
4.        select 2 composite operators from P based on their fitness values for
            crossover through tournament selection.
5.        select 2 composite operators with the lowest fitness values in P for
            replacement.
6.        perform crossover operation and let the 2 offspring replace the 2
            composite operators selected for replacement.
7.        execute the 2 offspring and evaluate their fitness values.
        until crossover rate is met.
8.      perform mutation on each composite operator with probability of
          mutation rate and evaluate mutated composite operators.
        // After crossover and mutation, a new population P' is generated.
9.      let the best composite operator from population P replace the worst
          composite operator in P' and let P = P'.
10.     if the fitness value of the best composite operator in P is above fitness
          threshold value, then stop.
11.     for each composite operator in P, do loop 2
12.       if its size exceeds the size limit and it is not the best composite
            operator in P, then replace it with one of its subtrees whose size is
            within the size limit.
        endfor // loop 2
        endfor // loop 1
```

Figure 2.1. Steady-state genetic programming algorithm.

Generational Genetic Programming Algorithm:

```
1.  randomly generate population P of size M and evaluate each composite
      operator in P.
2.  for gen = 1 to N do loop 1  // N is the number of generation
3.      keep the best composite operator in P.
4.      perform crossoveron the composite operators in P until crossover rate
          is satisfied and keep all the offspring from crossover separately.
5.      perform mutation on the composite operators in P and the offspring
          from crossover with the probability of mutation rate.
6.      perform selection on P to select some composite operators. The number
          of selected composite operators must be M minus the number of
          composite operators from crossover.
7.      combine the composite operators from crossover with those selected
          from P to get a new population P' of the same size as P.
8.      evaluate offspring from crossover and the mutated composite operators.
9.      let the best composite operator from P replace the worst composite
          operator in P' and let P = P'.
10.     if the fitness of the best composite operator in P is above fitness
          threshold, then stop.
11.     for each composite operator in P, do loop 2.
12.         if its size exceeds the size limit and it is not the best composite
              operator in P, then replace it with one of its subtrees whose size is
              within the size limit.
        endfor // loop 2
    endfor // loop 1
```

Figure 2.2. Generational genetic programming algorithm.

2.4 Experiments

Various experiments are performed to test the efficacy of genetic programming in extracting regions of interest from real synthetic aperture radar (SAR) images, infrared (IR) images and RGB color images. We provide detailed results using examples from remote sensing, target recognition, and survallence/monitoring application areas. We give several comparisons to demonstrate the effectiveness of the approach. These include comparisons with the image-based genetic programming and the traditional ROI extraction algorithm. We also provide the performance of the GP with hard limit on the composite operator size. The results from the hard size limit GP are compared with those from the MDL-based GP in chapter 3. We provide examples of both two-class classification and multi-class classification.

The size of SAR images is 128×128, except the tank SAR images whose size is 80×80, and the size of IR and RGB color images is 160×120. GP in chapter 2.4.1 Examples 1-5, 2.4.2, 2.4.5 and 2.4.6 is not applied to a whole training image, but only to a region or regions carefully selected from a training image, to generate the composite operators. The generated composite operator (with the highest fitness) is then applied to the whole training image and to some other testing images to evaluate it. The advantage of performing training on a small selected region is that it can greatly reduce the training time, making it practical for the GP system to be used as a subsystem of other learning systems, which improve the efficiency of GP by adapting the parameters of GP system based on its performance. Our experiments show that if the training regions are carefully selected from the training images, the best composite operator generated by GP is effective. In the following experiments in sections 2.4.1, 2.4.2, 2.4.3, and 2.4.6, the parameters are: population size (100), the number of generations (70), the fitness threshold value (1.0), the crossover rate (0.6), the mutation rate (0.05), the soft size limit of composite operators (30), and the segmentation threshold (0). In each experiment, GP is invoked ten times with the same parameters and the same training region(s). The coordinate of the upper left corner of an image is (0, 0). The ground-truth is used only during the training, it is not needed during testing. We use it in testing only for evaluating the performance of the composite operator on testing images. The size, orientation or shape of the objects in testing images is different from those in the training images.

2.4.1 SAR Images

Five experiments are performed with real SAR images. The experimental results from one run and the average performance of ten runs are given in Table 2.3. We select the run in which GP finds the best composite operator among the composite operators found in all ten runs. The first two rows show the average values of the above fitness values over all ten runs. The third and fourth rows show the fitness value of the best composite operator and the population fitness value (average fitness value of all the composite operators in the population) on *training region (s)* in the initial and final generations in the selected run. The fitness values of the best composite operators on the entire training image (numbers with a * superscript) and other testing images in their entirety are also given. The regions extracted during the training and testing by the best composite operator from the selected run are shown in the following examples.

Example 1 — Road extraction: Three images contain road, the first one contains horizontal paved road and field (Figure 2.3(a)); the second one contains unpaved road and field (Figure 2.10 (a)); the third one contains vertical paved road and grass (Figure 2.10(d)). Training is done on the training regions of training image shown in Figure 2.3(a). After the training, the learned composite operator is evaluated on the whole training image and testing images. There are two training regions, locating from (5, 19) to (50, 119) and from (82, 48) to (126, 124), respectively. Figure 2.3(b) shows the ground-truth provided by the user and the training regions. The white region corresponds to the road and only the training regions of the ground-truth are used in the evaluation during the training. Figure 2.4 shows the sixteen primitive feature images of the training image.

Table 2.3. The performance on various examples of SAR images

Training Performance										
	Road		Lake		River		Field		Tank	
	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p
Ave. $f_{initial}$	0.55	0.27	0.59	0.32	0.48	0.18	0.54	0.37	0.61	0.17
Ave. f_{final}	0.83	0.60	0.95	0.92	0.85	0.77	0.76	0.59	0.86	0.68
$f_{initial}$	0.68	0.28	0.56	0.32	0.65	0.18	0.53	0.39	0.51	0.16
f_{final}	0.95 0.93*	0.67	0.97 0.93*	0.93	0.90 0.71*	0.85	0.78 0.89*	0.64	0.88 0.88*	0.80
Testing Performance										
	Road		Lake		River		Field		Tank	
f_{test}	0.90, 0.93		0.98		0.83		0.80		0.84	

f_{op}: fitness of the best composite operator on selected region(s),
f_p : fitness of population on selected region(s),
*: indicate fitness on the *entire* training images,
$f_{initial}$; fitness of the initial generation on selected region(s),
f_{final}: fitness of the final population on selected region(s),
f_{test}: fitness of the best composite operator on the *entire* testing images.

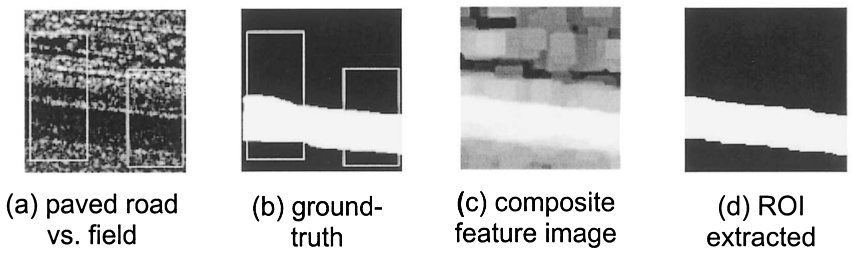


Figure 2.3. Training SAR image containing road.

The generational GP is used to synthesize a composite operator to extract the road and the results of the best of the ten runs (sixth run) are reported. The fitness value of the best composite operator in the initial population is 0.68 and the population fitness value is 0.28. The fitness value of the best composite operator in the final population is 0.95 and the population fitness value is 0.67. Figure 2.3(c) shows the output image of the best composite operator on the whole training image and Figure 2.3(d) shows the binary image after segmentation. The output image has both positive pixels in brighter shade and negative pixels in darker shade. Positive pixels belong to the region to be extracted. The fitness value of the extracted ROI is 0.93. The best composite operator has 17 nodes and its depth is 16. It has only one leaf node containing 5×5 median image. The median image is less noisy, since median filtering is effective in eliminating speckle noises. The best composite operator is shown in Figure 2.5, where PFIM14 is 5×5 median image. Figure 2.6 shows how the average fitness of the best composite operator and average fitness of population over all 10 runs change as GP explores the composite operator space. Unlike [17] where the population fitness approaches the fitness of the best composite operator as GP proceeds, in Figure 2.6, population fitness is much lower than that of best composite operator even at the end of GP search. It is reasonable, since we don't restrict the selection of crossover points. The population fitness is not important since only the best composite operator is used in testing. If GP finds one effective composite operator, the GP learning is successful. The large difference between the fitness of the best composite operator and the population indicates that the diversity of the population is always maintained during the GP search, which is very helpful in preventing premature convergence.

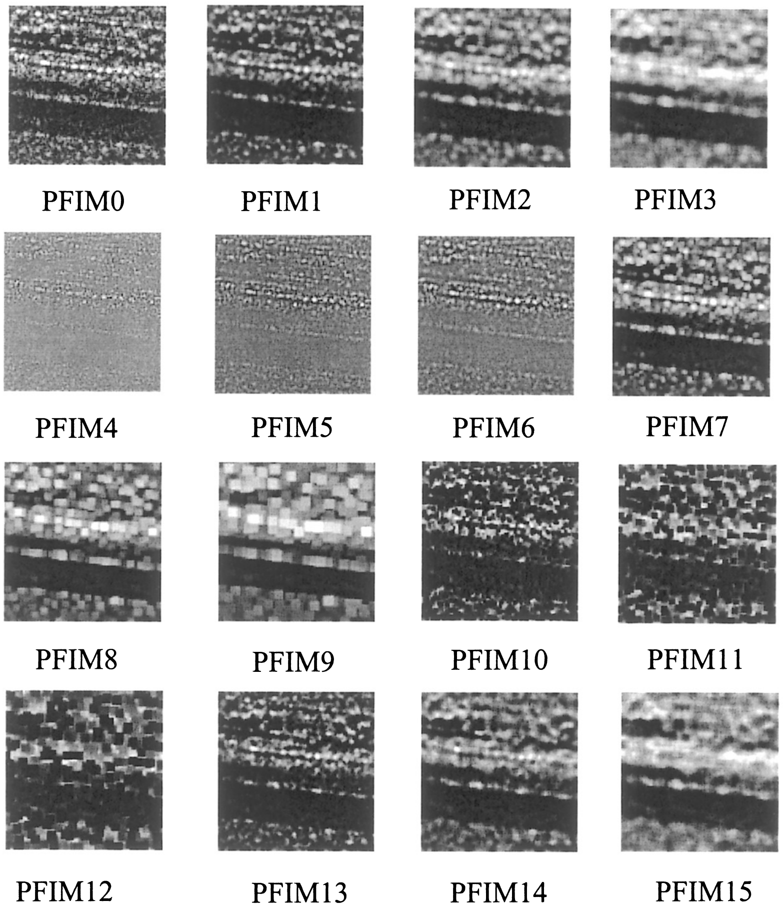


Figure 2.4. Sixteen primitive feature images of training SAR image containing road.

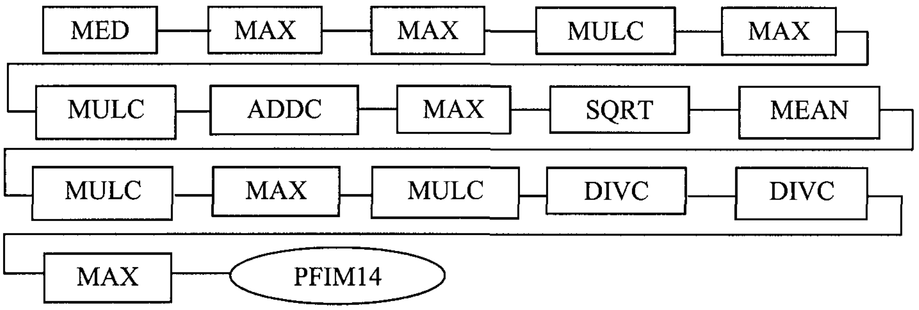


Figure 2.5. Learned composite operator tree.

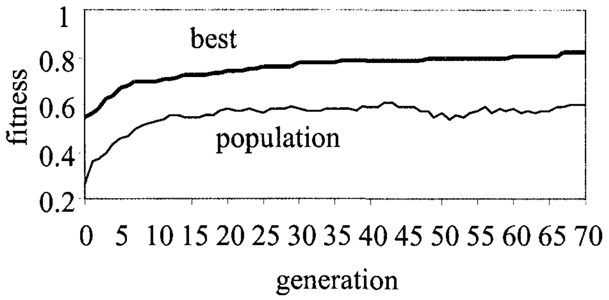
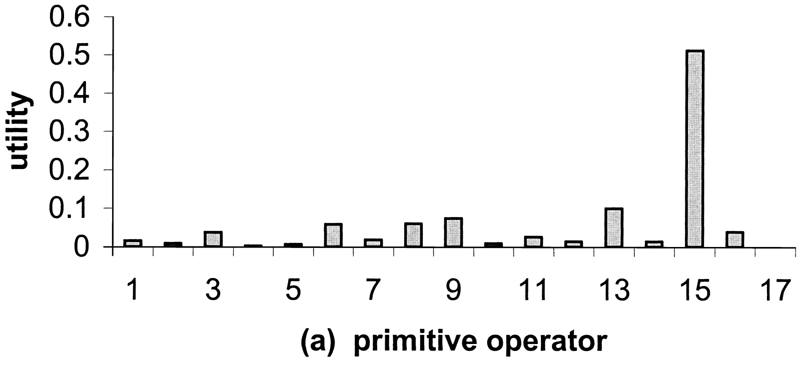


Figure 2.6. Fitness versus generation (road vs. field).

Ten best composite operators are learned in ten runs. After computing the percentage of each primitive operator and primitive feature image among the total number of internal nodes (representing primitive operators) and the total number of leaf nodes (representing primitive feature images) of these ten best compsite operators, we get the utility (frequency of occurence) of primitive operators and primitive feature images, which is shown in Figure 2.7(a) and (b). MED (primitive operator 15) and PFIM5 (5×5 deviation image) have the highest frequency of utility. Figure 2.8 shows the output image of each node of the best composite operator shown in Figure 2.5. From left to right and top to bottom, the images correspond to nodes sorted in the pre-order traversal of the binary tree representing the best composite operator. The output of the root node is shown in Figure 2.3(c), and Figure 2.8 shows the outputs of other nodes. The primitive operators in Figure 2.8 are connected by arrow. The operator at the tail of an arrow provides input to the operator at the head of the arrow. After segmenting the output image of a node, we get the ROI (shown as the white region) extracted by the corresponding subtree rooted at the node. The extracted ROIs and their fitness values are shown in Figure 2.9. If an output image of a node has no positive pixel (for example, the output of MEAN primitive operator), nothing is extracted and the fitness value is 0; if an output image has positive pixels only (for example, PFIM14 has positive pixels only), everything is extracted and the fitness is 0.25. The output of the root node storing primitive operator MED is shown in Figure 2.3(d).



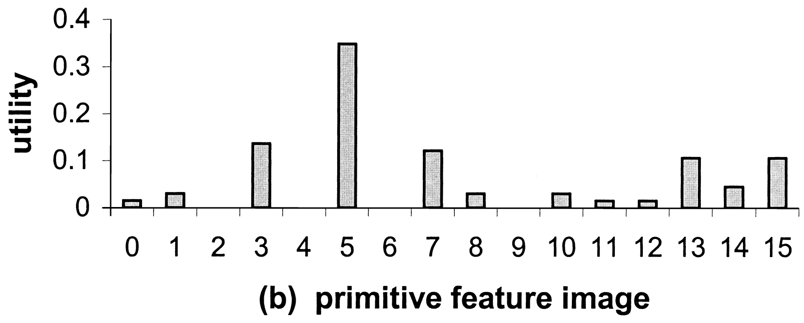


Figure 2.7. Utility of primitive operators and primitive feature images.

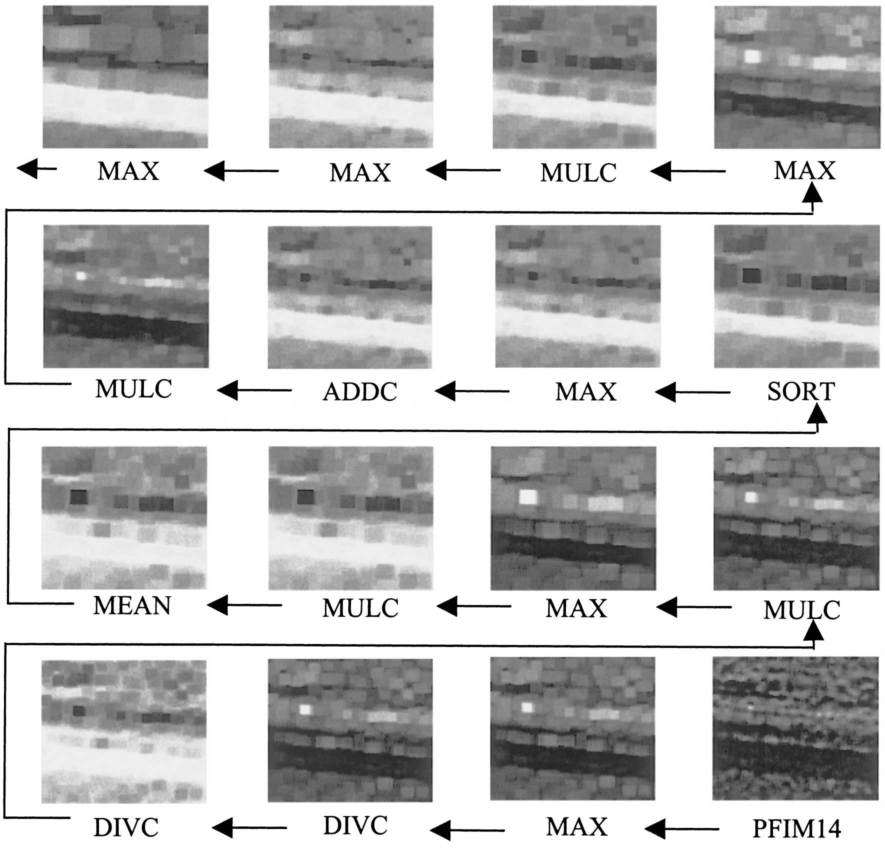


Figure 2.8. Feature images output by the nodes of the best composite operator. The ouput of the root node is shown Figure 2.3(c).

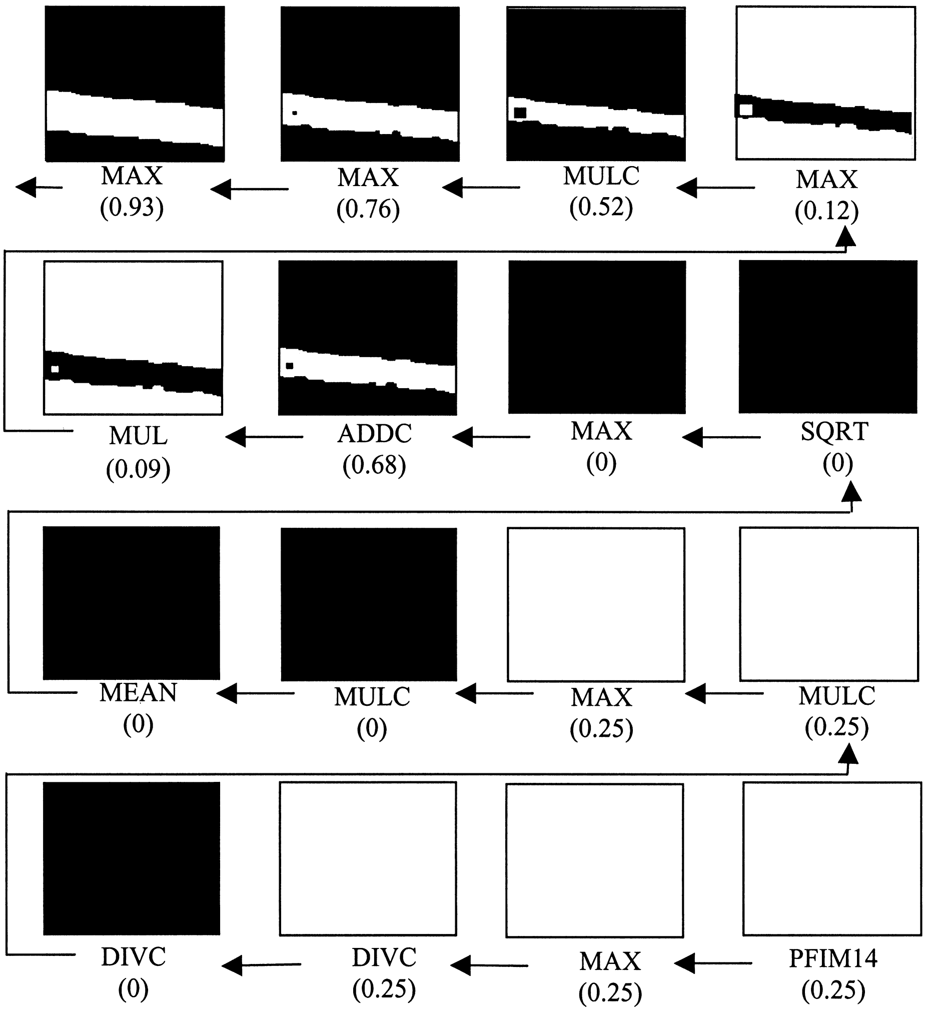


Figure 2.9. ROIs extracted from the output images of the nodes of the best composite operator. The fitness value is shown for the entire image. The ouput of the root node is shown Figure 2.3(d).

We applied the composite operator obtained in the above training to the other two real SAR images shown in Figure 2.10 (a) and Figure 2.10 (d). Figure 2.10 (b) and Figure 2.10 (e) show the output of the composite operator and Figure 2.10 (c) shows the region extracted from Figure 2.10 (a). The fitness value of the region is 0.90. Figure 2.10 (f) shows the region extracted from Figure 2.10(d). The fitness value of the region is 0.93.

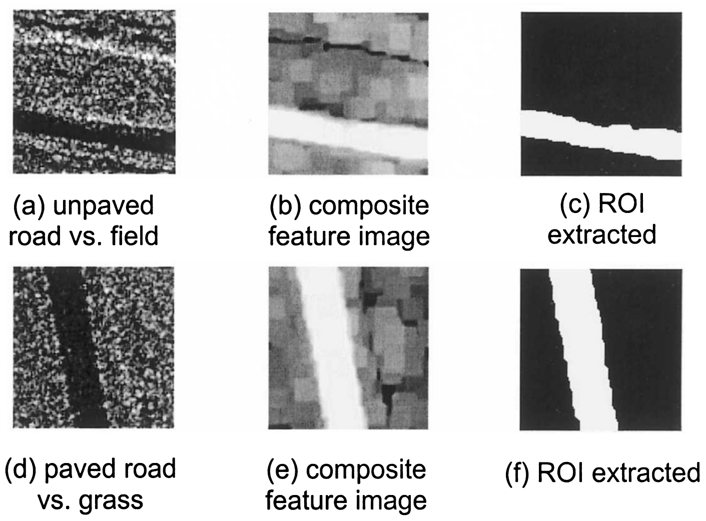

(a) unpaved road vs. field
(b) composite feature image
(c) ROI extracted
(d) paved road vs. grass
(e) composite feature image
(f) ROI extracted

Figure 2.10. Testing SAR images containing road.

Example 2 — Lake Extraction: Two SAR images contain lake (Figure 2.11(a), Figure 2.12(a)), the first one contains a lake and field, and the second one contains a lake and grass. Figure 2.11(a) shows the original training image containing lake and field and the training region from (85, 85) to (127, 127). Figure 2.11(b) shows the ground-truth provided by the user. The white region corresponds to the lake to be extracted. Figure 2.12 (a) shows the image containing lake and grass used only in testing.

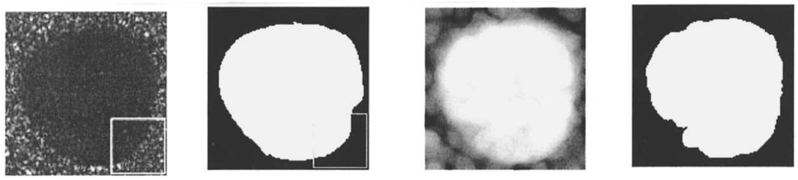


Figure 2.11. Training SAR image containing lake.

The steady-state GP is used to generate the composite operator and the results of the best of ten runs (ninth run) are shown. The fitness value of the best composite operator in the initial population is 0.56 and the population fitness value is 0.32. The fitness value of the best composite operator in the final population is 0.97 and the population fitness value is 0.93. Figure 2.11(c) shows the output image of the best composite operator on the whole training image and Figure 2.11(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.93.

We apply the composite operator to the testing image containing lake and grass. Figure 2.12(b) shows the output of the composite operator and Figure 2.12(c) shows the region extracted from Figure 2.12(a). The fitness of the region is 0.98.

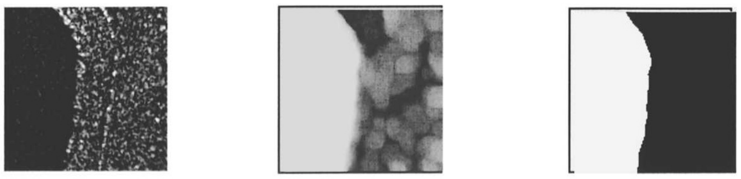


Figure 2.12. Testing SAR image containing lake.

Example 3 — River Extraction: Two SAR images contain river and field. Figure 2.13(a) and Figure 2.13(b) show the original training image and the ground-truth provided by the user. The white region in Figure 2.13(b) corresponds to the river to be extracted. The training regions are from (68, 31) to (126, 103) and from (2, 8) to (28, 74). The testing SAR image is shown in Figure 2.16(a).

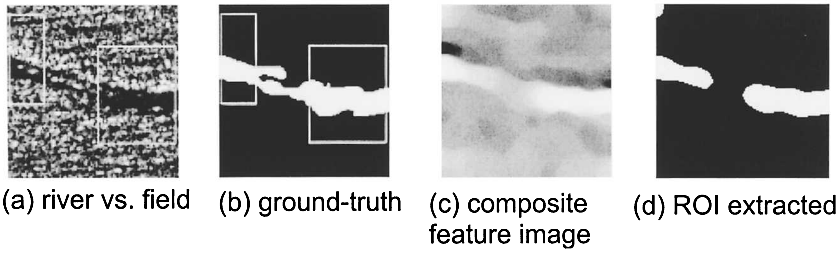


Figure 2.13. Training SAR image containing river.

The steady-state GP was used to generate the composite operator and the results from the best of ten runs (fourth run) are reported. The fitness value of the best composite operator in the initial population is 0.65 and the population fitness value is 0.18. The fitness value of the best composite operator in the final population is 0.90 and the population fitness value is 0.85. Figure 2.13(c) shows the output image of the best composite operator on the whole training image and Figure 2.13(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.71. The best composite operator has 29 nodes and a depth of 19. It has five leaf nodes that all contain 7×7 median image shown in Figure 2.14. There are 17 MED operators that are very useful in eliminating speckle noise. Figure 2.15 shows how the average fitness of the best composite operator and average fitness of population over all 10 runs change as GP explores the composite operator space.

```
(MED (MED (MED (ADD (MED (STDV (MED
PFIM15))) (MED (MED (MED (MED (MED
(MIN2 (MED PFIM15) (MED (MED (MED
(MIN2 (MED PFIM15) (MED (MED (MIN2
PFIM15 (SUBC (DIVC PFIM15))))))))))))))))))
```

Figure 2.14. Learned composite operator tree.

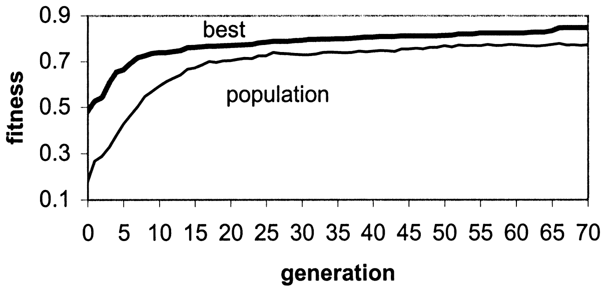


Figure 2.15. Fitness versus generation (river vs. field).

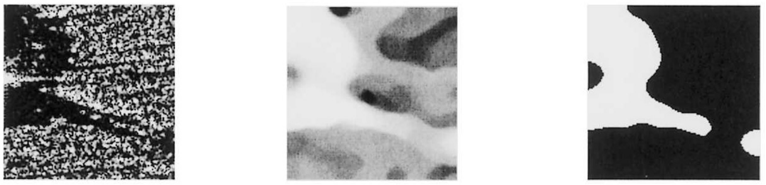
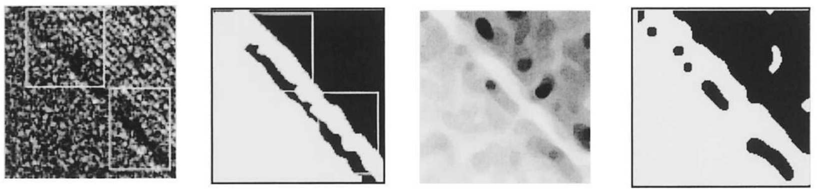


Figure 2.16. Testing SAR image containing river.

We apply the composite operator to the testing image containing a river and field. Figure 2.16(b) shows the output of the composite operator and Figure 2.16(c) shows the region extracted from Figure 2.16(a) and the fitness value of the region is 0.83. There are some islands in the river and these islands along with part of the river around them are not extracted.

Example 4 — Field Extraction: Two SAR images contain field and grass. Figure 2.17(a) and (b) show the original training image and the ground-truth. The training regions are from (17, 3) to (75, 61) and from (79, 62) to (124, 122). Extracting field from a SAR image containing field and grass is the most difficult task among the five experiments, since the grass and field are similar to each other and some small regions between grassy areas are actually field pixels.



(a) field vs. grass (b) ground-truth (c) composite feature image (d) ROI extracted

Figure 2.17. Training SAR image containing field.

The generational GP was used to generate the composite operator and the results from the best of ten runs (second run) are reported. The fitness value of the best composite operator in the initial population is 0.53 and the population fitness value is 0.39. The fitness value of the best composite operator in the final population is 0.78 and the population fitness value is 0.64. Figure 2.17(c) shows the output image of the best composite operator on the whole training image and Figure 2.17(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.89.

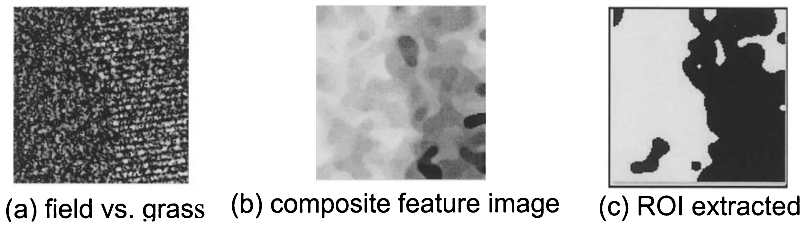


Figure 2.18. Testing SAR image containing field.

We apply the composite operator to the testing image containing field and grass shown in Figure 2.18(a). Figure 2.18(b) shows the output of the composite operator and Figure 2.18(c) shows the region extracted from Figure 2.18(a). The fitness value of the region is 0.80.

Example 5 — Tank Extraction: We use 80×80 size SAR images of a T72 tank that are taken under different depression and azimuth angles. The training image contains a T72 tank at a 17° depression angle and 135° azimuth angle, which is shown in Figure 2.19(a). The training region is from (19, 17) to (68, 66). The testing SAR image contains a T72 tank at a 20° depression angle and 225° azimuth angle, which is shown in Figure 2.22(a). The ground-truth is shown in Figure 2.19(b).

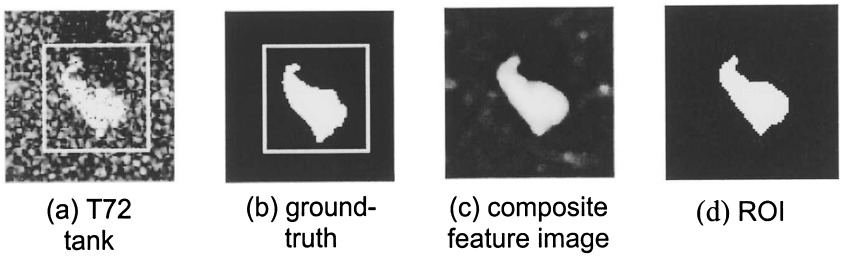


Figure 2.19. Training SAR image containing tank.

The generational GP is applied to synthesize composite operators for tank detection and the results from the best of ten runs (first run) are reported. The fitness value of the best composite operator in the initial population is 0.51 and the population fitness value is 0.16. The fitness value of the best composite operator in the final population is 0.88 and the population fitness value is 0.80. Figure 2.19(c) shows the output image of the best composite operator on the whole training image and Figure 2.19 (d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.88. The best composite operator, shown in Figure 2.20, has 10 nodes and its depth is 9. It has only one leaf node, which contains the 5×5 mean image. Figure 2.21 shows how the average fitness of the best composite operator and average fitness of population over all 10 runs change as GP proceeds.

```
(MED (SQRT (MULC (MULC (SUBC (MULC
(SQRT (SUBC (SQRT PFIM2)))))))))
```

Figure 2.20. Learned composite operator tree in LISP notation.

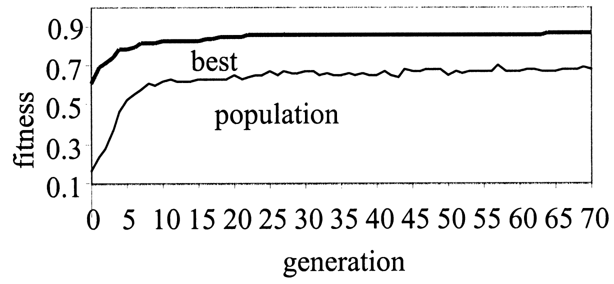
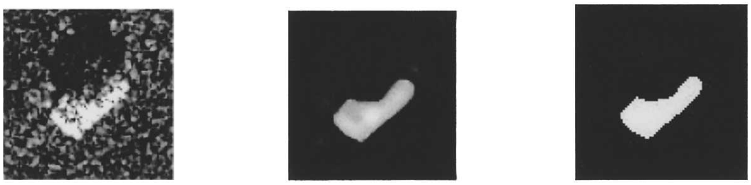


Figure 2.21. Fitness versus generation (T72 tank).

We apply the composite operator to the testing image containing T72 tank under depression angle 20° and azimuth angle 225°. Figure 2.22(b) shows the output of the composite operator and Figure 2.22(c) shows the region corresponding to the tank. The fitness of the extracted ROI is 0.84.



(a) T72 tank (b) composite feature image (c) ROI extracted

Figure 2.22. Testing SAR image containing tank.

Our results show that GP is very much capable of synthesizing composite operators for target detection. With more and more SAR images collected by satellites and airplanes, it is impractical for human experts to scan each SAR image to find targets. Applying the synthesized composite operators on these images, regions containing potential targets can be quickly detected and passed on to automatic target recognition systems or to human experts for further examination. Concentrating on the regions of interest, the human experts and recognition systems can perform recognition task more effectively and more efficiently.

Note that composite operators shown in Figure 2.5 and Figure 2.20 may be called as "processing chains," which is a simpler binary tree in which each internal node has only one child. Most of the composite operators learned by GP in our experiments are not processing chains.

2.4.2 Infrared and color images

One experiment is performed with infrared (IR) images and two are performed with RGB color images. The experimental results from one run and the average performance of ten runs are shown in Table 2.4. As we did in chapter 2.4.1, we select the run in which GP finds the best composite operator among the composite operators found in all the ten runs. The regions extracted during the training and testing by the best composite operator from the selected run are shown in the following examples.

Table 2.4. The performance results on IR and RGB color images.

Training performance						
	IR image - people		RGB image - car		RGB image - SUV	
	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p
Ave. $f_{initial}$	0.59	0.21	0.47	0.18	0.34	0.21
Ave. f_{final}	0.85	0.65	0.72	0.67	0.61	0.56
$f_{initial}$	0.56	0.23	0.35	0.18	0.33	0.22
f_{final}	0.93 0.85*	0.79	0.84 0.82*	0.79	0.69 0.69*	0.65
Testing performance						
	IR image - people		RGB image - car		RGB image - SUV	
f_{test}	0.84, 0.81, 0.86		0.76		0.58	

f_{op}: fitness of the best composite operator on selected region(s),
f_p : fitness of population on selected region(s),
*: indicate finess on the *entire* training images,
$f_{initial}$: fitness of the initial generation on selected region(s),
f_{final}: fitness of the final population on selected region(s),
f_{test}: fitness of the best composite operator on the *entire* testing images.

People extraction in IR images: In IR images, pixel values correspond to the temperature in the scene. We have four IR images with one used in training and the other three used in testing. Figure 2.23(a) and (b) show the training image and the ground-truth. Two training regions are from (59, 9) to (106, 88) and from (2, 3) to (21, 82), respectively. The left training region contains no pixel belonging to the person. The reason for selecting it during the training is that there are major pixel intensity changes among the pixels in this region. Nothing in this region should be detected. The fitness of composite operator on this region is defined as one minus the percentage of pixels detected in the region. If nothing is detected, the fitness value is 1.0. Averaging the fitness values of the two training regions, we get the fitness during the training. When the learned composite operator is applied to the whole training image, the fitness is computed as a measurement of the overlap between the ground-truth and the extracted ROI, as we did in the previous experiments. Three testing IR images are shown in Figure 2.26(a), (d) and (g).

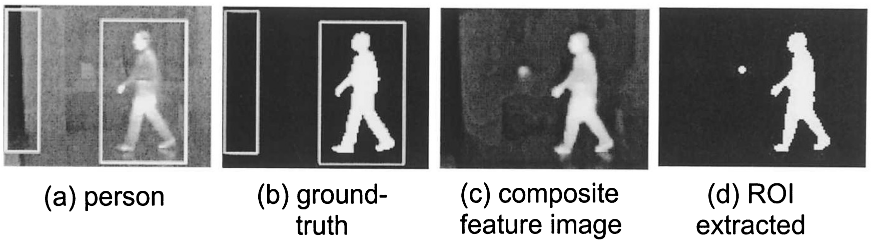


Figure 2.23. Training IR image containing a person.

The generational GP is applied to synthesize composite operators for person detection and the results from the best of ten runs (third run) are reported. The fitness value of the best composite operator in the initial population is 0.56 and the population fitness value is 0.23. The fitness value of the best composite operator in the final population is 0.93 and the population fitness value is 0.79. Figure 2.23(c) shows the output image of the best composite operator on the whole training image and Figure 2.23(d) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.85. The best composite operator (shown in Figure 2.24) has 28 nodes and a depth of 13 with 9 leaf nodes. Figure 2.25 shows how the average fitness of the best composite operator and average fitness of population over all the 10 runs change as GP proceeds.

```
(SQRT (SQRT (SUBC (SQRT (MAX2 (MAX2 PFIM1 (SUB (MAX2
PFIM14 PFIM15) (DIV (MULC (SQRT (MAX (MAX (ADD
PFIM12 PFIM15))))) PFIM9))) (DIV (MULC (SQRT (MAX (ADD
PFIM12 PFIM9)))) PFIM9))))))
```

Figure 2.24. Learned composite operator tree in LISP notation.

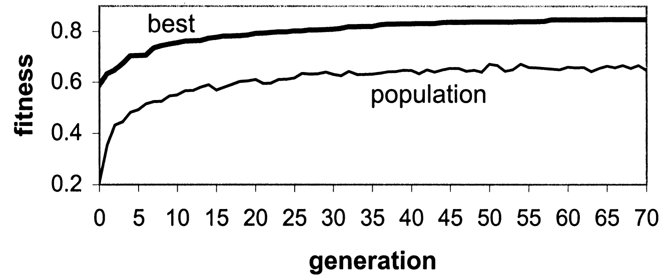
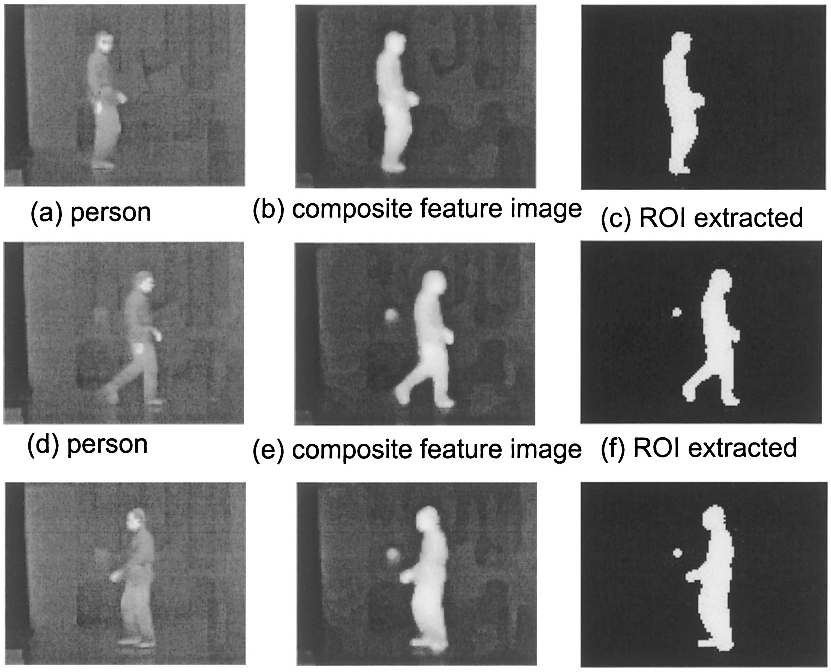


Figure 2.25. Fitness versus generation (person).

We apply the composite operator to the testing images shown in Figure 2.26. Figure 2.26(b), (e) and (h) show the output of the composite operator and Figure 2.26(c), (f) and (i) show the ROI extracted. Their fitness values are 0.84, 0.81 and 0.86 respectively.

Car extraction in RGB color images: GP is applied to learn features to detect a car in RGB color images. Unlike previous experiments, the primitive feature images in this experiment are RED, GREEN and BLUE planes of a RGB color image. Figure 2.27(a), (b) and (c) show the RED, GREEN and BLUE planes of the training image. The ground-truth is shown in Figure 2.27(d). The training region is from (21, 3) to (91, 46).

The steady-state GP is applied to synthesize composite operators for car detection and the results from best of ten runs (fourth run) are reported. The fitness value of the best composite operator in the initial population is 0.35 and the population fitness value is 0.18. The fitness value of the best composite operator in the final population is 0.84 and the population fitness value is 0.79. Figure 2.27(e) shows the output image of the best composite operator on the whole training image and Figure 2.27(f) shows the binary image after segmentation. The fitness value of the extracted ROI is 0.82. The best composite operator has 44 nodes and its depth is 21. It has ten leaf nodes with one containing GREEN plane and the others containing BLUE plane. It is shown in Figure 2.28, where PFG means GREEN plane and PFB means BLUE plane. Note that only green and blue planes are used by the composite operator. Figure 2.29 shows how the average fitness of the best composite operator and average fitness of population over all 10 runs change as GP runs.



(a) person (b) composite feature image (c) ROI extracted

(d) person (e) composite feature image (f) ROI extracted

(g) person (h) composite feature image (i) ROI extracted

Figure 2.26. Testing IR images containing a person.

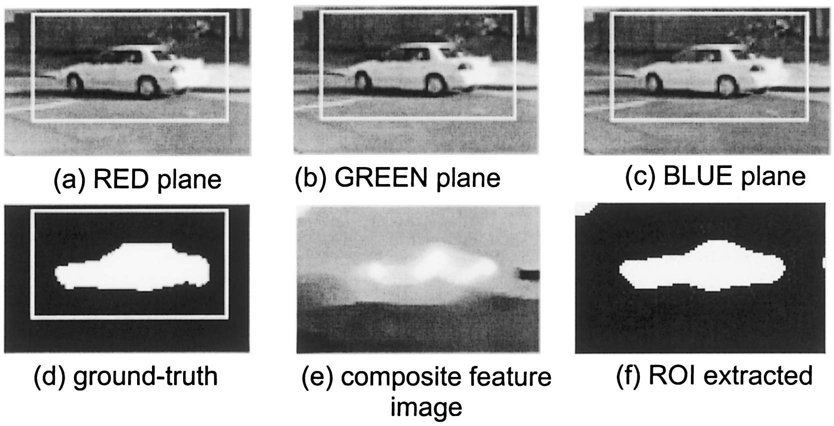


Figure 2.27. Training RGB color image containing car.

```
(MED (MED (MED (MULC (MUL (SUB (MIN
(MEAN (MAX2 (MED (ADDC (MAX2 (ADDC
(ADDC (MED (MAX2 (MED (MED (MAX2 (MED
(ADDC PFB)) PFB))) PFB)))) PFB))) (MED PFG))))
(ADDC (MAX2 (ADDC (ADDC (MED (MAX2 (MED
(MED (MAX2 (MED (ADDC PFB)) PFB))) PFB))))
PFB))) (ADDC PFB))))))
```

Figure 2.28. Learned composite operator tree in LISP notation.

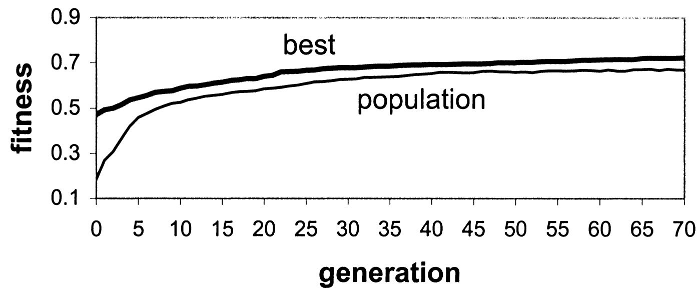


Figure 2.29. Fitness versus generation (car).

We apply the composite operator to the testing image whose RED plane is shown in Figure 2.30(a). Figure 2.30 (b) shows the output of the composite operator and Figure 2.30(c) shows the ROI extracted. The fitness value of extracted ROI is 0.76.

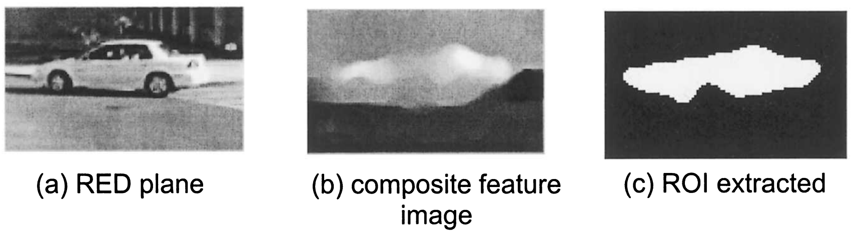


Figure 2.30. Testing RGB color image containing car.

SUV extraction in RGB color images: In this subsection, GP is applied to learn features to detect SUV (sports utility vehicle) in RGB color images. The images containing a SUV have more complicated background than the images containing the car, increasing the difficulty in SUV detection. This will be a difficult example for any segmentation technique in computer vision and pattern recognition. Figure 2.31(a), (b) and (c) show the RED, GREEN and BLUE planes of the training image and Figure 2.31(d) shows the ground-truth. The training region is from (20, 21) to (139, 100). Figure 2.31(f) and (g) show the RED plane and the ground-truth of the testing image.

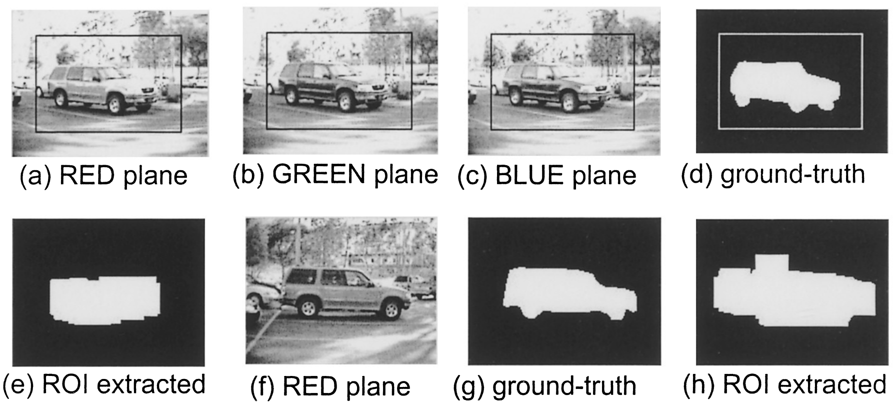


Figure 2.31. Training and testing RGB color image containing SUV.

The steady-state GP is applied to synthesize composite operators for SUV detection and the results from the best of ten runs (fourth run) are reported. The fitness value of the best composite operator in the initial population is 0.33 and the population fitness value is 0.22. The fitness value of the best composite operator in the final population is 0.69 and the population fitness value is 0.65. Figure 2.31(e) and (h) show the ROI extracted by the best composite operator from training and testing images. The fitness values of the extracted ROIs are 0.69 and 0.58, respectively. The extracted ROIs are not very satisfactory, since the shapes of ROIs differ from the shapes of vehicles in images. However, the extracted ROIs contain SUVs in the training and testing images, which means the locations of the vehicle are correctly detected.

2.4.3 Comparison with GP with hard limit on composite operator size

As stated in chapter 2.3, GP has a well-known code bloat problem in that the size of individuals becomes larger and larger as GP proceeds if no measure is taken to control the size. Large individuals cause problems such as reducing the speed of GP, taking up a lot of computer memory, and overfitting the training data. To resolve this problem, a simple way is to set a limit on the size of individuals. If crossover or mutation produces an individual above the size limit, the individual is discarded and crossover or mutation is performed again.

In this section, the performance of a *hard-size GP* (GP with hard limit on composite operator size = 30) is compared with the *soft-size GP* (GP with soft limit on composite operator size) whose performance is reported before. The major difference between the soft-size GP and the hard-size GP, as stated in chapter 2.3, is that in the soft-size GP, a composite operator with size above the size limit is kept in the population if its fitness value is the highest (used in this chapter) or above a certain threshold value. All the other parameters of these two GPs are the same.

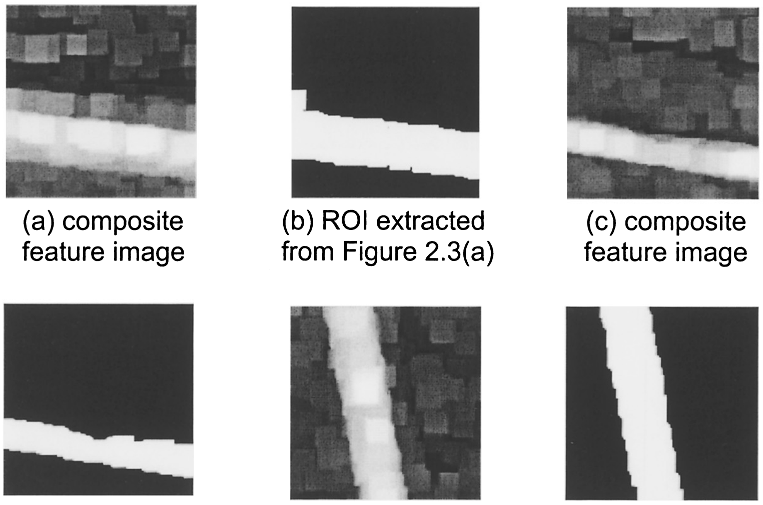
Table 2.5 shows the performance of the hard-size GP. In the following, the results from the best composite operator found in ten runs are shown. The average performance of hard-size GP over ten runs is compared with that of a MDL-based GP with smart operators in chapter 3.

Table 2.5. The performance results on various examples of SAR images. The hard limit on composite operator size is used.

Training Performance										
	Road		Lake		River		Field		Tank	
	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p
Ave. $f_{initial}$	0.47	0.26	0.64	0.32	0.49	0.18	0.53	0.38	0.49	0.16
Ave. f_{final}	0.82	0.81	0.93	0.92	0.82	0.77	0.73	0.72	0.85	0.83
$f_{initial}$	0.60	0.27	0.62	0.30	0.59	0.19	0.52	0.38	0.65	0.17
f_{final}	0.94 0.90*	0.93	0.99 0.95*	0.95	0.89 0.72*	0.86	0.78 0.88*	0.77	0.88 0.88*	0.87
Testing Performance										
	Road		Lake		River		Field		Tank	
f_{test}	0.90, 0.93		0.97		0.83		0.81		0.84	

f_{op}: fitness of the best composite operator on selected region(s),
f_p : fitness of population on selected region(s),
*: indicate finess on the *entire* training images,
$f_{initial}$: fitness of the initial generation on selected region(s),
f_{final}: fitness of the final population on selected region(s),
f_{test}: fitness of the best composite operator on the *entire* testing images.

• **Road extraction:** Figure 2.3(a) shows the training image and Figure 2.10(a), (d) show the testing images. The generational GP is used to generate a composite operator to extract the road and the best composite operator is found in the seventh run. The fitness value of the best composite operator in the initial population is 0.60 and the population fitness value is 0.27. The fitness value of the best composite operator in the final population is 0.94 and the population fitness value is 0.93. The fitness of the extracted ROI is 0.90. Figure 2.32(a) shows the output image of the best composite operator in the final population and Figure 2.32(b) shows the extracted ROI. We apply the composite operator obtained in the above training to the two testing SAR images. Figure 2.32(c) and (d) show the output image of the composite operator and the ROI extracted from Figure 2.10(a), respectively. The fitness value of the extracted ROI is 0.90. Figure 2.32 (e) and (f) show the output image of the composite operator and the ROI extracted from Figure 2.10(d), respectively. The fitness value of the extracted ROI is 0.93.



(a) composite feature image

(b) ROI extracted from Figure 2.3(a)

(c) composite feature image

(d) ROI extracted from Figure 2.10(a)

(e) composite feature image

(f) ROI extracted from Figure 2.10(d)

Figure 2.32. Results on SAR images containing road.

The best composite operator has 27 nodes and its depth is 16. It has five leaf nodes, three contain 5×5 median image and the other two contain 7×7 median image. It is shown in Figure 2.33, where PFIM14 and PFIM15 are 5×5 and 7×7 median images, respectively. The median images have less speckle noise, since median filtering is effective in eliminating speckle noise. Figure 2.34 shows the change in the average fitness of the best composite operators and the average fitness of the populations over all the 10 runs as GP explores the composite operator space. GP gradually shifts the population to the regions of space containing good composite operators.

```
(MAX (MAX (MIN (DIVC (DIV (ADDC
(ADD (ADDC (ADD (SUBC (ADDC (ADD
(SUBC (STDV (MAX (SUBC PFIM15))))
(MAX (SUBC PFIM14))))) (MAX (SUBC
PFIM14)))) (MAX (SUBC PFIM14))))
PFIM15)))))
```

Figure 2.33. Learned composite operator tree in LISP notation.

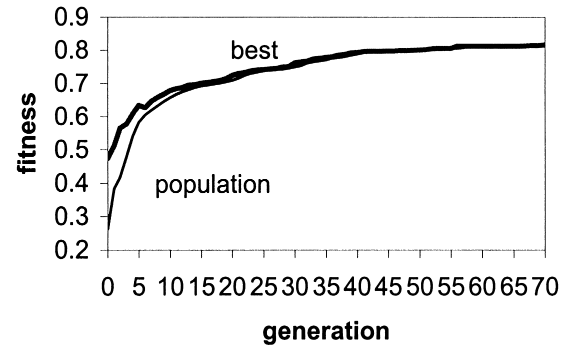


Figure 2.34. Fitness versus generation (road vs. field).

• **Lake extraction:** Figure 2.11(a) shows the training image and Figure 2.12(a) shows the testing image. The steady-state GP is used to generate the composite operator and the best composite operator is found in the 4th run. The fitness value of the best composite operator in the initial population is 0.62 and the population fitness value is 0.30. The fitness value of the best composite operator in the final population is 0.99 and the population fitness value is 0.95. The fitness of the extracted ROI is 0.95. Figure 2.35(a) shows the output image of the best composite operator in the final population and Figure 2.35(b) shows the extracted ROI. We apply the composite operator to the testing SAR image. Figure 2.35(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.97, respectively. In Figure 2.35(a) and (c), pixels in the small dark regions have very low pixel values (negative values with very large absolute value), thus making many pixels appear bright, although some of them have negative pixel values.

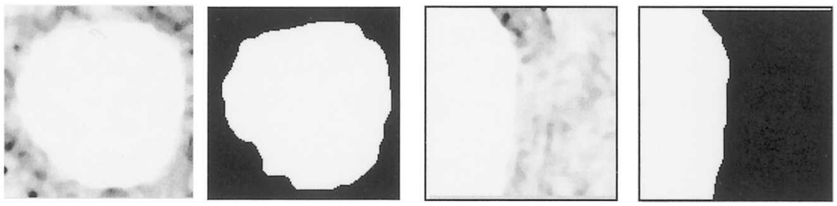

(a) composite feature image
(b) ROI extracted from Figure 2.11(a)
(c) composite feature image
(d) ROI extracted from Figure 2.12(a)

Figure 2.35. Results on SAR images containing lake.

- **River extraction:** Figure 2.13(a) shows the training image and Figure 2.16(a) shows the testing image. The steady-state GP is used to generate the composite operator and the results from the first run are reported. The fitness value of the best composite operator in the initial population is 0.59 and the population fitness value is 0.19. The fitness value of the best composite operator in the final population is 0.89 and the population fitness value is 0.86. The fitness of the extracted ROI is 0.72. Figure 2.36(a) shows the output image of the best composite operator in the final population and Figure 2.36(b) shows the extracted ROI. We apply the composite operator to the testing image. Figure 2.36(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.83.

The best composite operator has 30 nodes and its depth is 23. It has four leaf nodes, three contain 5×5 mean image and the other one contains 3×3 mean image. There are more than ten MED operators that are very useful in eliminating speckle noise. It is shown in Figure 2.37. Figure 2.38 shows how the average fitness of the best composite operators and the average fitness of the populations over all the 10 runs change as GP explores the composite operator space.

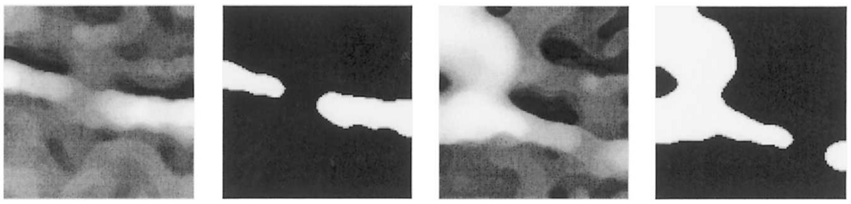

(a) composite feature image
(b) ROI extracted from Figure 2.13(a)
(c) composite feature image
(d) ROI extracted from Figure 2.16(a)

Figure 2.36. Results on SAR images containing river.

```
(MULC (MED (MED (MED (MED (MED
(MED (MED (MED (MED (MED (MIN
(ADDC (LOG (ADD (MAX (MIN (MULC
PFIM2))) (DIV (MIN (MULC (MED (MIN
(MAX (SUB PFIM2 (MULC PFIM2))))))
PFIM1))))))))))))))))
```

Figure 2.37. Learned composite operator tree in LISP notation.

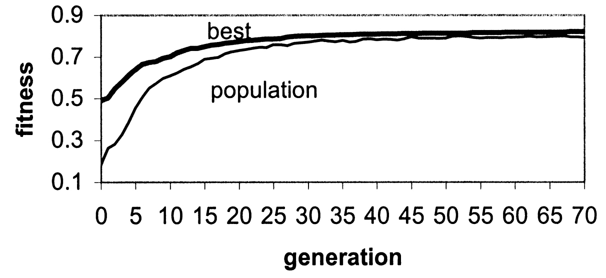


Figure 2.38. Fitness versus generation (river vs. field).

• **Field extraction:** Figure 2.17(a) shows the training image and Figure 2.18(a) shows the testing image. The generational GP is used to generate the composite operator and the results from the 7th run are reported The fitness value of the best composite operator in the initial population is 0.52 and the population fitness value is 0.38. The fitness value of the best composite operator in the final population is 0.78 and the population fitness value is 0.77. The fitness of the extracted ROI is 0.88. Figure 2.39(a) shows the output image of the best composite operator in the final population and Figure 2.39(b) shows the extracted ROI. We apply the composite operator to the testing image. Figure 2.39(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.81.

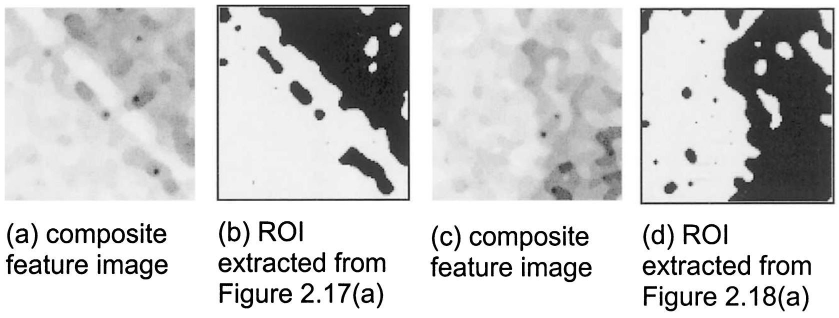

(a) composite feature image
(b) ROI extracted from Figure 2.17(a)
(c) composite feature image
(d) ROI extracted from Figure 2.18(a)

Figure 2.39. Results on SAR images containing field.

- **Tank extraction:** Figure 2.19(a) shows the training image and Figure 2.22(a) shows the testing image. The generational GP is used to generate the composite operator and the results from the 6th run are reported. The fitness value of the best composite operator in the initial population is 0.65 and the population fitness value is 0.17. The fitness value of the best composite operator in the final population is 0.88 and the population fitness value is 0.87. The fitness of the extracted ROI is 0.88. Figure 2.40(a) shows the output image of the best composite operator in the final population and Figure 2.40(b) shows the extracted ROI. We apply the composite operator to the testing image. Figure 2.40(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.84.

The best composite operator has 28 nodes and its depth is 17. It has four leaf nodes with two containing a 3×3 minimum image, one containing a 7×7 maximum image and one containing a 7×7 minimum image. It is shown in Figure 2.41. Figure 2.42 shows how the average fitness of the best composite operators and the average fitness of the populations over all the 10 runs change as GP proceeds.

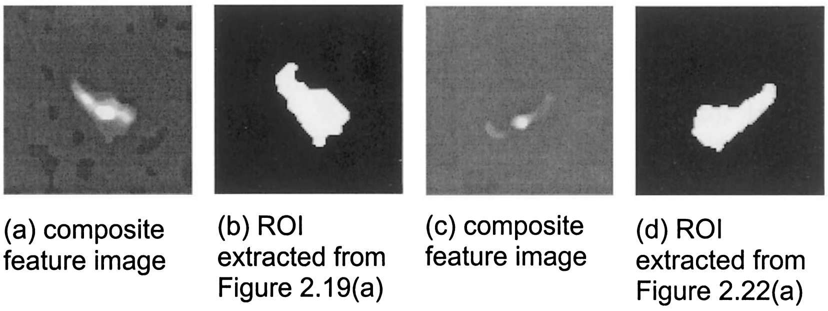

(a) composite feature image
(b) ROI extracted from Figure 2.19(a)
(c) composite feature image
(d) ROI extracted from Figure 2.22(a)

Figure 2.40. Results on SAR images containing tank.

```
(MED (MED (MUL (MIN PFIM10) (MUL
(MAX PFIM12) (MIN2 (MAX (SUBC
(DIVC (MIN (MEAN PFIM9))))) (SUBC
(MED (SUBC (MAX (MAX (SUBC (MAX
(MAX (SUBC (MAX (MAX (SUBC
PFIM10))))))))))))))))))
```

Figure 2.41. Learned composite operator tree in LISP notation.

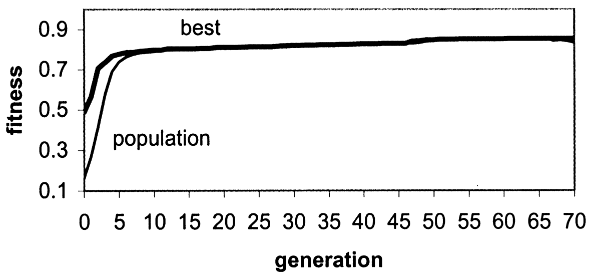


Figure 2.42. Fitness versus generation (T72 tank).

Comparing Table 2.3 and Table 2.5, we find that there is not much difference between the soft and hard size limits since both use a limit of 30 as the size of a composite operator. The only difference between them is that in the case of the soft size limit an exception is allowed if the fitness of an individual is the highest in the population.

2.4.4 Comparison with image-based GP

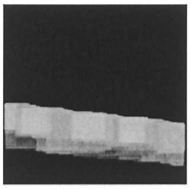
This subchapter is an advancement to our previous work in [17], where we also applied genetic programming to learn composite operators for object detection. The three major differences between the method presented here and that in [17] are:

1. Unlike [17] where a whole training image is used during training (Image-based GP), here GP runs on carefully selected region(s) (Region-based GP) to reduce the training time.
2. The hard size limit on the composite operator is replaced by the soft size limit in this chapter. This removes the restriction on the selection of crossover point in the parent composite operators to improve the search efficiency of GP, as stated in chapter 2.3.2.
3. Only the first mutation type in chapter 2.3.2 and only the first seven primitive feature images are used in [17]. With more mutation types and more primitive feature images used, the diversity of the composite operator population can be further increased.

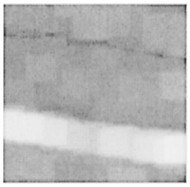
We summarize the experimental results on SAR images in [17] for the purpose of comparison. The parameters are: the same population size (100), 100 generations (vs. 70), the same fitness threshold value (1.0), the same crossover rate (0.6), 0.1 mutation rate (vs. 0.05), a hard limit of 30 on the maximium size (number of internal nodes) of composite operators (vs. soft limit of 30), and the same segmentation threshold (0). In each experiment, GP is invoked ten times with the same parameters. The experimental results from the run in which GP finds the best composite operator among the composite operators found in all ten runs and the average performance of ten runs are shown in Table 2.6.

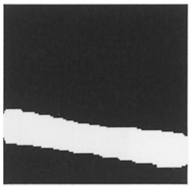
• **Road extraction:** In this case, the entire training image (shown in Figure 2.3(a)) is used and the same testing images in Figure 2.10(a), (d) are used. The generational GP was used to generate a composite operator to extract the road. The fitness value of the best composite operator in the initial population is 0.47 and the population fitness value is 0.19. The fitness value of the best composite operator in the final population is 0.92 and the population fitness value is 0.89. Figure 2.43(a) shows the output image of the best composite operator in the final population and Figure 2.43(b) shows the extracted ROI. We applied the composite operator obtained in the above training to the two testing SAR images. Figure 2.43(c) and (d) show the output image of the composite operator and the ROI extracted from Figure 2.10(a). The fitness value of the extracted ROI is 0.89. Figure 2.43(e) and (f) show the output image of the composite operator and the ROI extracted from Figure 2.10(d). The fitness value of the extracted ROI is 0.92.

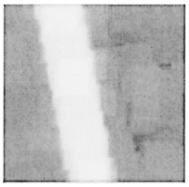
• **Lake extraction:** The entire training image in Figure 2.11(a) is used and Figure 2.12(a) shows the testing image. The steady-state GP was used to generate the composite operator. The fitness value of the best composite operator in the initial population is 0.65 and the population fitness value is 0.42. The fitness value of the best composite operator in the final population is 0.93 and the population fitness value is 0.92. Figure 2.44(a) shows the output image of the best composite operator in the final population and Figure 2.44(b) shows the extracted ROI. We applied the composite operator to the testing SAR image. Figure 2.44(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.92. In Figure 2.44(a) and (c), pixels in the small dark regions have very low pixel values (negative values with very large absolute values), thus making many pixels appear bright, although some of them have negative pixel values.


(a) composite feature image

(b) ROI extracted from Figure 2.3(a)


(c) composite feature image


(d) ROI extracted from Figure 2.10(a)


(e) composite feature image

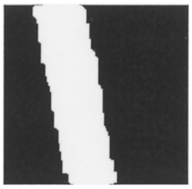
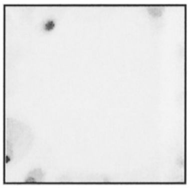
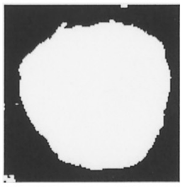
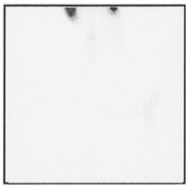

(f) ROI extracted from Figure 2.10 (d)

Figure 2.43. Results on SAR images containing road.


(a) composite feature image


(b) ROI extracted from Figure 2.11(a)


(c) composite feature image

(d) ROI extracted from Figure 2.12(a)

Figure 2.44. Results on SAR images containing lake.

Table 2.6. The performance results of image-based GP on various SAR images.

Training performance								
	Road		Lake		River		Field	
	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p	f_{op}	f_p
Ave. $f_{initial}$	0.47	0.18	0.73	0.39	0.37	0.11	0.65	0.41
Ave. f_{final}	0.81	0.76	0.92	0.87	0.68	0.58	0.84	0.77
$f_{initial}$	0.47	0.19	0.65	0.42	0.43	0.21	0.62	0.44
f_{final}	0.92*	0.89	0.93*	0.92	0.74*	0.68	0.87*	0.86
Testing performance								
	Road		Lake		River		Field	
f_{test}	0.89, 0.92		0.92		0.84		0.68	

f_{op}: fitness of the best composite operator on selected region(s),
f_p : fitness of population on selected region(s),
*: indicate finess on the *entire* training images,
$f_{initial}$: fitness of the initial generation on selected region(s),
f_{final}: fitness of the final population on selected region(s),
f_{test}: fitness of the best composite operator on the *entire* testing images.

- **River extraction:** The entire training image in Figure 2.13(a) is used and Figure 2.16(a) shows the testing image. The steady-state GP was used to generate the composite operator. The fitness value of the best composite operator in the initial population is 0.43 and the population fitness value is 0.21. The fitness value of the best composite operator in the final population is 0.74 and the population fitness value is 0.68. Figure 2.5(a) shows the output image of the best composite operator in the final population and Figure 2.5(b) shows the extracted ROI. We applied the composite operator to the testing image. Figure 2.5(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.84. Like Figure 2.44(c), pixels in the small dark region have very low pixel values, thus making many pixels with negative pixel values appear bright.

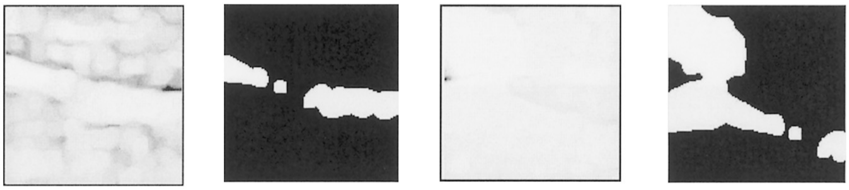

(a) composite feature image
(b) ROI extracted from Figure 2.13(a)
(c) composite feature image
(d) ROI extracted from Figure 2.16(a)

Figure 2.45. Results on SAR images containing river.

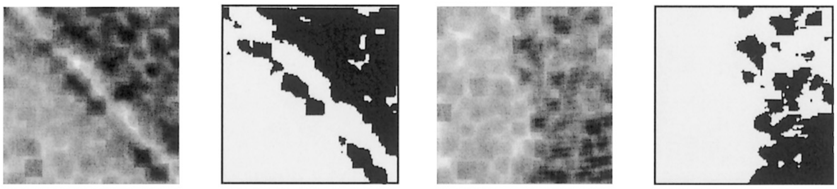

(a) composite feature image
(b) ROI extracted from Figure 2.17(a)
(c) composite feature image
(d) ROI extracted from Figure 2.18(a)

Figure 2.46. Results on SAR images containing field.

• **Field extraction:** In this case, the entire training image in Figure 2.17(a) is used and Figure 2.18(a) shows the testing image. The generational GP was used to generate the composite operator. The fitness value of the best composite operator in the initial population is 0.62 and the population fitness value is 0.44. The fitness value of the best composite operator in the final population is 0.87 and the population fitness value is 0.86. Figure 2.46(a) shows the output image of the best composite operator in the final population and Figure 2.46(b) shows the extracted ROI. We applied the composite operator to the testing image. Figure 2.46(c) and (d) show the output image of the composite operator and the extracted ROI with fitness value 0.68.

Table 2.7. Average training time of region GP and image GP (in seconds).

	Road	Lake	River	Field
Region GP	12876	2263	6560	9685
Image GP	23608	9120	66476	21485

From Table 2.3 and associated figures, it can be seen that if the carefully selected training regions represent the characteristics of training images, the composite operators learned by GP running on training regions are effective in extracting the ROIs containing the object and their performance is comparable to the performance of composite operators learned by GP running on whole training images. By running on the selected regions, the training time is greatly reduced. Table 2.7 shows the average training time of GP running on selected regions (Region GP) and GP running on the whole training images (Image GP) over all ten runs. Since the number of generations in [17] is 100 and the number of generations in this chapter is 70, the training time of "Image GP" stated in Table 2.7 is normalized as the actual training time of "Image GP" times 0.7. It can be seen that the training time using selected training regions is much shorter than that using the whole image.

2.4.5 Comparison with a traditional ROI extraction algorithm

To show the effectiveness of composite operators in ROI extraction, they are compared with a traditional ROI extraction algorithm. The *traditional* ROI extraction algorithm uses a threshold value to segment the image into foreground and background. The region consisting of pixels with value greater than the threshold value is called the bright region and its complement is called the dark region. If the bright region has a higher fitness than the dark region, the bright region is the foreground. Otherwise, the dark region is the foreground. The foreground is the ROI extracted by this traditional algorithm. The threshold value plays a vital role in the ROI extraction and selecting an appropriate threshold value is the key to the success of this traditional ROI extraction algorithm. The performance of composite operators is compared with that of the traditional ROI extraction algorithm when the best threshold value is used. To find the best threshold value, every possible threshold value is tried by the algorithm and its performance is recorded. In order to show the effectiveness of composite features over that of primitive features, the traditional ROI extraction algorithm is applied to all the 16 primitive feature images (for SAR and IR images) or the 3 primitive feature images (RED, GREEN AND BLUE planes of RGB color images), and the best result from the 16 or 3 primitive feature images is recorded in Table 2.8 and Figure 2.47.

The Traditional ROI Extraction Algorithm

```
1. find the maximum and minimum pixel values of the image.
2. if the maximum pixel value is greater than 1000
3.     normalize the pixel values into the range of 0 to 1000. The pixel values
       are changed according to the following equation.
       new_pixval = (org_pixval – min_pixval) / (max_pixval – min_pixval) *
       1000
       where new_pixval and org_pixval are the new and original pixel values,
       respectively and min_pixval and max_pixval are the minimum and
       maximum pixel values in the original image. After normalization, the
       minimum and maximum pixel values are 0 and 1000, respectively.
     else
4.     do not normalize the image.
     endif
5. each integer value between the minimum and maximum pixel values is
   used as the threshold value and its performance in ROI extraction is
   recorded.
6. select the best threshold value and output its corresponding ROI.
```

The fitness values of the extracted ROIs and their corresponding threshold values are shown in Table 2.8. Figure 2.47 shows the ROIs extracted by the traditional ROI extraction algorithm corresponding to the best threshold value. For the purpose of comparison, Figure 2.48 shows the corresponding performance of the GP-learned composite operators on the same images. From Figure 2.47, Figure 2.48 and Table 2.8, it is clear that the composite operators learned by GP are more effective in ROI extraction. Actually, its performance is better than the best performance of the traditional ROI extraction algorithm in all the examples except a couple of them where there is a minor difference. Table 2.9 shows the average running time of the composite operators and the traditional ROI extraction algorithm in extracting ROIs from training and testing images. From Table 2.9, it is obvious that the composite operators are more efficient.

Table 2.8. Comparison of the performance of traditional ROI extraction algorithm and composite operators generated by GP.

Examples	Traditional techiniue				GP-based technique	
	Thresold value	Primitive feature	Fig. No.	Fitness	Fitness	Fig. No.
paved road vs. field	31	PFIM15	2.47(a)	0.68	0.93	2.48(a)
unpaved road vs. field	32	PFIM15	2.47(b)	0.72	0.90	2.48(b)
paved road vs. grass	31	PFIM15	2.47(c)	0.82	0.93	2.48(c)
lake vs. field	32.9	PFIM3	2.47(d)	0.92	0.93	2.48(d)
lake vs. grass	30	PFIM15	2.47(e)	0.95	0.98	2.48(e)
river vs. field	45	PFIM15	2.47(f)	0.62	0.71	2.48(f)
river vs. field	45.5	PFIM15	2.47(g)	0.84	0.83	2.48(g)
field vs. grass	72.3	PFIM3	2.47(h)	0.82	0.89	2.48(h)
field vs. grass	196	PFIM9	2.47(i)	0.71	0.80	2.48(i)
T72 tank	99.6	PFIM2	2.47(j)	0.86	0.88	2.48(j)
T72 tank	82.2	PFIM2	2.47(k)	0.86	0.84	2.48(k)
person	96	PFIM1	2.47(l)	0.84	0.85	2.48(l)
person	94	PFIM13	2.47(m)	0.83	0.84	2.48(m)
person	99	PFIM7	2.47(n)	0.81	0.81	2.48(n)
person	95	PFIM1	2.47(o)	0.84	0.86	2.48(o)
car	101	PFIM2	2.47(p)	0.45	0.82	2.48(p)
car	107	PFIM2	2.47(q)	0.41	0.76	2.48(q)
SUV	68	PFIM1	2.47(r)	0.30	0.69	2.48(r)
SUV	106	PFIM1	2.47(s)	0.44	0.58	2.48(s)

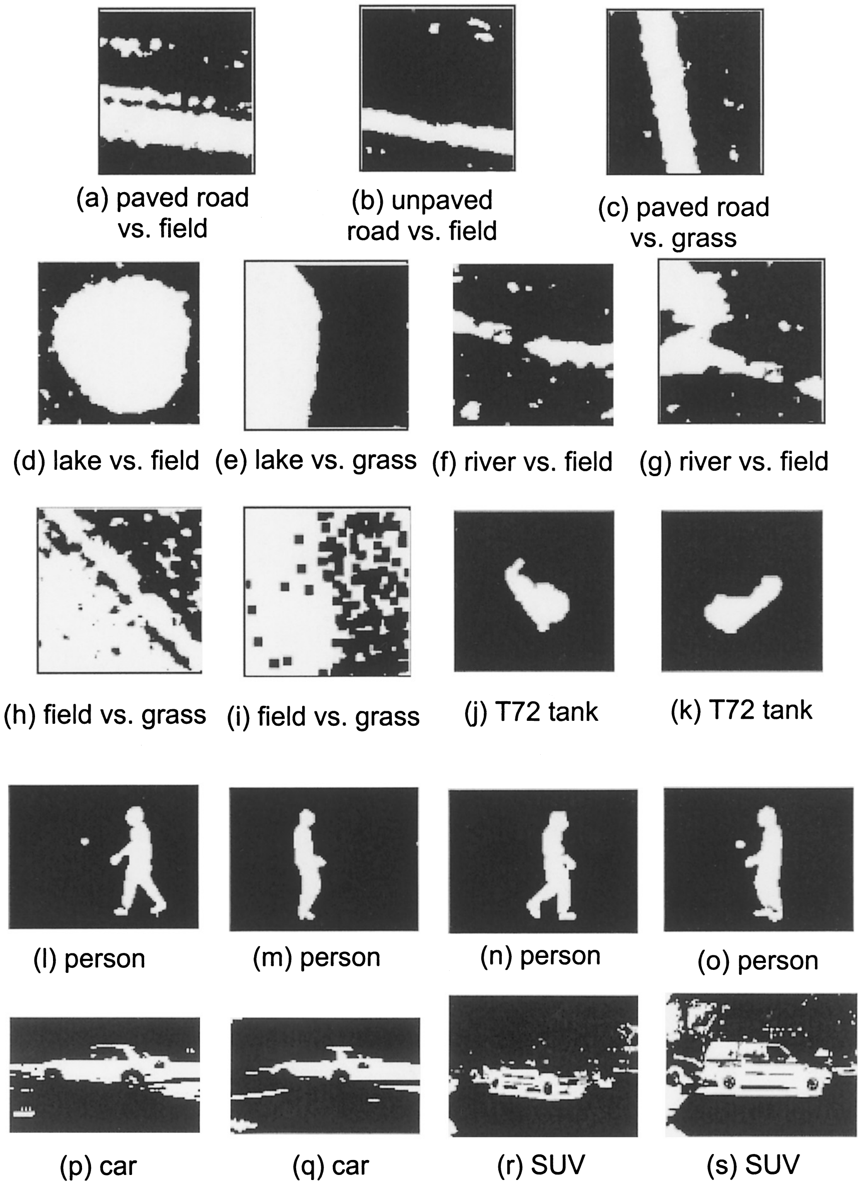


Figure 2.47. ROIs extracted by the traditional ROI extraction algorithm.

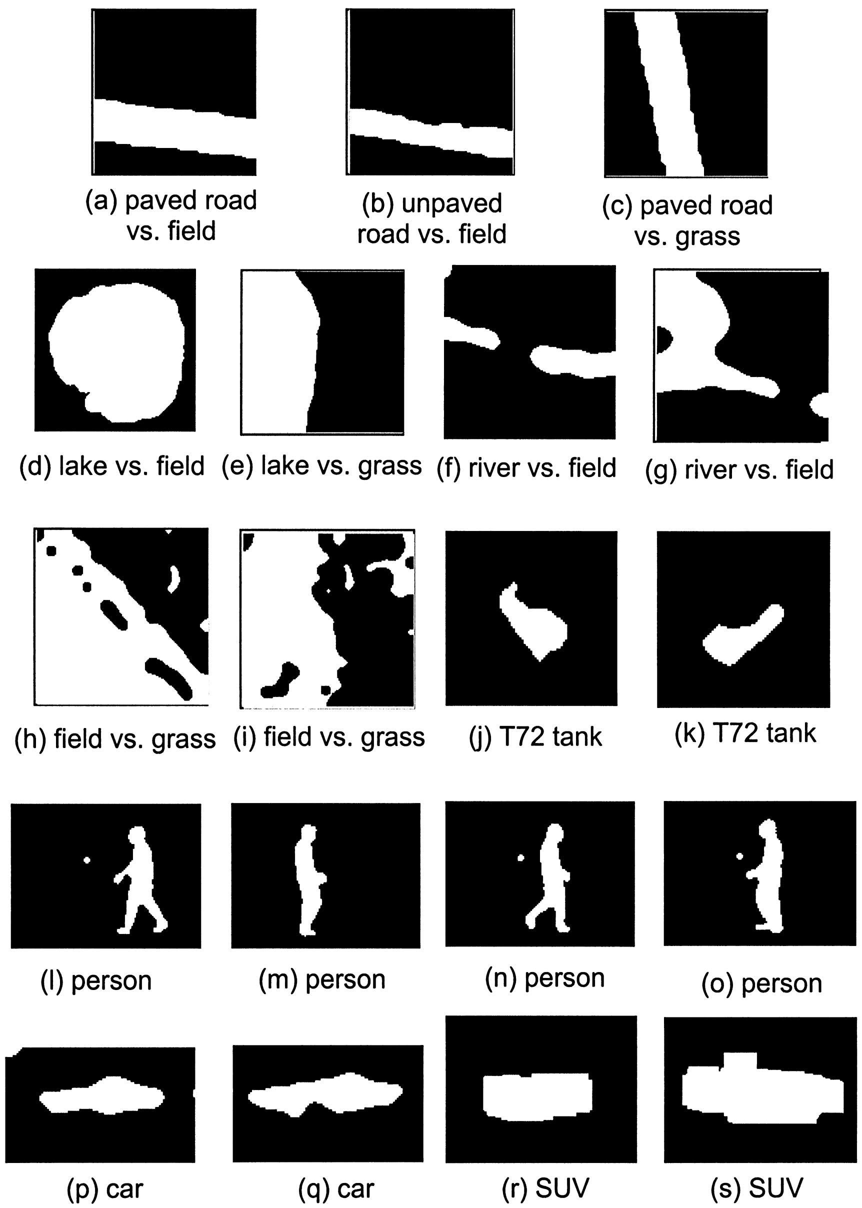


Figure 2.48. ROIs extracted by the GP-evolved composite operators.

Table 2.9. Average running time (in seconds) of the composite operators and the traditional ROI extraction algorithm.

	Road	Lake	River	Field	Tank	Person	Car	SUV
Composite operator	5	15	33	8	3	1	2	6
Traditional ROI exaction algorithm	12.3	10	50.5	32	23	3.4	5.5	20.5

2.4.6 A multi-class example

In the above examples, we showed the effectiveness and efficiency of composite operators learned by GP in ROI extraction. In this section, a complicated SAR image (shown in Figure 2.49(a)) containing lake, road, field, tree and shadow is used as a testing image. Note that shadow is an unknown region (reject class, not considered in this chapter) in this example. Figure 2.49(b), (c) and (d) show the ground-truth for lake, road and field.

We apply the composite operators for lake, road and field learned in Examples 2, 1 and 4, respectively, to the above testing image. The lake operator is applied first; then the road operator is applied to the rest of the image excluding the lake ROIs; finally, the field operator is applied to the rest of the image excluding both lake and road ROIs. The ROIs extracted are shown in Figure 2.50. The fitness values are 0.85, 0 and 0.75, respectively. These results are not promising. Since the pixel values of road and lake regions are quite similar (see Figure 2.53, many pixels in the road and lake regions have values between 0 and 20), the lake composite operator extracts part of the road and the road composite operator extracts no road pixel. In order to force GP to learn composite operators that can distinguish the subtle difference between the lake and road pixels, a SAR image (shown in Figure

2.52) containing both lake and road is used as a training image. Figure 2.52(a) shows the original image with the training regions from (4, 96) to (124, 119) and from (2, 25) to (127, 86) used by GP to learn composite operators for the lake extraction. To learn composite operators for the road extraction, the same image with the training region from (90, 30) to (135, 117) shown in Figure 2.52(c) and the training image used in Example 1 are used for training. Figure 2.52(b) and (d) show the ground-truth for the lake and road, respectively. Note that the images in Figure 2.49(a) and Figure 2.52(a) are quite different.

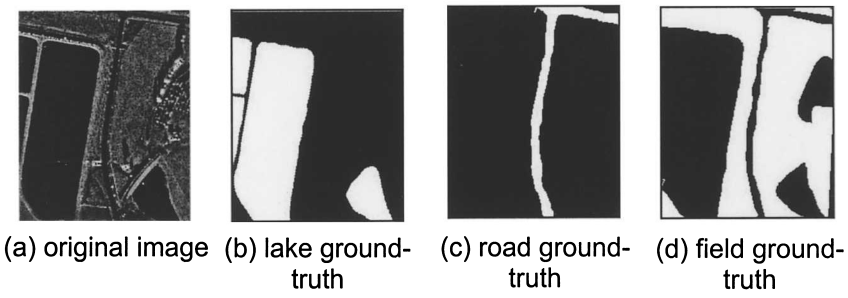


Figure 2.49. SAR image containing lake, road, field, tree and shadow.

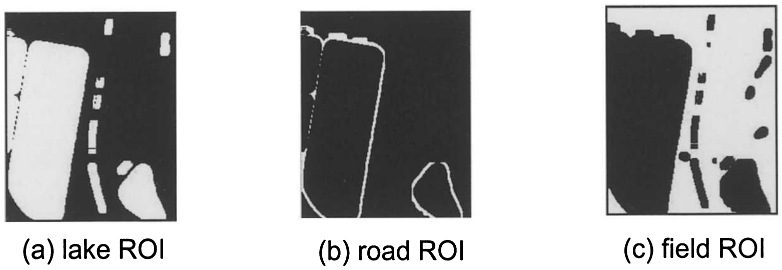
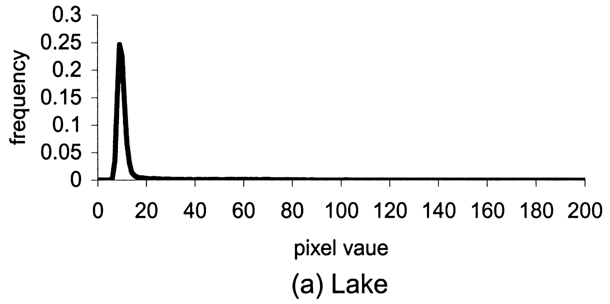


Figure 2.50. Lake, road and field ROIs extracted by the composite operators learned in Examples 1, 2 and 4.



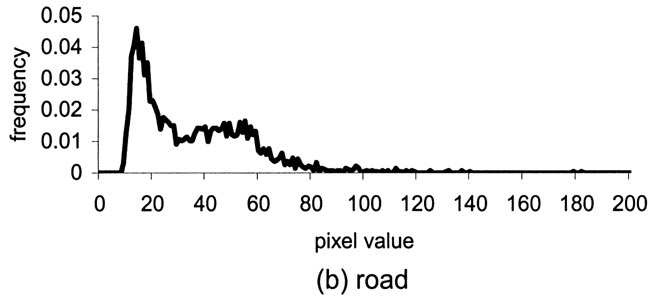


Figure 2.51. Histogram of pixel values (range 0 to 200) within lake and road regions.

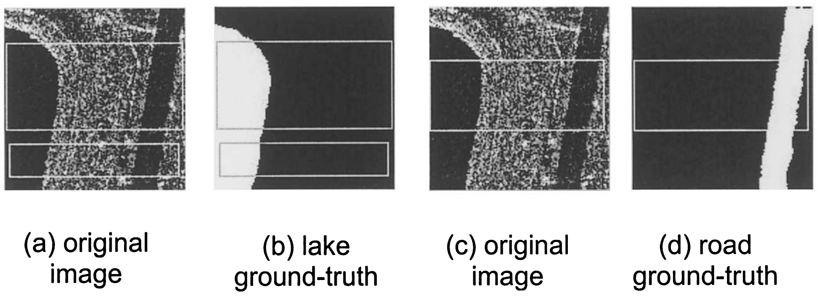


Figure 2.52. SAR image containing lake and road.

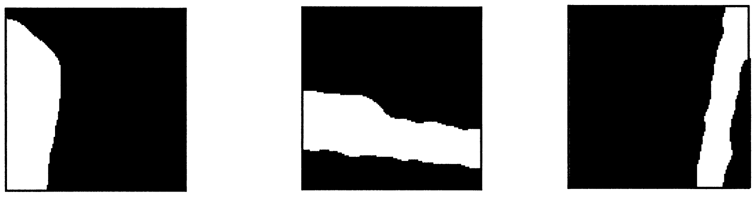


Figure 2.53. lake and road ROIs extracted from training images.

The steady-state GP is applied to synthesize composite operators for lake detection and the ROI extracted by the learned composite operator from the training image is shown in Figure 2.53(a). The fitness value of the extracted ROI is 0.95. The generational GP is applied to synthesize composite operators for road detection. The ROIs extracted by the learned composite operator from the training images (Figure 2.3(a) and Figure 2.52(c)) are shown in Figure 2.53(b) and (c). The fitness values are 0.78 and 0.90, respectively.

We apply the newly learned lake and road composite operators to the testing image in Figure 2.49(a). The extracted lake and road ROIs are shown in Figure 2.54(a) and (b). The fitness values of the extracted ROIs are 0.93 and 0.46, respectively. After extracting the lake and road from the image, we exclude the regions corresponding to the extracted lake and road ROIs and apply the field composite operator learned in Example 4 to the rest of the image. The extracted ROI is shown in Figure 2.54(c) and its fitness value is 0.81. The running times are 53, 127 and 26 seconds, respectively for the results shown in Figure 2.54.

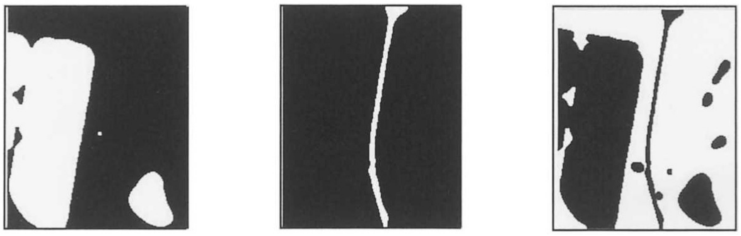


Figure 2.54. Lake, road and field ROIs extracted from the testing image.

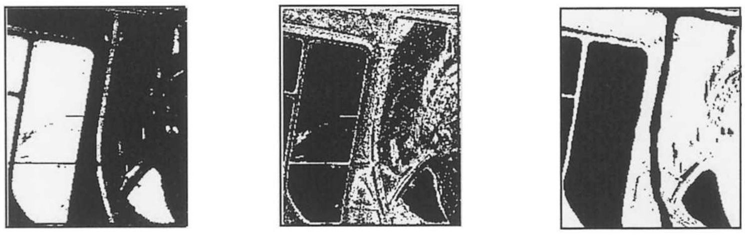


Figure 2.55. Lake, road and field ROIs extracted by the traditional algorithm.

Finally, the traditional ROI extraction algorithm is applied to the above testing image. The extracted ROIs, corresponding to the best threshold values, of the lake, road and field are shown in Figure 2.55(a), (b) and (c), respectively. To extract road, the regions corresponding to the extracted lake ROIs are removed and the algorithm is applied to the rest of the image. Figure 2.55(b) demonstrates that it is very difficult for the traditional ROI extraction algorithm to distinguish road from field in SAR images. To extract field, the regions corresponding to the ground-truth of the lake and road (not the ROIs corresponding to the lake and road) are excluded. The reason that we do not use extracted ROIs is that the extracted road ROIs are very bad. The fitness values of the extracted lake, road and field ROIs are 0.86, 0.15 and 0.79, respectively. The best threshold values are 16, 55 and 28.5, and the running times are 192, 176 and 195 seconds, respectively. It can be seen that the GP learned composite operators are more effective in the lake, road and field detection, compared to the traditional ROI extraction algorithm.

2.5 Conclusions

In this chapter, we use genetic programming to synthesize composite operators and composite features to detect potential objects in images. We use a soft composite operator size limit to avoid code-bloating and severe restrictions on GP search. We also compare it with the hard limit on composite operator size. Our experimental results show that the primitive operators and primitive features defined by us are effective. GP can synthesize effective composite operators for object detection by running on the carefully selected training regions of images and the synthesized composite operators can be applied to the whole training images and other similar testing images. We do not find significant difference between generational and steady-state genetic programming algorithms.

As discussed, GP has code bloat problem. Controlling code bloat due to the limited computational resources inevitably restricts the search efficiency of GP. How to reach the balance between the two conflicting factors (size of the composite operator and performance) is critical in the implementation of GP. In the next chapter, we address this problem by designing a new fitness function based on the minimum description length (MDL) principle to incorporate the size of composite operators into the fitness evaluation process.

Chapter 3

MDL-BASED EFFICIENT GENETIC PROGRAMMING FOR OBJECT DETECTION

3.1 Introduction

In chapter 2, the efficacy of genetic programming in learning composite features for object detection is demonstrated. The motivation for using GP is to overcome the human experts' limitation of considering only a very limited number of conventional combinations of primitive features. Chapter 2 shows that GP is an effective way of synthesizing composite features from primitive ones for object detection. However, genetic programming is computationally expensive. In the traditional GP with hard limit on the individual size (also called a *normal GP*), crossover and mutation locations are randomly selected, leading to the disruption of the effective components (subtree in this approach) of composite operators especially at the later stage of the GP search. This greatly reduces the efficiency of GP. It is very important for GP to identify and keep the effective components of composite operators to improve the efficiency. In this chapter, smart crossover and smart mutation are proposed to better choose crossover and mutation points to prevent effective components of a composite operator from being disrupted. Also, a public library is established to save the effective components of composite operators for later reuse. Finally, a fitness function based on the minimum description length (MDL) principle is designed to incorporate the size of a composite operator into the fitness evaluation to address the well-known code bloat problem of GP without imposing severe restrictions on the GP search. The GP with smart crossover, smart mutation and MDL-based fitness function is called a *smart GP*.

3.2 Motivation and Related Research

Crossover and mutation are two major mechanisms employed by GP to search the composite operator space (also called feature combination space). As GP proceeds, effective components are generated. The power of crossover lies in the fact that by swapping sub-trees between two effective composite operators (parents), the effective components (sub-trees) in these two parents can be assembled together into child composite operators (offspring) and the new offspring may be better than both parents. However, although crossover can assemble good components to yield better offspring, it is also a destructive force in the sense that it can disrupt good components due to the random selection of crossover points. When the search begins, since the initial population is randomly generated, it is unlikely that a composite operator contains large good components and the probability of crossover breaking up a good component is small. At this time, crossover is a constructive force and the fitness of a composite operator is increased. As search proceeds, small good components are generated and assembled into larger and larger good components. When more and more composite operators contain large good components to achieve high fitness, the good component accounts for a large portion of a composite operator and the composite operator becomes more and more fragile because the good components are more prone to being broken up by subsequent crossover due to the random selection of crossover points. The crossover can damage the fitness of a composite operator in ways other than disrupting good components. Sometimes, a good component is moved into an inhospitable context, that is, the crossover inserts a good component into a composite operator that does not use the good component in any useful way or other nodes of the composite operator cancel out the effect of the good component. According to [82], crossover has an overwhelmingly negative effect on the fitness of the offspring from crossover, especially in the later stage of GP search.

Mutation is introduced to maintain the diversity of a population, since a serious weakness of evolutionary algorithms is that the population recombined repeatedly will develop uniformity sooner or later [82]. However, in the later stage of GP search when more and more composite operators contain large good components, the random selection of mutation points leads to a high probability of disrupting good components and makes mutation a destructive force. When both crossover and mutation become negative factors in the GP search, it is very unlikely that better composite operators will be generated. To improve the efficiency and effectiveness of GP, it is highly beneficial if good components can be identified and kept from destructive crossover and mutation operations and stored in a public library for later reuse. These components are treated as atomic terminals and are directly inserted into composite operators as a whole when the mutations are performed or during initialization.

GP has a well-known code bloat problem in which the sizes of individuals become larger and larger. In normal GP with individuals represented by tree structures, a crossover operation is performed by swapping sub-trees rooted at the randomly selected nodes called crossover points, and one of the mutation operations is performed by substituting a randomly selected sub-tree with another randomly generated tree. It is easy to see that the size of one offspring (i.e., the number of nodes in the binary tree representing the offspring) may be greater than both parents if crossover and mutation are performed in this simple way. If we do not control the sizes of composite operators, they will become larger and larger as GP proceeds, as stated in chapter 2. When the size becomes too large, it takes a long time to execute a composite operator, greatly reducing the speed of GP. Also, large-size composite operators may overfit training data by approximating the noise in images. Although the result on the training image is very good, the performance on unseen testing images may be bad. Finally, large composite operators take up a lot of computer memory.

Usually in normal GP, a limit on the size of composite operators is established when performing crossover or mutation. If the size of an offspring exceeds the size limit, the crossover or mutation operation is performed again until the sizes of both offspring are within the limit. Although this simple method prevents the code bloat, the size limit may greatly restrict the search performed by GP [17], since after randomly selecting a crossover point in one composite operator, GP cannot select some nodes of the other composite operator as crossover point in order to guarantee that both offspring do not exceed the size limit. Also, the size limit restricts the size of trees used to replace sub-trees in mutation. However, the composite operator space is huge [17], and to find effective composite operators, GP must search extensively. Restricting the search greatly reduces the efficiency of GP, making it less likely to find good composite operators. To allieviate the restrictions, in chapter 2, the soft limit on the composite operator size is proposed. With a soft size limit, GP allows the generation of large compsite operators above the size limit and keeps those large composite operators only when they are the best, or with performance very close the best composite operator(s) in the population, thus taking off the restrictions on the selection of crossover and mutation points without causing code bloat. However, with little knowledge on the composite operator space and the object characteristics, it is very difficult, if not impossible, to determine the appropriate hard or soft size limit to prevent code bloat and overfitting while allowing the resulted composite operators to capture the characteristics of objects. How to avoid restricting the GP search without causing code bloat is the key to the success of GP search. Also, with little knowledge on the objects to be detected, it is critical for GP to automatically determine the appropriate size of composite operators that are needed to capture the characteristics of objects. In this chapter, a fitness function is designed based on the minimum description length (MDL) principle [100] to take the size of a composite operator into the fitness evaluation process. According to the MDL principle, large composite operators effective on training regions may not have good fitness and will be culled out by selection. Thus, we can take off the restriction on crossover and mutation while preventing composite operators from growing too large.

To improve the efficiency of GP, Tackett [116] devises a method called brood recombination to reduce the destructive effect of crossover. In this method, when crossover is performed, many offspring are generated from two parents and only the best two offspring are kept. D'haeseleer [24] devises strong context preserving crossover (SCPC) to preserve the context. SCPC only permits crossover between nodes that occupied exactly the same position in the two parents. He finds modest improvement in results by mixing regular crossover and SCPC. Smith [111] proposes a conjugation operator for GP to transfer genetic information from one individual to another. In his conjugation method, the parent with higher fitness becomes the donor and the other with lower fitness becomes the recipient. The conjugation operator is different from crossover and it simulates one of the ways in which individuals exchange genetic materials in nature. Ito et al. [48] propose a depth-dependent crossover for GP in which the depth selection ratio is varied according to the depth of a node. A node closer to the root node of a tree has a better chance of being selected as a crossover point to lower the chance of disrupting small good components near leaves. Their experimental results show the superiority of the depth-dependent crossover to the random crossover in which crossover points are randomly selected. Bhanu and Lin [16], [69] propose smart crossover and mutation operators to identify and keep the good components of composite operators. Their initial experiments show that with smart GP operators, GP can search the composite operator space more efficiently.

Unlike the work of Ito [48] that used only the syntax of a tree (the depth of a node), the smart crossover and smart mutation proposed in this chapter evaluate the performance of each node to determine the interactions among them and use the fitness values of the nodes to determine crossover and mutation points. Also, unlike our previous work [16], a public library is introduced to keep the good components for later reuse and more types of mutations are added to increase the population diversity. Nine more primitive feature images are included to build composite operators. To reduce the training time, the training in this chapter is performed on the selected regions of training images, not the whole images as in the previous work. More importantly, a new MDL-based fitness function is designed to reach a balance point between the conflicting factors of code bloat and less restriction on the GP search.

Quinlan and Rivest [97] explore the use of the minimum description length principle for the construction of decision trees. The MDL defines the best decision tree to be the one that yields the minimum combined length of the decision tree itself plus the description of the misclassified data items. Their experimental results show that the MDL provides a unified framework for both growing and pruning the decision tree, and these trees seem to compare favorably with those created by other techniques such as C4.5 algorithm. Gao et al. [34] use the MDL principle to determine the best model granularity such as the sampling interval between the adjacent sampled points along the curve of Chinese characters or the number of nodes in the hidden layer of a three layer feed-forward neural network. Their experiments show that in these two quite different settings the theoretical value determined by the MDL principle coincides with the best value found experimentally. The key point of their work is that using the MDL principle, the optimal granularity of the model parameters can be computed automatically rather than being tuned manually. In this chapter, a fitness function is designed based on the MDL principle to incorporate the size of composite operators into the fitness evaluation to address the code-bloat problem without imposing severe restrictions on the GP search.

3.3 Improving the Efficiency of GP

The primitive feature images and primitive operators are the same as those used in chapter 2. There are 16 primitive feature images: the original image (0), mean (1–3), deviation (4–6), maximum (7–9), minimum (10–12) and median (13–15) images obtained by applying templates of sizes 3×3, 5×5 and 7×7. 17 primitive operators are ADD, SUB, MUL, DIV, MAX2, MIN2, ADDC, SUBC, MULC, DIVC, SQRT, LOG, MAX, MIN, MED, MEAN and STDV. The key parameters are the population size M, the number of generation N, the crossover rate, the mutation rate and the goodness threshold (defined in chapter 3.3.1). The GP stops whenever it finishes the pre-specified number of generations or whenever the best composite operator in the population has goodness value greater than the goodness threshold.

3.3.1 MDL principle-based fitness function

To address the code bloat problem and prevent severe restriction on the GP search, we design a MDL-based fitness function to incorporate the composite operator size into the fitness evaluation process. The fitness of a composite operator is defined as the sum of the description length of the composite operator and the description length of the segmented training regions with respect to this composite operator as a predictor for the label (object or background) of each pixel in the training regions. Here, both lengths are measured in bits and the details of the coding techniques are relevant. The trade-off between the simplicity and complexity of composite operators is that if the size of the composite operators is too small, it may not capture the characteristics of objects to be detected, on the other hand, if the size is too large, the composite operator may overfit the training image, thus performing poorly on the unseen testing images. With the MDL-based fitness function, the composite operator with the minimum combined description lengths of both the operator itself and image-to-operator error is the best composite operator and may perform best on the unseen testing images. Based on minimum description length principle, we propose the following fitness function for GP to maximize:

$$F(CO_i) = -\,(r \times \log (N_{po}) \times Size(CO_i) +$$

$$(1 - r) \times (n_o + n_b) \times (\log(W_{im}) + \log(H_{im}))) \qquad (3.1)$$

where CO_i is the *i*th composite operator in the population, N_{po} is the number of primitive operators (including primitive feature images) available for GP to synthesize composite operators, $Size(CO_i)$ is the size of the composite operator which is the number of nodes in the binary tree representing it, n_o and n_b are the number of object and background pixels misclassified, W_{im} and H_{im} are the width and height of the training image and r is a parameter determining the relative importance of the composite operator size and the detection rate, which is 0.7 in this chapter. The value $r = 0.7$ is selected experimentally. In our experiments, we find 0.7 is an appropriate value to balance the composite operator size and its performance. Note that the first term of the fitness function is the description length of the composite operator. The description length is the number of bits needed to encode a composite operator and it is not the size of a composite operator (the number of nodes in the composite operator). However, the description length is closely related to the size of a composite operator. The larger the size of a composite operator, the longer is its description length.

We now give a brief explanation of this fitness function. Suppose a sender and a receiver both have the training image and the training regions and they agree in advance that composite operators can be used to locate the object in the image, that is, to determine the label (object or background) of each pixel in the training regions. But only the sender knows the ground-truth (the label of each pixel). Now, the sender wants to tell the receiver which pixels belong to the object and which pixels belong to the background. One simple approach to do this is to send a bit sequence of n (n is the number of pixels in the training regions) bits where 1 represents the object and 0 represents background, provided that both the sender and the receiver know the order of the training regions and they agree that the pixels are scanned in the top-to-bottom and left-to-right fashion. However, n is usually very large, thus the communication burden is heavy. To reduce the number of bits to be transmitted, the sender can send the composite operator to the receiver. Then the receiver applies the composite operator on the training regions to get segmented training regions. When sending the composite operator, the sender can send its nodes in a preorder traversal. Given N_{po} primitive operators (including primitive features), $\log(N_{po})$ bits are needed to encode each node. Thus, the cost of sending composite operator is $\log(N_{po}) \times \mathrm{Size}(CO_i)$. However, some pixels may be misclassified by the composite operator. In order for the receiver to get the truth, the sender needs to tell the receiver which pixels are misclassified. Each pixel is represented by its coordinate in the image. If the width and height of the image are W_{im} and H_{im} respectively, then $\log(W_{im})+\log(H_{im})$ bits are needed to encode each pixel. Thus, the cost of sending the misclassified pixels is $(n_o + n_b) \times(\log(W_{im})+\log(H_{im}))$. If the composite operator is very effective and its size is not too large, then only few pixels are misclassified and the number of bits to send is much smaller than n.

In chapter 2, the fitness function is defined as $n(G \cap G') / n(G \cup G')$, where G and G' are foregrounds in the ground-truth image and the resultant image of a composite operator respectively and $n(X)$ denote the number of pixels within the intersection of region X and the training region. It measures how the ground-truth and the detection results are overlapped. In this chapter, this measure is called the *goodness* of a composite operator. It is not used to drive smart GP, but only used to measure the effectiveness of a composite operator.

3.3.2 Genetic programming with smart crossover and smart mutation

The selection operation selects composite operators from the current population. In this chapter, as before, we use tournament selection with a tournament size equal to five.

In the normal GP, to perform crossover, two composite operators are selected on the basis of their fitness values. The higher the fitness value, the more likely the composite operator is selected for crossover. These two composite operators are called parents. One internal node in each of these two parents is randomly selected, and the two subtrees rooted at these two nodes are exchanged between the parents to generate two new composite operators, called offspring. The crossover is called random crossover due to the random selection of the crossover point. Usually, at the later stage of GP search, effective composite operators contain large effective components. These components are prone to be disrupted by random crossover, leading to a reduction in the efficiency of genetic programming.

To avoid this problem, we propose a smart crossover that can identify and keep the effective components. To define smart crossover, the output image of each node, not just the resultant image from the root node, is evaluated and its fitness value is recorded. Based on the node fitness values, we define the fitness of an edge as the fitness difference between the parent node and the child node linked by the edge. If the fitness value of a node is smaller than that of its parent node, the edge linking them is a good edge. Otherwise, the edge is labeled as a bad edge. During crossover, all the bad edges are identified and one of them is selected by random selection or roulette selection (based on the fitness of the bad edges) invoked with equal probability. The child node of the selected edge is the crossover point and the subtrees rooted at the crossover points are swapped between parents. If a composite operator has no bad edge, the crossover point is randomly selected.

Since GP evaluates the fitness of each node, GP knows the fitness of each component (subtree) of a composite operator. A public library is established to store effective components for later reuse by smart mutation. The larger the library, the more effective components can be kept for later reuse, but the likelihood of each effective component being reused is reduced. In this chapter, the size of the public library is 100. After the library is full, a new effective component replaces the worst one in the library if it is better than the replaced one.

To avoid premature convergence, mutation is introduced to randomly change the structure of some individuals to maintain the diversity of the populationComposite operators are randomly selected for mutation. In a normal GP, there are three types of random mutation, invoked with equal probability, that involve randomly selecting mutation points as described in chapter 2.3.2.

In the smart mutation, however, the mutation point is the parent or child node of a bad edge or a bad node whose goodness is below the average goodness of all the nodes in the tree. The mutation point is selected from those qualified nodes randomly or by roulette selection based on the goodness of bad edges or bad nodes. There are four smart mutations invoked with equal probability:

1. Select the parent node of a bad edge as the mutation point. If the parent node has only one child, it is deleted and the child node is linked to the grandparent node (parent node of the parent node). If no grand parent node exists, the child becomes the root node; if the parent node has two children, the parent node and the sub-tree rooted at the child with smaller goodness value are deleted and the other child is linked directly to the grand parent node. If no grand parent node exists, the child becomes the root node.
2. Select the parent node of a bad edge as the mutation point and replace the primitive operator stored in the node with another primitive operator of the same number of input as the replaced one.
3. Select two subtrees whose roots are child nodes of two bad edges within the composite operator and swap them. Neither of the two sub-trees can be the sub-tree of the other.
4. Select a bad node as the mutation point. Delete the subtree rooted at the node and replace it with another randomly generated tree or a randomly selected effective component from the public library.

The first two mutations delete a node that cancels the effect of its child or children; the third mutation moves two components away from unfriendly contexts that cancel their effects and inserts them into new contexts to see if the new contexts are appropriate to them; the fourth mutation deletes a bad component and replaces it with a new component or a good one stored in the public library.

We use an ε-greedy policy to determine whether a smart operator (smart crossover or mutation) or a random operator (random crossover or mutation) is used. With probability ε, the smart operator is invoked; with probability 1 - ε, the random operator is invoked. In this chapter, ε is a variable that is adjusted by the following formula:

$$\varepsilon = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}) \times Good_{popu} \qquad (3.2)$$

where ε_{min} is 0.5 and ε_{max} is 0.9, $Good_{popu}$ is population goodness (the average goodness of the composite operators in the current population). The reason for using the random operators is that smart operators bias the selection of crossover and mutation points. They avoid disrupting effective components, but at the same time they restrict the GP search. According to our experiments, restricting the search reduces the efficiency of GP. At the beginning when the population is just initialized, few composite operators contain effective components. At this time, GP should search extensively to generate effective components and assemble them together. It is harmful to apply smart operators at the early stage of GP search since they just restrict the search. Only after some time when the effective components are gathered in composite operators, smart operators should be applied to identify the effective components to avoid disrupting them and keep them in a public library for later reuse. So, in this chapter, smart operators are not used in the first 20 generations. In the last 50 generations, smart operators are applied with higher and higher probability as the population goodness becomes larger and larger. Here, the number 20 is experimentally determined, since in our experiments, it is observed that the population fitness (the average fitness of all the composite operators in the population) increases significantly in the first 20 generations, which means after 20 generations, some effective components are generated and assembled together.

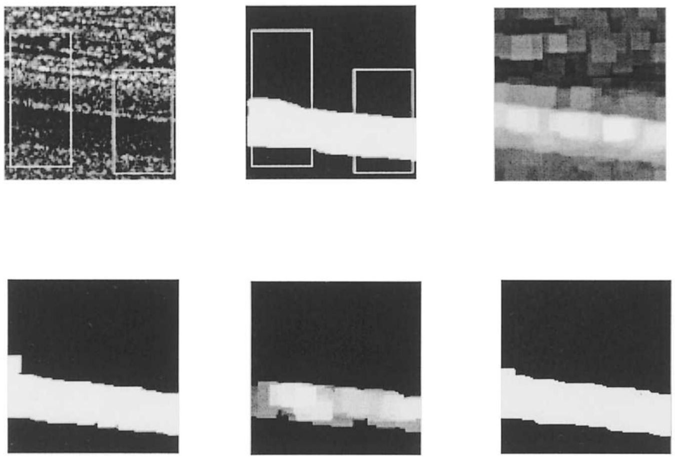
3.3.3 Steady-state and generational genetic programming

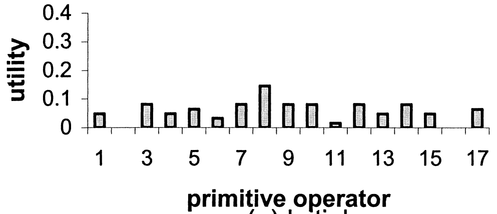
As in chapter 2, steady-state genetic programming and generational genetic programming are used to synthesize composite operators. The major difference is that in generational GP, the offspring from crossover are kept aside and do not participate in the crossover operation on the current population. The current population is not changed during crossover. But in steady-state GP, the offspring from crossover are evaluated and replace the worst individuals in the population immediately, and they participate in the crossover operations on the current population. In smart GP, a MDL-based fitness function is used, smart GP crossover and smart mutation are invoked with probability determined by ε-greedy policy and a public library is set up to store the effective components of composite operators. Similarly, we adopt an elitism replacement method to keep the best composite operator from generation to generation. At the end of each generation, GP checks each composite operator and replaces it with the subtree whose root node has the highest goodness value among all the nodes of the composite operator. This is helpful to further control the size of composite operators and avoid overfitting. Figure 3.1 and Figure 3.2 show the pseudo code for modified steady-state and generational genetic programming algorithms, respectively.

Modified Steady-state Genetic Programming Algorithm:

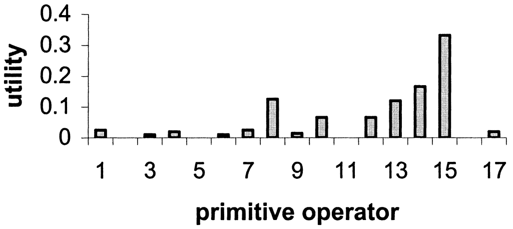
```
0.  randomly generate population P of size M and evaluate composite operators in P.
1.        for gen = 1 to N do   // N is the number of generation
2.    keep the best composite operator in P.
3.        repeat
4.        select 2 composite operators from P based on  their fitness values for
           crossover.
5.        select 2 composite operators with the lowest fitness values in P for
           replacement.
6.         if gen < 20 then
7.           perform random crossover and let the 2 offspring replace the 2 composite
              operators selected for replacement.
         else
8.           perform smart or random crossover and let the 2 offspring replace the 2
              composite operators selected for replacement.
         endif
9.        execute the 2 offspring and evaluate their fitness values.
10.   until crossover rate is met.
11.       if gen < 20 then
12.       perform random mutation on each composite operator with probability of
           mutation rate.
      else
13.       perform smart or random mutation on each composite operator with
           probability of mutation rate.
      endif
14.   execute and evaluate mutated composite operators.
      // after crossover and mutation, a new populationP' is generated.
15.   perform elitism mechanism. let the best composite operator of P replace the
        worst composite operator in P' and  let P = P'.
16.   update the value of ε according to equation (3.2).
17.   store good components of composite operators in the public library.
18.     if the goodness of the best composite operator in P is above goodness threshold
         value, then
19.        stop.
      endif
20.   check each composite operator in P and use its best component to replace it.
    endfor  // loop
```

Figure 3.1. Modified Steady-state genetic programming.

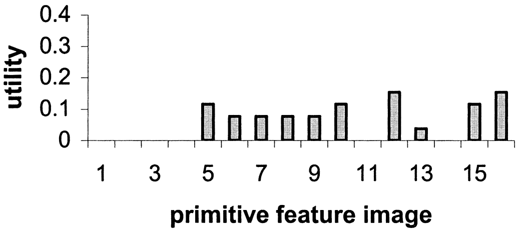




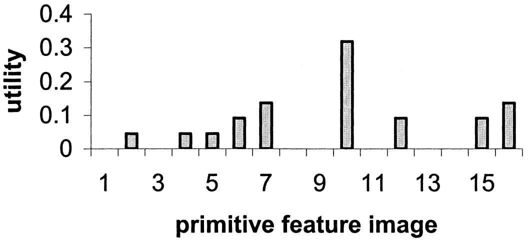
(a) Initial



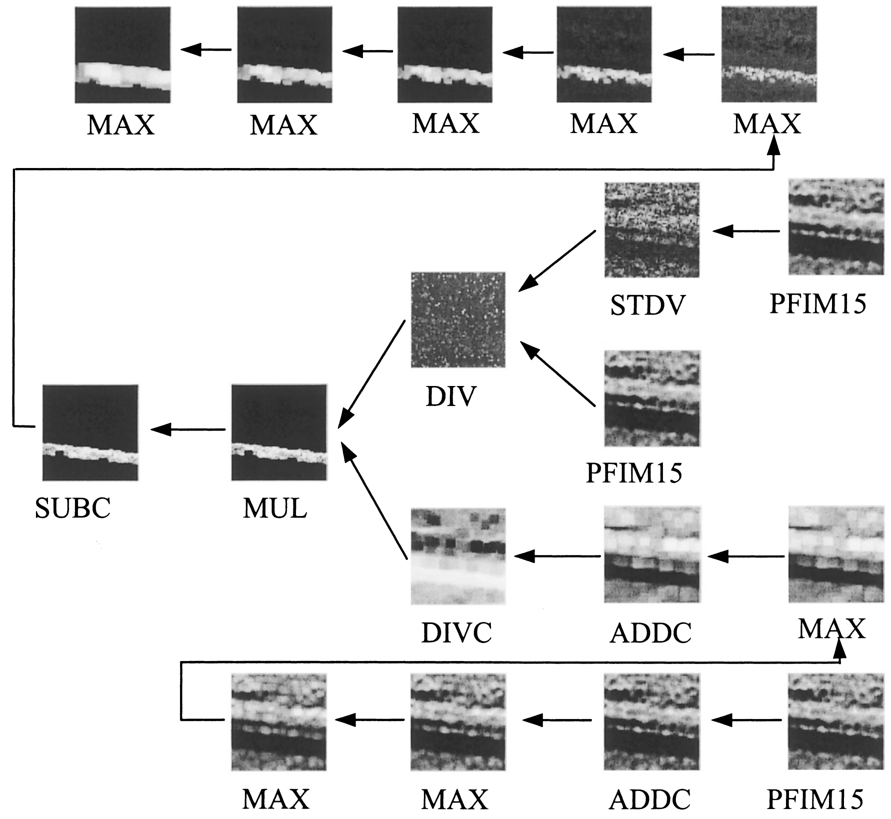
(b) final

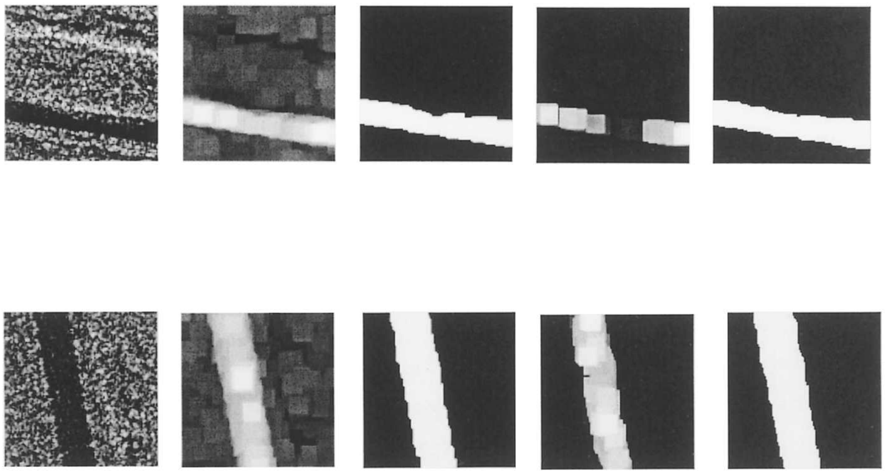


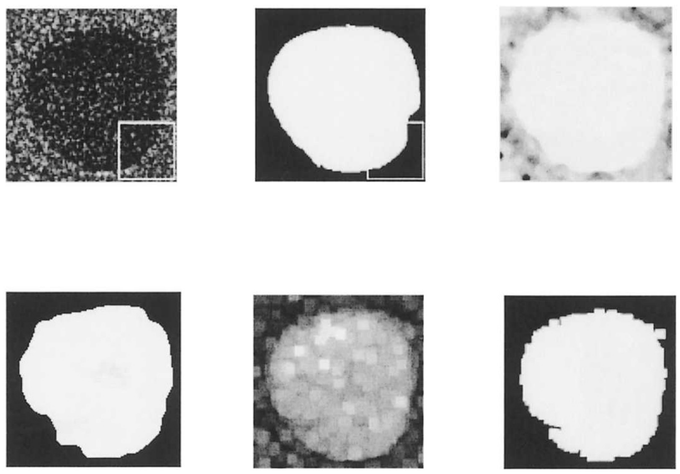
(c) Initial

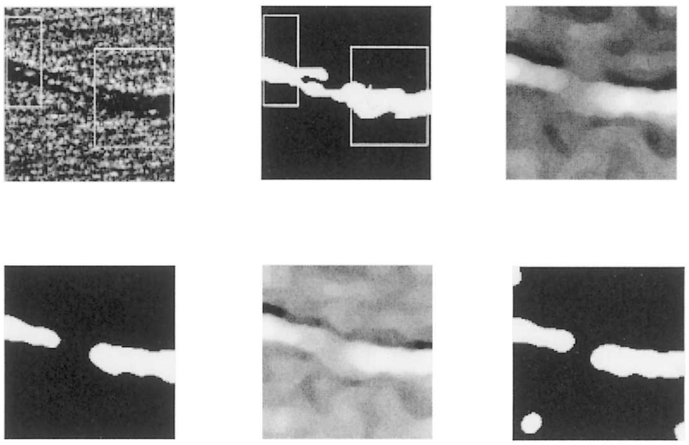


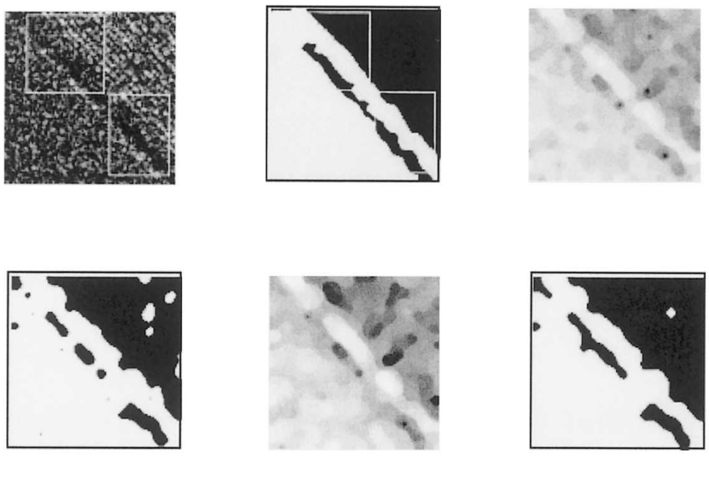
(d) final

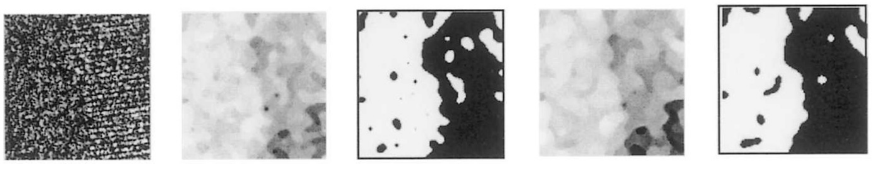

MAX
MAX
MAX
MAX
MAX
STDV
PFIM15
DIV
PFIM15
SUBC
MUL
DIVC
ADDC
MAX
MAX
MAX
ADDC
PFIM15

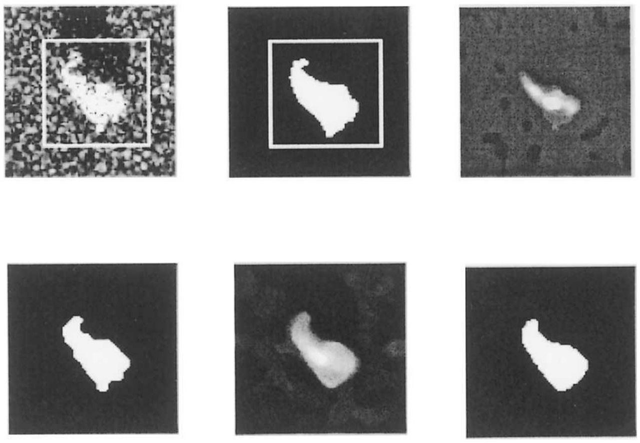












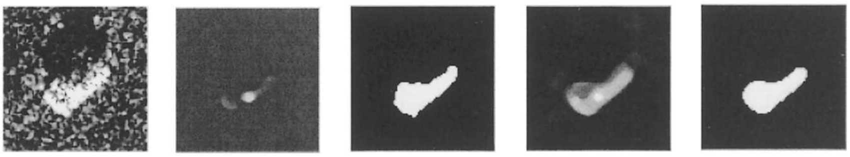
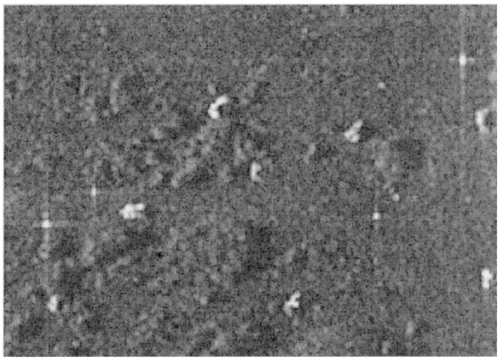
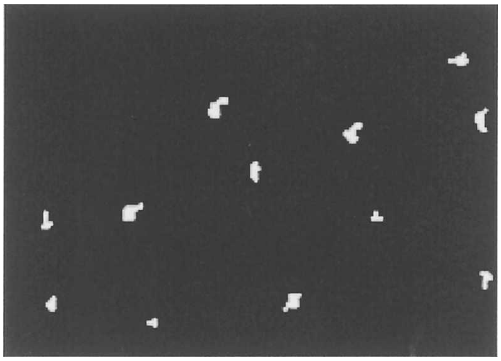


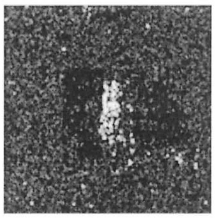
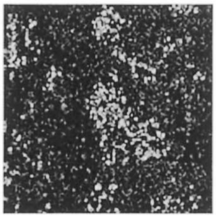
Table 3.3. The average size and performance of the best composite operators from normal and smart GPs.

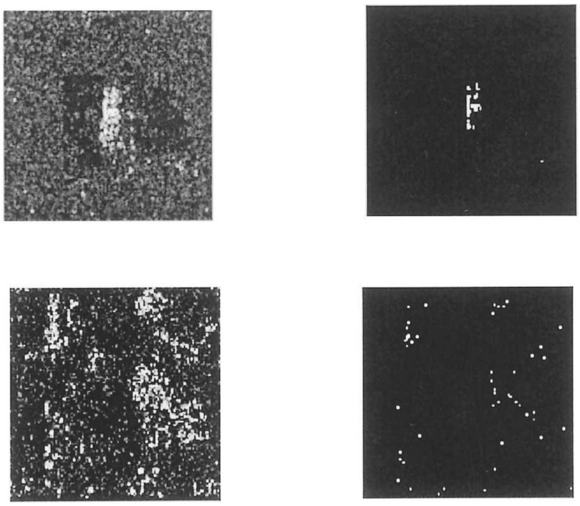
		Normal GP				
		Road	Lake	River	Field	Tank
Size	mean	29.4	28.4	27.6	20.2	24.6
	stdv	1.07	1.74	4.43	8.93	6.17
Training	mean	0.789	0.891	0.583	0.794	0.829
	stdv	0.080	0.128	0.112	0.101	0.035
Testing	mean	0.620, 0.797	0.913	0.754	0.675	0.766
	stdv	0.274, 0.151	0.161	0.129	0.124	0.042
		Smart GP				
		Road	Lake	River	Field	Tank
Size	mean	24.6	11.8	16.8	14.9	5.7
	stdv	4.58	5.65	7.19	9.98	1.9
Training	mean	0.860	0.916	0.650	0.839	0.849
	stdv	0.038	0.021	0.049	0.039	0.025
Testing	mean	0.831, 0.914	0.972	0.836	0.784	0.821
	stdv	0.115, 0.025	0.009	0.023	0.033	0.012

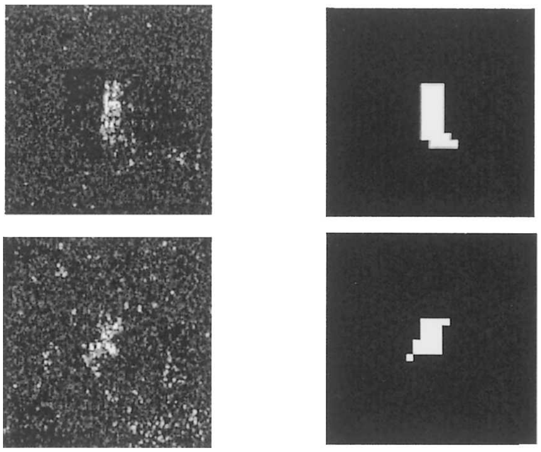
Table 3.4. Average training time of Normal GP and Smart GP.

		Road	Lake	River	Field	Tank
Normal GP	mean	6915	2577	7951	3606	2686
	stdv	5348	1213	8006	2679	2163
Smart GP	mean	10249	770	11035	5251	649
	stdv	8893	724	10310	5506	589





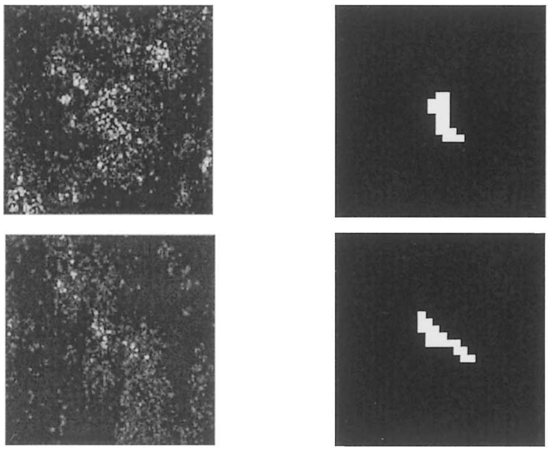


Table 4.2. Experimental results with 500 training target and clutter chips (MDL, equation (4.2); ε = 0.0015).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	17	35	5	0100001001 1000100000	0.002	1	1	0.006	0	7
2	13	31	5	0100001001 0000001001	0.002	1	1	0.006	0	7
3	19	38	5	0100001001 0000011000	0.002	1	1	0.006	0	7
4	20	38	5	0100001001 0000011000	0.002	1	1	0.006	0	7
5	10	28	5	0100001001 0010100000	0.002	1	1	0.006	0	7
6	26	44	5	0100001001 1100000000	0.002	1	1	0.003	0	3
7	25	43	5	0100001001 0000010100	0.002	1	1	0.007	0	8
8	9	27	6	0000001011 0000011010	0.002	1	1	0.007	0	8
9	8	26	5	0100001001 0000011000	0.002	1	1	0.006	0	7
10	17	35	5	0001001001 0011000000	0.002	1	1	0.004	0	4
Ave.	16.4	34.5	5.1		0.002	1	1	0.0057	0	6.5

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.3. Experimental results with 700 training target and clutter chips (MDL, equation (4.2); $\varepsilon = 0.0015$).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	8	8	9	0101101001 1010001001	0.0014	1	1	0.006	0	4
2	9	9	10	1101001001 1010101010	0.0014	1	1	0.001	0	1
3	7	7	7	0000001011 0100101010	0.0014	1	1	0.012	0	8
4	2	2	10	1101001001 0110011010	0.0014	1	1	0.001	0	1
5	5	5	8	0100001001 0011111000	0.0014	1	1	0.007	0	5
6	2	2	7	1000011011 0100001000	0.0014	1	1	0.012	0	8
7	5	5	10	1101001001 0110101100	0.0014	1	1	0.001	0	1
8	3	3	10	1100101011 0101010001	0.0014	1	1	0.003	0	2
9	4	4	11	1101011001 1010111000	0.0014	1	1	0.001	0	1
10	4	4	10	1101001001 0011111000	0.0014	1	1	0.001	0	1
Ave.	4.9	4.9	9.2		0.0014	1	1	0.0045	0	3.2

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.4. Experimental results with 700 training target and clutter chips (MDL, equation (4.2); ε = 0.0011).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	10	28	6	0001001001 1000001010	0.0014	1	1	0.004	0	3
2	19	37	5	0100001001 0000001010	0.0014	1	1	0.012	0	8
3	17	35	5	0100001001 0010100000	0.0014	1	1	0.01	0	7
4	16	34	6	0001011001 0010001000	0.0014	1	1	0.006	0	4
5	16	34	5	0100001001 0000011000	0.0014	1	1	0.01	0	7
6	19	37	5	0100001001 0010100000	0.0014	1	1	0.01	0	7
7	10	28	5	0100001001 0000010100	0.0014	1	1	0.01	0	7
8	15	33	5	0100001001 0000011000	0.0014	1	1	0.01	0	7
9	10	28	6	0100011001 1000010000	0.0014	1	1	0.007	0	5
10	23	41	5	0100001001 0000001001	0.0014	1	1	0.01	0	7
Ave.	15.5	33.5	5.3		0.0014	1	1	0.0089	0	6.1

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

two features, leading to the unsatisfactory training and testing error rates. In order to balance the number of features selected and the error rate, parameter γ must be finely tuned. This is not an easy task and it usually takes a lot of time. The MDL-based fitness function is based on a sound theory and it balances these two terms very well. Only a few features are selected while the training and testing error rates are kept low.

Table 4.5. Experimental results with 500 training target and clutter chips (penalty function, equation (4.4); $\varepsilon = 0.0015$).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	4	22	13	0111111011 1100111000	0.002	1	1	0.004	0	4
2	11	11	10	1011011011 0001100100	0.001	1	0	0.005	0	5
3	2	20	9	0101101001 1011000100	0.002	1	1	0.005	0	5
4	4	22	11	1010011011 0101011100	0.002	1	1	0.004	0	4
5	3	21	10	1110001011 1010010100	0.002	1	1	0.003	0	3
6	10	10	9	0011011011 0000110100	0.001	1	0	0.005	0	5
7	8	26	10	1101101001 0011010010	0.002	1	1	0.001	0	1
8	2	20	11	1110101011 0001001110	0.002	1	1	0.003	0	3
9	3	21	10	0110011011 1101100000	0.002	1	1	0.005	0	5
10	3	21	9	1110011011 0000110000	0.002	1	1	0.008	0	9
Ave.	5	19.4	10.2		0.0018	1	1	0.0043	0	4.4

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.6. Experimental results with 500 training target and clutter chips (penalty and # of features, equation (4.5); $\gamma = 0.1$; $\varepsilon = 0.0015$).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	18	36	2	0000001001 0000000000	0.005	2	3	0.024	0	26
2	12	30	2	0000001000 0000001000	0.007	1	6	0.007	0	8
3	17	35	2	0000001001 0000000000	0.005	2	3	0.024	0	26
4	20	38	2	0000001001 0000000000	0.005	2	3	0.024	0	26
5	16	34	2	0000001001 0000000000	0.005	2	3	0.024	0	26
6	11	29	2	0000001001 0000000000	0.005	2	3	0.024	0	26
7	15	33	2	0000001001 0000000000	0.005	2	3	0.024	0	26
8	17	35	2	0000001001 0000000000	0.005	2	3	0.024	0	26
9	14	32	2	0000001001 0000000000	0.005	2	3	0.024	0	26
10	12	30	2	0000001000 0000001000	0.007	1	6	0.007	0	8
Ave.	15.2	33.2	2		0.0054	1.8	3.6	0.0206	0	22.4

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.7. Experimental results with 500 training target and clutter chips (penalty and # of features, equation (4.5); $\gamma = 0.3$; $\varepsilon = 0.0015$).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	23	41	1	0000001000 0000000000	0.01	1	9	0.036	0	39
2	20	38	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	11	29	2	1000001000 0000000000	0.005	1	4	0.033	0	36
4	8	26	3	0000000010 0010010000	0.008	4	4	0.005	0	5
5	30	48	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	14	32	1	0000001000 0000000000	0.01	1	9	0.036	0	39
7	25	43	1	0000001000 0000000000	0.01	1	9	0.036	0	39
8	20	38	1	0000001000 0000000000	0.01	1	9	0.036	0	39
9	22	40	1	0000001000 0000000000	0.01	1	9	0.036	0	39
10	27	45	1	0000001000 0000000000	0.01	1	9	0.036	0	39
Ave.	20	38	1.3		0.0093	1.3	8	0.0326	0	35.3

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.8. Experimental results with 500 training target and clutter chips (penalty and # of features, equation (4.5); $\gamma = 0.5$; $\varepsilon = 0.0015$).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	17	35	1	0000001000 0000000000	0.01	1	9	0.036	0	39
2	29	41	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	22	40	1	0000001000 0000000000	0.01	1	9	0.036	0	39
4	15	33	1	0000001000 0000000000	0.01	1	9	0.036	0	39
5	32	50	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
7	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
8	23	41	1	0000001000 0000000000	0.01	1	9	0.036	0	39
9	9	27	2	0000000010 0000001000	0.012	5	7	0.011	0	12
10	23	41	1	0000001000 0000000000	0.01	1	9	0.036	0	39
Ave.	19.2	37.2	1.1		0.01	1.5	8.8	0.0335	0	36.3

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.9. Experimental results with 500 training target and clutter chips (error rate and # of features, equation (4.6); $\gamma = 0.1$; $\varepsilon = 0.0015$).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	21	39	1	0000001000 0000000000	0.01	1	9	0.036	0	39
2	16	34	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	14	32	2	0000100010 0000000000	0.01	7	3	0.006	0	7
4	25	43	1	0000001000 0000000000	0.01	1	9	0.036	0	39
5	13	31	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	17	35	1	0000001000 0000000000	0.01	1	9	0.036	0	39
7	17	35	2	0000100010 0000000000	0.01	7	3	0.006	0	7
8	33	51	1	0000001000 0000000000	0.01	1	9	0.036	0	39
9	22	40	1	0000001000 0000000000	0.01	1	9	0.036	0	39
10	12	30	1	0000001000 0000000000	0.01	1	9	0.036	0	39
Ave.	19	37	1.2		0.01	2.2	7.8	0.03	0	32.6

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.10. Experimental results with 500 training target and clutter chips (penalty and # of features, equation (4.6); $\gamma = 0.3$; $\varepsilon = 0.0015$)

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
2	27	45	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	17	35	1	0000001000 0000000000	0.01	1	9	0.036	0	39
4	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
5	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	11	29	1	0000001000 0000000000	0.019	7	12	0.028	0	31
7	20	38	1	0000000010 0000000000	0.01	1	9	0.036	0	39
8	30	48	1	0000000010 0000000000	0.019	7	12	0.028	0	31
9	7	25	1	0000000010 0000000000	0.019	7	12	0.028	0	31
10	12	30	1	0000001000 0000000000	0.01	1	9	0.036	0	39
Ave.	15.7	33.7	1		0.013	2.8	9.9	0.0336	0	36.3

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.11. Experimental results with 500 training target and clutter chips (penalty and # of features, equation (4.6); $\gamma = 0.5$; $\varepsilon = 0.0015$).

Run	B_g	T_g	F_n	Features selected	Training error rate	Number of errors		Testing error rate	Number of errors	
						T	C		T	C
1	25	43	1	0000000010 0000000000	0.019	7	12	0.028	0	31
2	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
3	8	26	1	0000000010 0000000000	0.019	7	12	0.028	0	31
4	11	29	1	0000001000 0000000000	0.01	1	9	0.036	0	39
5	8	26	1	0000001000 0000000000	0.01	1	9	0.036	0	39
6	15	33	1	0000001000 0000000000	0.01	1	9	0.036	0	39
7	9	27	1	0000001000 0000000000	0.01	1	9	0.036	0	39
8	12	30	1	0000001000 0000000000	0.01	1	9	0.036	0	39
9	29	47	1	0000001000 0000000000	0.01	1	9	0.036	0	39
10	24	42	1	0000001000 0000000000	0.01	1	9	0.036	0	39
Ave.	15.2	33.2	1		0.013	2.2	9.4	0.0344	0	37.4

B_g: best generation. T_g: total generation. F_n: number of features selected.
T: target. C: clutter.

Table 4.12. Experimental results using only one feature for discrimination (target chips = 500, clutter chips = 500).

Feature	Error rate	Number of errors		Feature	Error rate	Number of errors	
		Target	Clutter			Target	Clutter
1	0.119	17	102	11	0.118	18	100
2	0.099	16	83	12	0.111	6	105
3	0.056	7	49	13	0.126	9	117
4	0.057	17	40	14	0.131	7	124
5	0.068	13	55	15	0.09	5	85
6	0.354	0	354	16	0.069	3	66
7	0.01	1	9	17	0.075	3	72
8	0.5	480	20	18	0.209	0	209
9	0.019	7	12	19	0.2	2	198
10	0.073	15	58	20	0.244	0	244

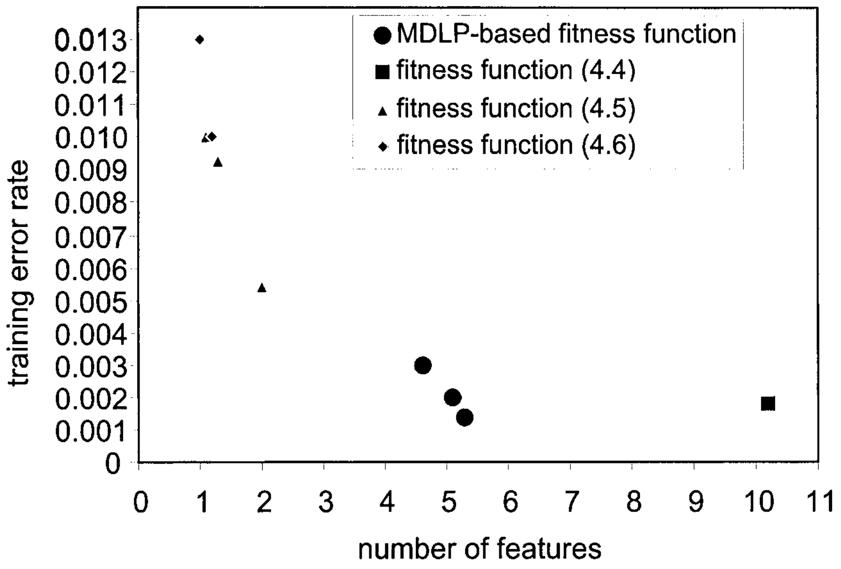
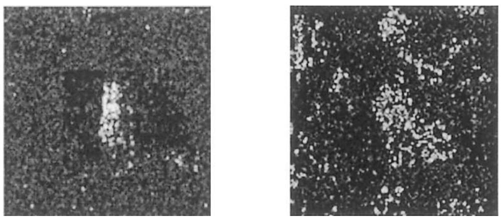
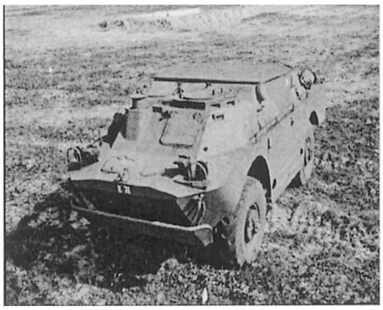
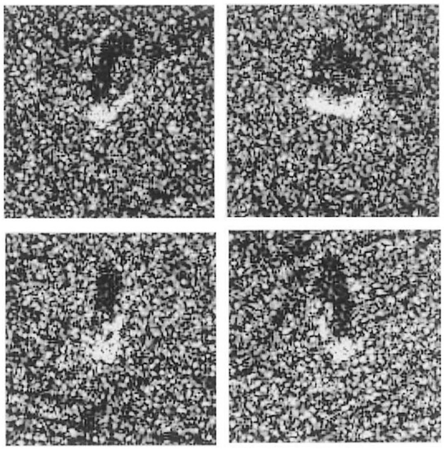


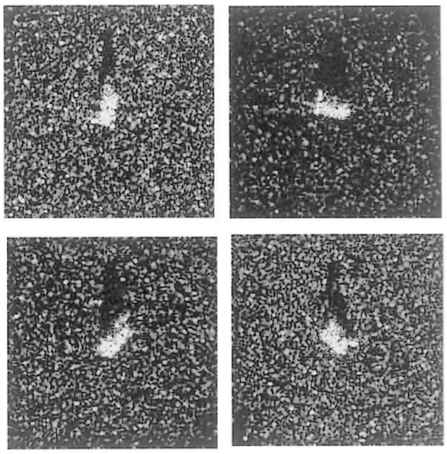
Figure 4.9. Average performance of various fitness functions

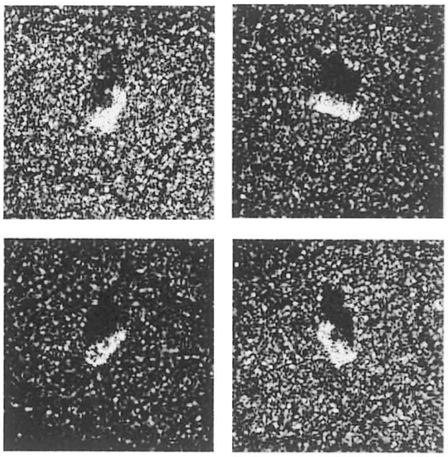
Table 4.13. The number of times each feature is selected in MDL Experiments 1, 2 and 4.

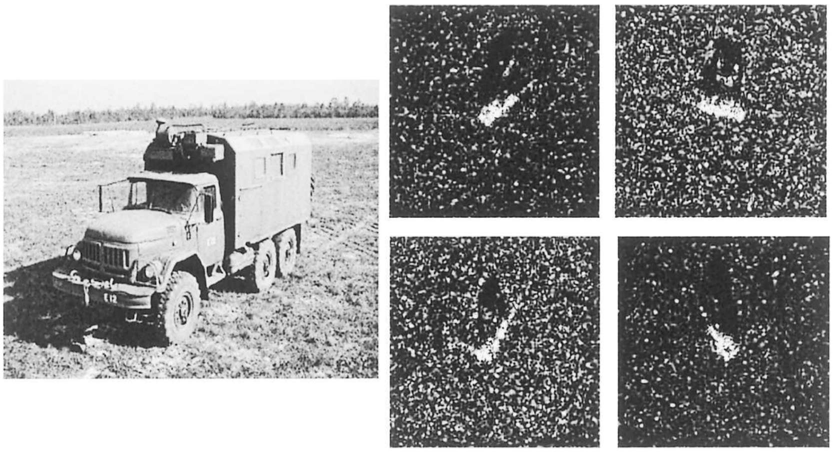
Features	Exp 1	Exp 2	Exp 4	Total
1	0	0	0	0
2*	8	8	8	24
3	2	0	0	2
4	1	1	2	4
5	1	0	0	1
6	1	0	2	3
7*	10	10	10	30
8	0	0	0	0
9	2	1	0	3
10*	10	10	10	30
11	1	2	1	4
12	2	1	0	3
13	3	2	3	8
14	1	1	0	2
15	4	2	2	8
16	0	4	4	8
17	0	5	6	11
18	0	1	1	2
19	0	1	2	3
20	0	1	1	2









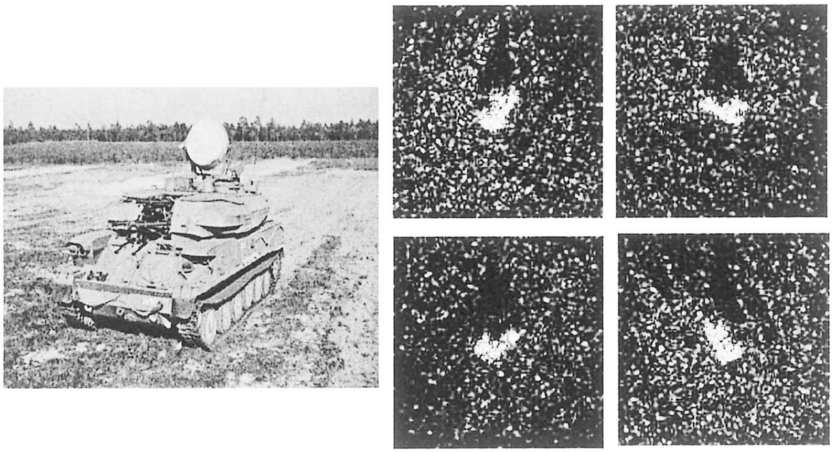


Table 5.5. Recognition rates of 20 primitive features (3 objects).

Feature Number	Primitive Feature	Recognition Rate	Feature Number	Primitive Feature	Recognition Rate
1	Standard deviation	0.376	11	Horizontal projection	0.414
2	Fractal dimension	0.662	12	Vertical projection	0.545
3	Weight-rank fill ratio	0.607	13	Major diagonal projection	0.460
4	Blob mass	0.717	14	Minor diagonal projection	0.455
5	Blob diameter	0.643	15	Minimum distance	0.505
6	Blob inertia	0.495	16	Maximum distance	0.417
7	Maximum CFAR	0.588	17	Mean distance	0.376
8	Mean CFAR	0.726	18	Moment μ_{20}	0.421
9	Percent bright CFAR	0.607	19	Moment μ_{02}	0.443
10	Count	0.633	20	Moment μ_{22}	0.512

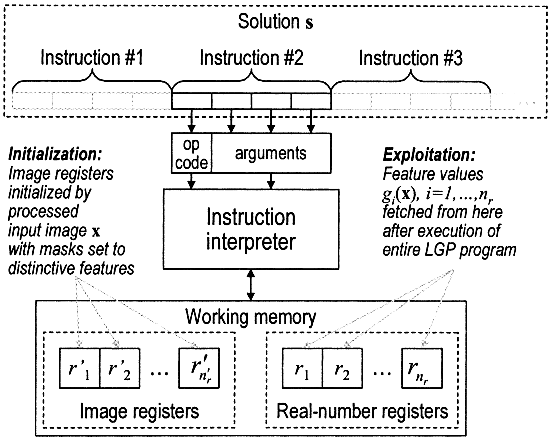
Table 5.6. Performance of composite and primitive features on 3-object discrimination.

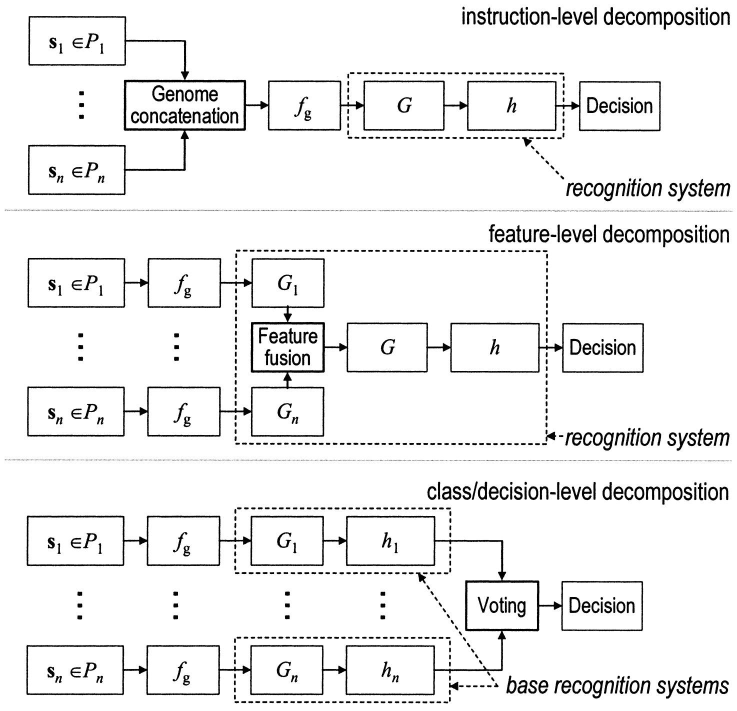
Runs	Recognition Rate			
	10f		20f	
	Training	Testing	Training	Testing
3-dimensional composite feature vector	0.880	0.843	0.969	0.943
5-dimensional composite feature vector	0.921	0.857	0.990	0.961
8-dimensional composite feature vector	0.962	0.870	0.999	0.970
Primitive feature vector	0.863	0.812	0.995	0.962

Table 5.10. Average recognition performance of multi-layer neural networks trained by backpropagation algorithms (5 objects).

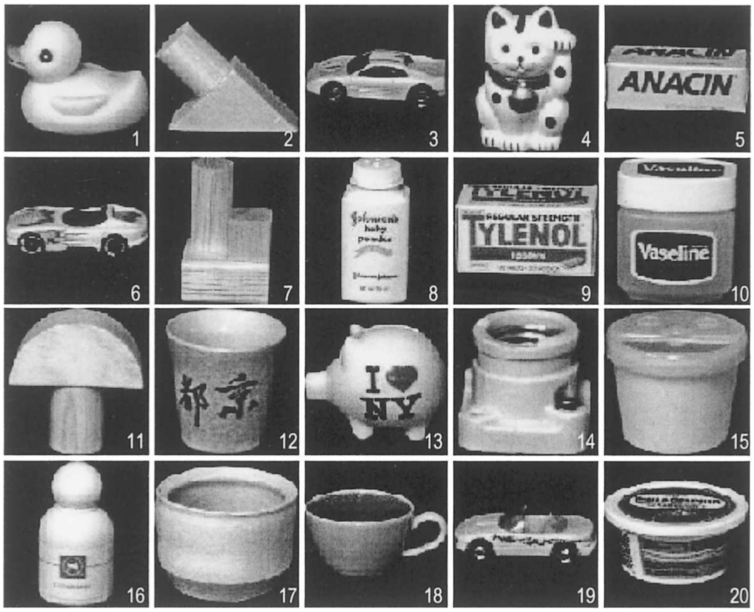
Number of hidden nodes	Recognition Rate (Backpropagation)			
	10f		20f	
	Training	Testing	Training	Testing
5	0.274	0.267	0.346	0.340
8	0.292	0.290	0.330	0.325
Number of hidden nodes	Recognition Rate (Backpropagation - Stochastic)			
	10f		20f	
	Training	Testing	Training	Testing
5	0.302	0.300	0.370	0.366
8	0.296	0.296	0.366	0.366
Number of hidden nodes	Recognition Rate (Backpropagation – Stochastic with momentum)			
	10f		20f	
	Training	Testing	Training	Testing
5	0.319	0.304	0.376	0.367
8	0.331	0.328	0.369	0.368

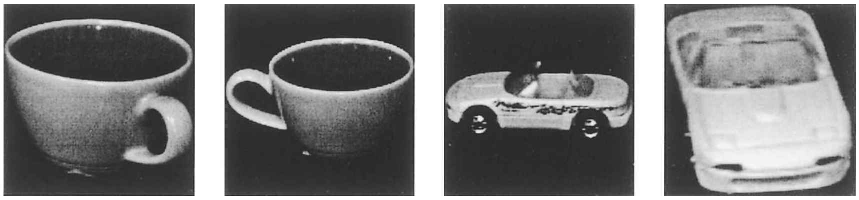
10f – means only 10 common primitive features are used in feature synthesis
20f – means all the 20 primitive features are used in feature synthesis


Solution **s**
Instruction #1
Instruction #2
Instruction #3
op code
arguments
Initialization: *Image registers initialized by processed input image* **x** *with masks set to distinctive features*
Instruction interpreter
Exploitation: *Feature values* $g_i(\mathbf{x})$, $i=1,\dots,n_r$ *fetched from here after execution of entire LGP program*
Working memory
r'_1 r'_2 … $r'_{n'_r}$
Image registers
r_1 r_2 … r_{n_r}
Real-number registers


instruction-level decomposition
$\mathbf{s}_1 \in P_1$
$\mathbf{s}_n \in P_n$
Genome concatenation
f_g
G
h
Decision
recognition system
feature-level decomposition
$\mathbf{s}_1 \in P_1$
$\mathbf{s}_n \in P_n$
f_g
G_1
G_n
Feature fusion
G
h
Decision
recognition system
class/decision-level decomposition
$\mathbf{s}_1 \in P_1$
$\mathbf{s}_n \in P_n$
f_g
G_1
h_1
G_n
h_n
Voting
Decision
base recognition systems

Category	Operations
Image → Image	
Convolution filters (for mask sizes 3x3 and 5x5)	Prewitt, Sobel, Laplacian, Gaussian, Highpass, Lowpass, Sharpening
Other filters	Median filter, Min filter, Thresholding, Normalized cross-correlation
Image transforms	2-D Fast Fourier Transform
Morphological operations	Erosion, Dilatation, Opening, Closing
Image arithmetic (pixelwise)	Absolute difference, Addition, Subtraction, Multiplication
Image logic operations (pixelwise)	And, Or, Xor
$Image^k \rightarrow \Re^l$	
Image norms	Dot product, L1 (city-block), L2 (Euclidean)
Feature extraction operations	Spatial 2D moments (up to 3rd order), Central 2D moments (up to 3rd order), Normalized central 2D moments (up to 3rd order), Mass center, Location of the brightest pixel, Location of the darkest pixel. Number of non-zero pixels, Sum/Average/Standard deviation of pixel intensities
$\Re^k \rightarrow \Re^l$	
Scalar arithmetic	+, -, *, % (protected division)
Scalar functions	Max, Min, Abs, Sgn, If (conditional expression) Sin, Cos, Tan, Exp, Log
Other	
Mask-related operations	Set rectangular mask, Set mask upper left corner, Set mask lower right corner, Shift mask in specific direction, Get mask height, Get mask width, Get mask mid X, Get mask mid Y

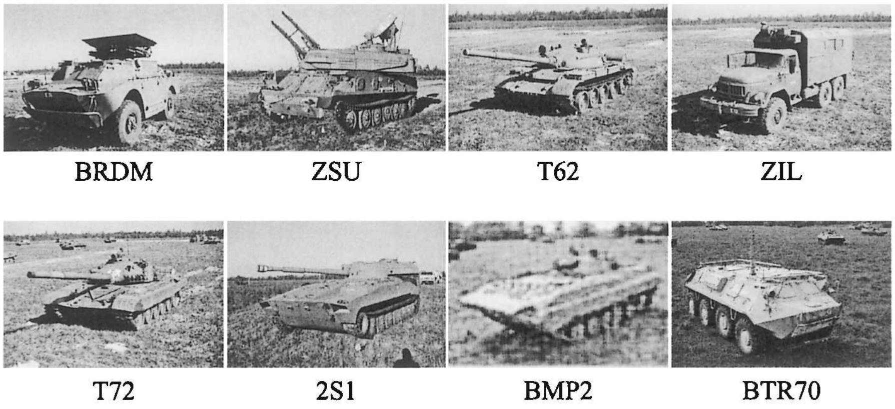

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

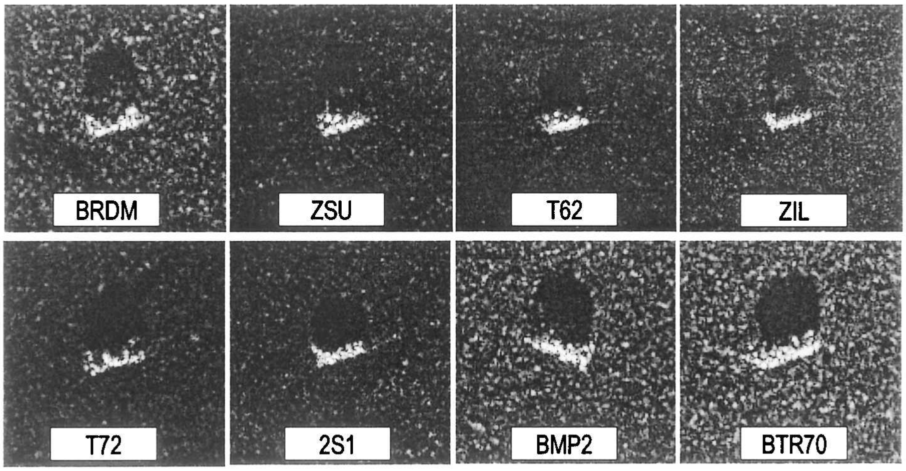


Parameter	Setting
Single population size $\|P_i\|$	500
# of populations n_P	2
# of registers (n_r=n'_r)	2
Mutation operator	bit flip, probability 0.1
Crossover operator	one point, probability 1.0
Selection operator	tournament selection, pool size: 5
Time limit for evolutionary search	4000 seconds

ANACIN
TYLENOL

ANACIN
TYLENOL


BRDM
ZSU
T62
ZIL
T72
2S1
BMP2
BTR70


BRDM
ZSU
T62
ZIL
T72
2S1
BMP2
BTR70

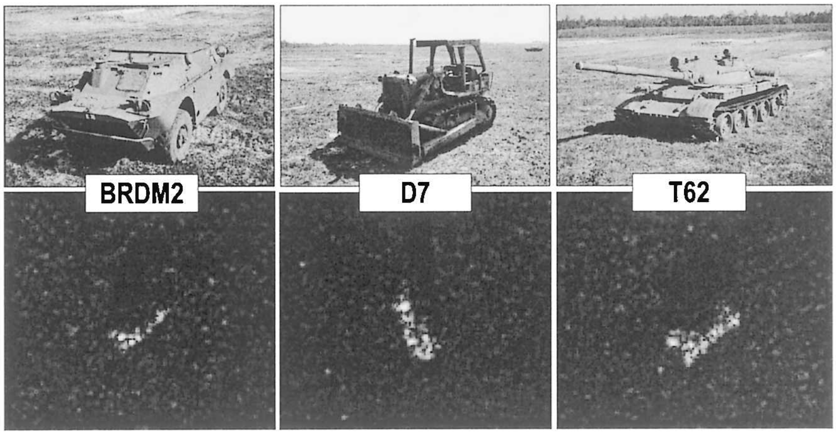

BRDM2
D7
T62

Table 7.4. Performance of recognition systems evolved by means of cooperation at genome level.

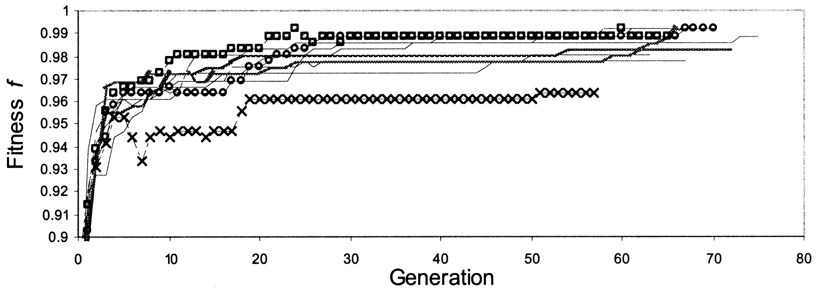
Method	Parameter setting			Recognition ratio			
				1000 seconds		*2000 seconds*	
	n_P	l	P_i	*Training set*	*Test set*	*Training set*	*Test set*
EFP	1	72	300	0.806	0.747	0.843	0.801
CFP	3	24	100	0.915	0.867	0.933	0.890
EFP	1	72	900	0.839	0.795	0.881	0.830
CFP	3	24	300	0.927	0.874	0.940	0.883

Table 7.5. Test set confusion matrix for selected EFP recognition system.

Actual class	Predicted class			
	BRDM2	D7	T62	None
BRDM2	**97**	3	22	2
D7	0	**115**	9	0
T62	1	0	**66**	0

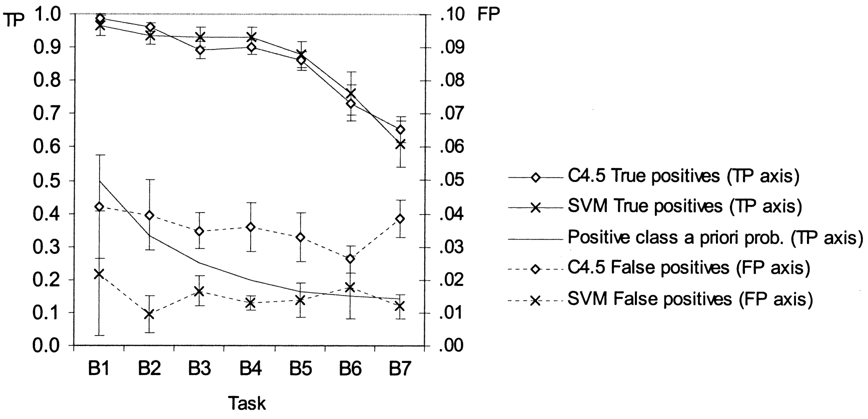
Table 7.6. Test set confusion matrix for selected CFP recognition system.

Actual class	Predicted class			
	BRDM2	D7	T62	None
BRDM2	**118**	1	4	1
D7	5	**114**	3	2
T62	5	1	**61**	0



1
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.9
Fitness *f*
0
10
20
30
40
50
60
70
80
Generation

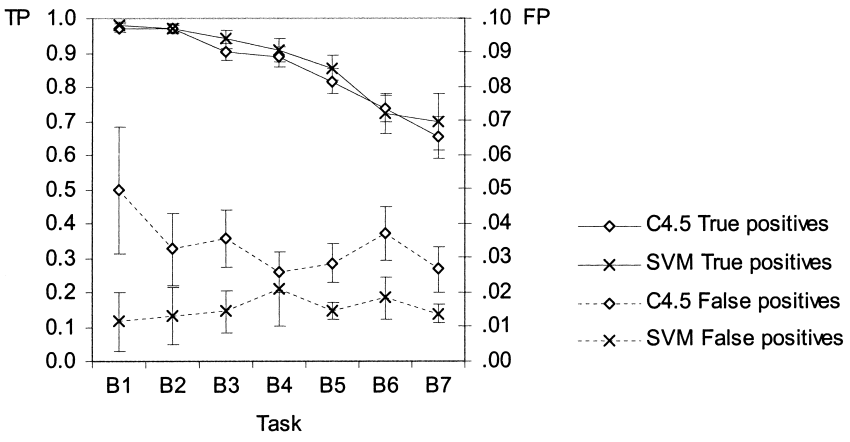
Task	C4.5		SVM	
	TP	FP	TP	FP
B1	.987 ± .007	.042 ± .016	.966 ± .032	.022 ± .019
B2	.960 ± .013	.040 ± .011	.935 ± .027	.010 ± .005
B3	.892 ± .025	.035 ± .005	.929 ± .030	.017 ± .005
B4	.901 ± .023	.036 ± .007	.929 ± .032	.013 ± .002
B5	.860 ± .030	.033 ± .007	.880 ± .039	.014 ± .005
B6	.733 ± .055	.026 ± .004	.762 ± .063	.018 ± .009
B7	.654 ± .041	.039 ± .006	.610 ± .069	.012 ± .004

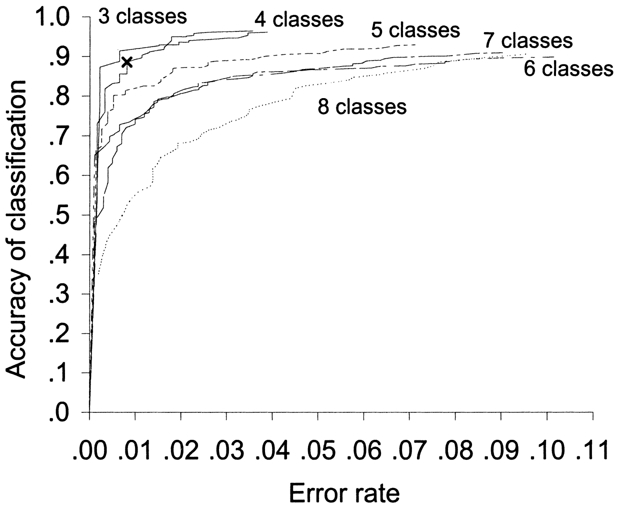


TP
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
.10 FP
.09
.08
.07
.06
.05
.04
.03
.02
.01
.00
B1
B2
B3
B4
B5
B6
B7
Task
C4.5 True positives (TP axis)
SVM True positives (TP axis)
Positive class a priori prob. (TP axis)
C4.5 False positives (FP axis)
SVM False positives (FP axis)

Task	C4.5		SVM	
	TP	FP	TP	FP
B1	.973 ± .012	.050 ± .019	.981 ± .010	.012 ± .008
B2	.969 ± .011	.033 ± .010	.972 ± .012	.013 ± .008
B3	.904 ± .025	.036 ± .008	.940 ± .026	.014 ± .006
B4	.888 ± .031	.026 ± .006	.908 ± .035	.021 ± .011
B5	.816 ± .036	.028 ± .006	.856 ± .038	.015 ± .003
B6	.736 ± .038	.037 ± .008	.723 ± .058	.018 ± .006
B7	.652 ± .062	.027 ± .007	.698 ± .082	.014 ± .003

Task	# of learned populations	
	Mean	Maximum
B1	5.6	7
B2	5.1	7
B3	5.8	6
B4	5.3	6
B5	5.0	6
B6	4.9	6
B7	4.4	5


TP
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
.10 FP
.09
.08
.07
.06
.05
.04
.03
.02
.01
.00
B1
B2
B3
B4
B5
B6
B7
Task
C4.5 True positives
SVM True positives
C4.5 False positives
SVM False positives



Object (class)	Predicted class		
	BMP2#C21	T72#132	No decision
BMP2#9563,9566	**295**	18	78
T72#812,s7	4	**330**	52

Object	Predicted class				
	BMP2#C21	T72#132	BTR#C71	ZSU#d08	No decision
BMP2#9563,9566	**293**	27	27	1	43
T72#812,s7	12	**323**	1	9	41

Task	C4.5		SVM	
	TP	FP	TP	FP
B1	1.000	.000	1.000	.000
B2	1.000	.000	.981	.000
B3	.955	.006	.981	.002
B4	.955	.006	.974	.000
B5	.955	.004	.961	.001
B6	.792	.002	.896	.004
B7	.721	.005	.708	.001

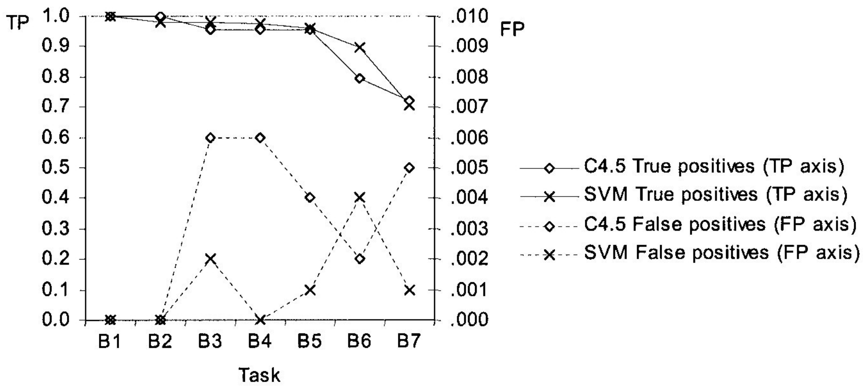


Figure 7.15. True positive and false positive ratios for binary recognition tasks (testing set, compound recognition systems).

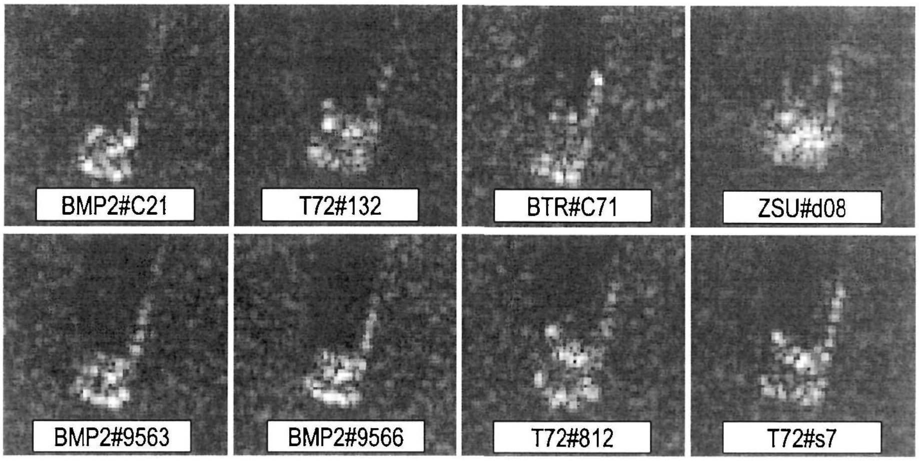
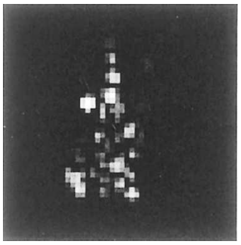
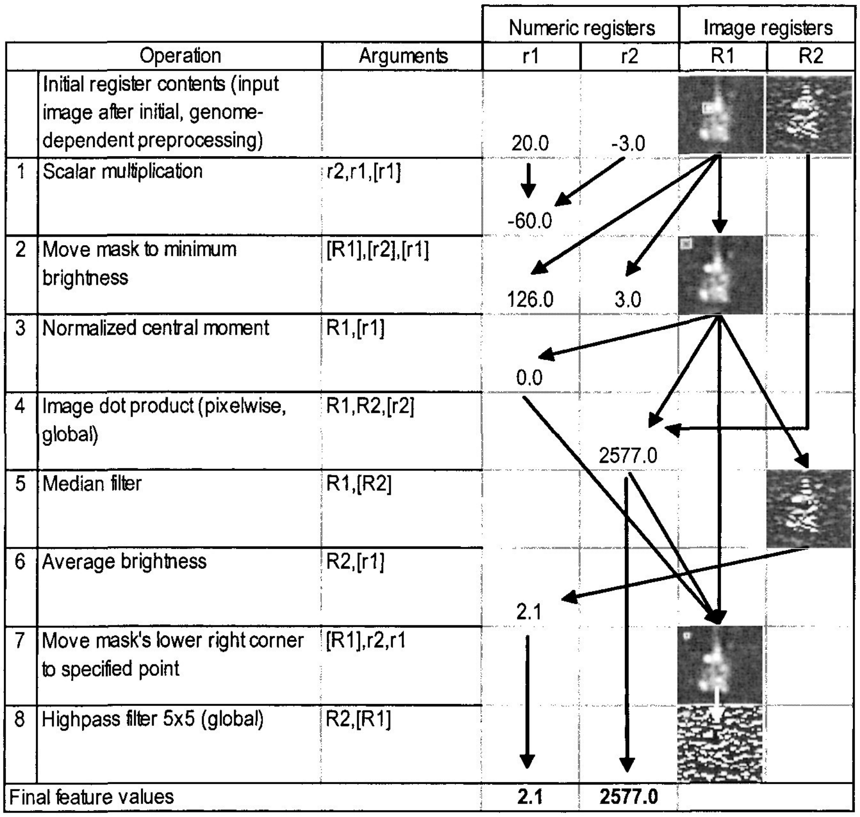


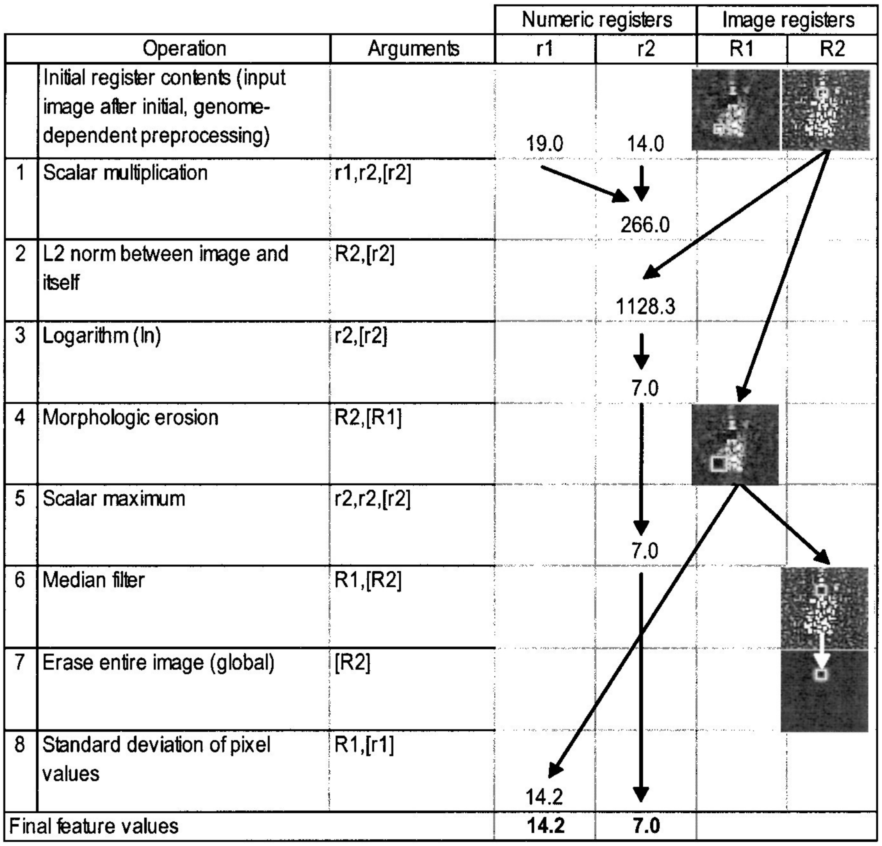
Figure 7.16. Representative images of objects used in experiments concerning object variants (all pictures taken at 191° aspect/azimuth, cropped to central 64×64 pixels, and magnified to show details).





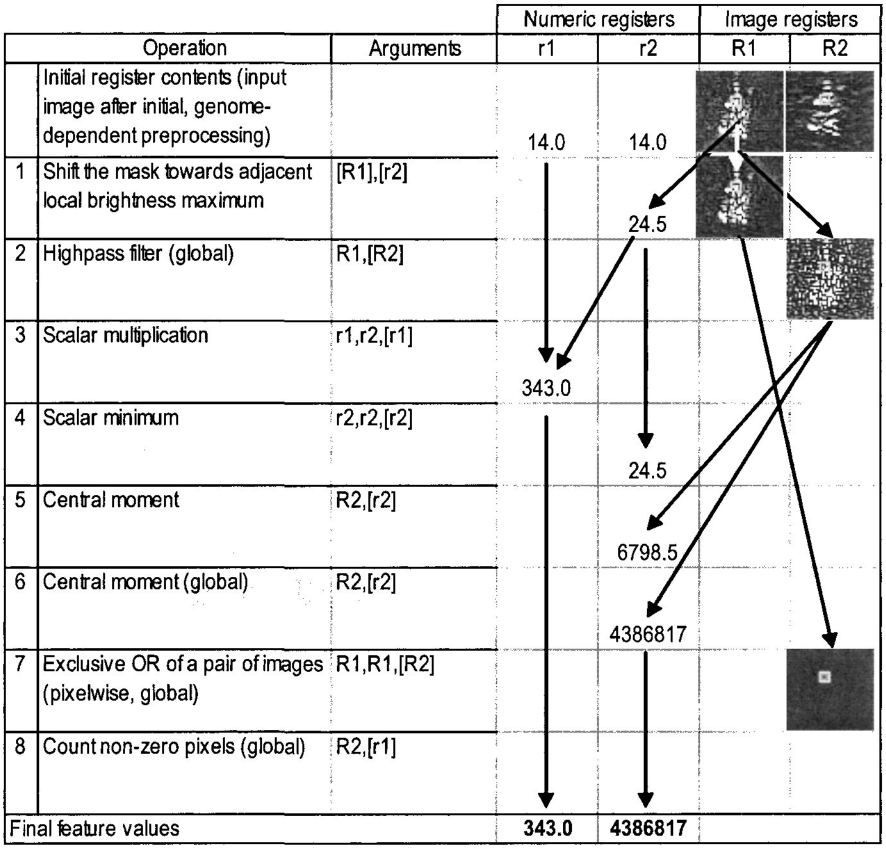
			Numeric registers		Image registers	
Operation		Arguments	r1	r2	R1	R2
	Initial register contents (input image after initial, genome-dependent preprocessing)		20.0	-3.0		
1	Scalar multiplication	r2,r1,[r1]	-60.0			
2	Move mask to minimum brightness	[R1],[r2],[r1]	126.0	3.0		
3	Normalized central moment	R1,[r1]	0.0			
4	Image dot product (pixelwise, global)	R1,R2,[r2]		2577.0		
5	Median filter	R1,[R2]				
6	Average brightness	R2,[r1]	2.1			
7	Move mask's lower right corner to specified point	[R1],r2,r1				
8	Highpass filter 5x5 (global)	R2,[R1]				
Final feature values			**2.1**	**2577.0**		

Figure 7.18. Processing carried out by one of the evolved solutions (individual 1 of 4; see text for details).



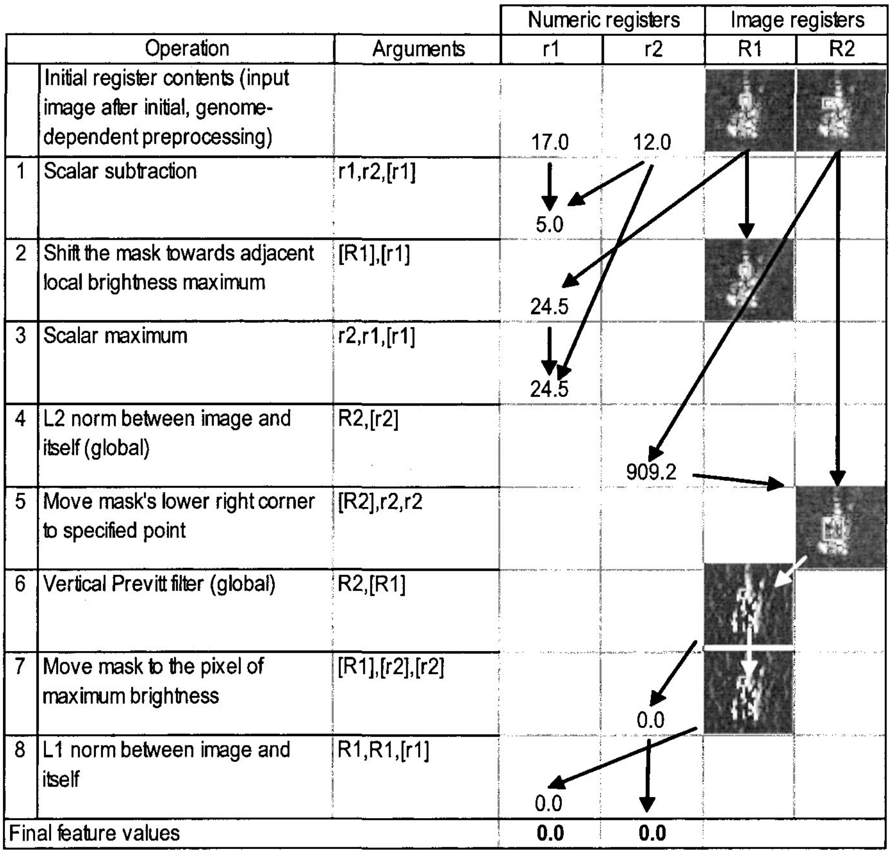
			Numeric registers		Image registers	
	Operation	Arguments	r1	r2	R1	R2
	Initial register contents (input image after initial, genome-dependent preprocessing)		19.0	14.0		
1	Scalar multiplication	r1,r2,[r2]		266.0		
2	L2 norm between image and itself	R2,[r2]		1128.3		
3	Logarithm (ln)	r2,[r2]		7.0		
4	Morphologic erosion	R2,[R1]				
5	Scalar maximum	r2,r2,[r2]		7.0		
6	Median filter	R1,[R2]				
7	Erase entire image (global)	[R2]				
8	Standard deviation of pixel values	R1,[r1]	14.2			
Final feature values			**14.2**	**7.0**		

Figure 7.19. Processing carried out by one of the evolved solutions (individual 2 of 4; see text for details).



			Numeric registers		Image registers	
	Operation	Arguments	r1	r2	R1	R2
	Initial register contents (input image after initial, genome-dependent preprocessing)		14.0	14.0		
1	Shift the mask towards adjacent local brightness maximum	[R1],[r2]		24.5		
2	Highpass filter (global)	R1,[R2]				
3	Scalar multiplication	r1,r2,[r1]	343.0			
4	Scalar minimum	r2,r2,[r2]		24.5		
5	Central moment	R2,[r2]		6798.5		
6	Central moment (global)	R2,[r2]		4386817		
7	Exclusive OR of a pair of images (pixelwise, global)	R1,R1,[R2]				
8	Count non-zero pixels (global)	R2,[r1]				
Final feature values			**343.0**	**4386817**		

Figure 7.20. Processing carried out by one of the evolved solutions (individual 3 of 4; see text for details).



			Numeric registers		Image registers	
Operation		Arguments	r1	r2	R1	R2
	Initial register contents (input image after initial, genome-dependent preprocessing)		17.0	12.0		
1	Scalar subtraction	r1,r2,[r1]	5.0			
2	Shift the mask towards adjacent local brightness maximum	[R1],[r1]	24.5			
3	Scalar maximum	r2,r1,[r1]	24.5			
4	L2 norm between image and itself (global)	R2,[r2]		909.2		
5	Move mask's lower right corner to specified point	[R2],r2,r2				
6	Vertical Previtt filter (global)	R2,[R1]				
7	Move mask to the pixel of maximum brightness	[R1],[r2],[r2]		0.0		
8	L1 norm between image and itself	R1,R1,[r1]	0.0			
Final feature values			**0.0**	**0.0**		

Figure 7.21. Processing carried out by one of the evolved solutions (individual 4 of 4; see text for details).