Towards the Automatic Construction of Machine Vision Systems using Genetic Programming

Olly Oechsle

A thesis submitted for the degree of

Doctor of Philosophy

at the

School of Computer Science and Electronic Engineering

University of Essex

September 2009

A Brief Glossary

Abbreviation	Description	Page
DRS	Dynamic Range Selection. A means of representing classifiers.	69
DRS2	Refers to the author's extension of DRS, which render it free of parameters.	78
DRS2C	Refers to the author's extension of DRS2, which renders it capable of returning confidence estimates.	78
ECJ	A popular Java-based toolkit for evolutionary computation.	98
ES	Evolution Strategies. An early genetic learning technique.	10
FULL	A tree builder, used for generating programs in Genetic Programming.	15
GA	Genetic Algorithm(s).	12
Genotype	The encoding of a solution, which may differ from its eventual behaviour (phenotype).	12
GLCM	Grey-Level Co-occurence Matrix. Summarises spatial relationships between different pixels, used to calculate texture descriptors.	51
GP	Genetic Programming.	12
GROW	A tree builder, used for generating programs in Genetic Programming.	15
ICS	Intelligent Classification System. The author's classification framework.	90
JASMINE	An end-user component of the author's software. Permits users to generate vision systems.	175
LDA	Linear Discriminant Analysis. A long-established statistical technique that may be used for binary classification.	123
Phenotype	The behaviour of a program or individual, which is not always entirely determined by its genotype.	12
PS	Partial Solutions. Another of the author's classification frameworks.	87
SXGP	The author's own Genetic Programming toolkit.	98

Abstract

Computer vision is a topic that has interested researchers and commercial organisations alike for some time: it provides both a considerable intellectual challenge, and a wide variety of useful applications, some of which are now becoming ubiquitous. In this thesis the author has studied means by which vision software may be constructed automatically using Genetic Programming (GP) – a technique that learns how to write programs during a simulation of Darwinian evolution. This research addresses the question of how one might create more "complete" vision systems using GP, beyond simply proving the applicability of evolutionary learning to particular image processing tasks. Research into making Genetic Programming more suitable for deployment as a generic learning tool is presented, evaluated and assessed, and novel means by which multi-stage vision systems can be constructed from evolved components is described. The author does not claim to have invented significant new paradigms in either GP or mainstream computer vision – rather the focus is on bridging the gap between task-specific applications and a generic learning framework. An architecture for creating such applications is presented, along with software that permits non-expert users to create vision systems rapidly of a complexity first equal to, then beyond that so-far published by GP researchers.

Acknowledgements

I would like to express my gratitude to a number of people before commencing this document. I am particularly grateful to my fiancée (now wife!) Rong Gao for her patient support and kindness, especially in the last stressful months of preparing a thesis document. It appears that most doctoral study inevitably deteriorates into mathematics to some degree; I also thank her for her assistance in those areas.

I am grateful for the support of my supervisor, Dr Adrian Clark, throughout the course of my study. I trust his assistance to have steered me along the right track, while allowing me to take the research my own direction and to indulge ideas that have interested me since childhood. I also gratefully acknowledge the assistance from the EPSRC for making the last three years financially tolerable!

I dedicate this thesis to my parents, Maggie and Gunther, who invested so much into my education – a sacrifice that, until now, I've never properly thanked them for.

Contents

List of Figures

List of Tables

Datasets

Although the author's software develops and uses bespoke datasets for a number of vision problems, the following benchmark datasets, mainly from the UCI machine learning repository, are used throughout this thesis for the purposes of comparison and validation. These are summarised here for the reader's convenience. Unless otherwise stated, results on these datasets are averages following the execution of 50 evolutionary learning runs.

Dataset	Samples	Purpose
BUPA	345	To predict whether or not a patient will suffer liver disorders based on the results of 5 blood tests and the number of alcohol units drunk per day. Validated using 10-fold cross validation.
Glass	214	To identify 6 different types of glass, using 10 attributes discovered following chemical analysis. Validated using 10-fold cross validation.
Heart	270	To determine the presence or absence of heart disease using 13 different attributes, including the patient's age/sex and the results of various diagnostic tests. Validated using 10-fold cross validation.

Dataset	Samples	Purpose
Ionosphere	351	To identify radar patterns indicating the presence or absence of “some type of structure” in the ionosphere. Uses 34 attributes collected from an array of 16 high-frequency antennas. Validated using 10-fold cross validation.
Iris	150	R.A. Fisher’s classic dataset concerning the recognition of three species of Iris using 4 attributes: measurements of the width and length of flowers’ petals and sepals. Validated using 10-fold cross validation.
PenDigits	7494 + 3498	To recognise ten different characters [0–9], hand-written by 44 authors, based on a feature vector of 16 attributes corresponding to different arc lengths. Validation using a separate test set.
Pima	768	To identify whether a female patient shows signs of diabetes or not, based on 8 different attributes. Validated using 10-fold cross validation.
SatImage	4435 + 2000	36 features from a multi-spectral imager are used to identify 6 different types of terrain on the ground below. Validated using a separate test set.
Thyroid	3772 + 3428	To determine whether or not a patient has an overactive thyroid, using 21 attributes, 16 of which are binary. Validated using a separate test set.
Vehicle	846	To classify a given silhouette as one of four types of vehicle, using a set of 18 image features extracted from the silhouette. Validated using 9 fold cross validation, for which the sets are pre-defined.
WDBC	569	To diagnose the presence of breast cancer using 30 features derived from a fine needle aspirate (FNA) of a breast mass describing cell nucleii. Validated using 10-fold cross validation.

Overview

The following is a high-level overview of the research described in this thesis.

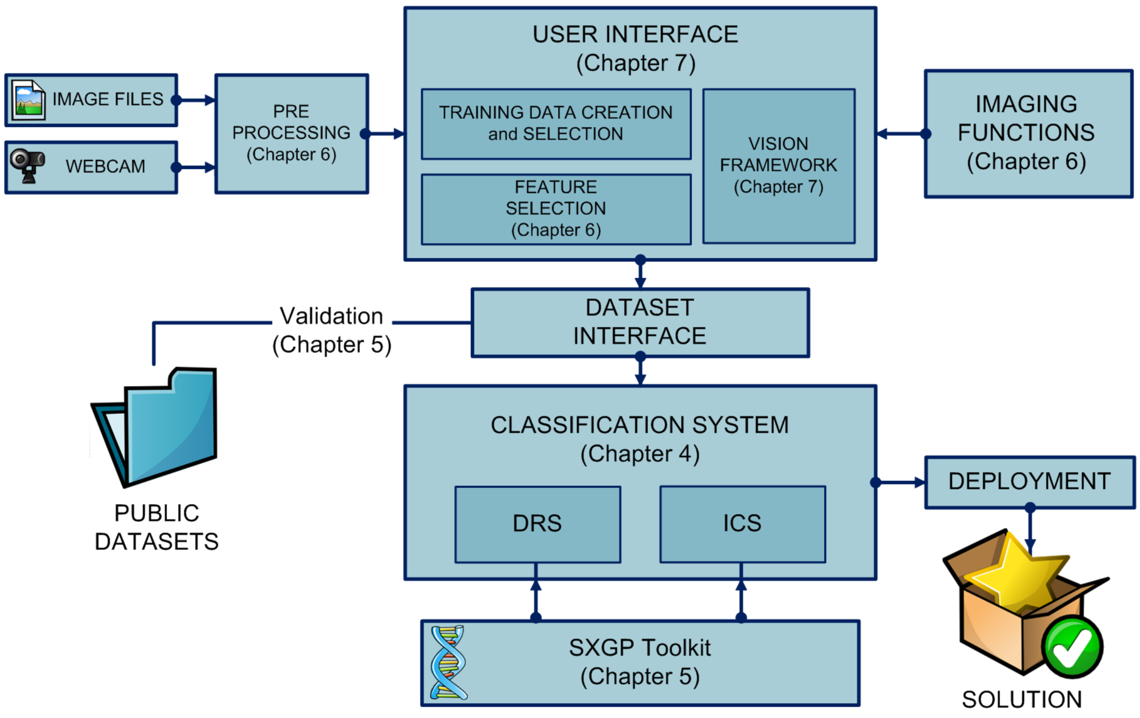


IMAGE FILES
WEBCAM
PRE PROCESSING (Chapter 6)
USER INTERFACE (Chapter 7)
TRAINING DATA CREATION and SELECTION
FEATURE SELECTION (Chapter 6)
VISION FRAMEWORK (Chapter 7)
IMAGING FUNCTIONS (Chapter 6)
DATASET INTERFACE
Validation (Chapter 5)
PUBLIC DATASETS
CLASSIFICATION SYSTEM (Chapter 4)
DRS
ICS
DEPLOYMENT
SXGP Toolkit (Chapter 5)
SOLUTION

Chapter 1

Introduction

Some 540 million years before you started reading this thesis, something interesting happened. Although living organisms have existed on Earth for over 3.5 billion years of our planet's 4.5 billion year history, the most dramatic evolution of new species occurred during a period of "just" 60–70 million years – an event we now refer to as the *Cambrian Explosion*. Although the fossil record isn't fully complete, the data suggest that life on Earth transformed during this short period from mainly single-celled species to a world filled with the major groups of complex, multicellular life still in evidence today.

The Cambrian Explosion was so rapid and pronounced that even Charles Darwin expressed concern that it undermined his theory of evolution by natural selection. However, later research [1] suggested that evolution is not, in fact, a process of gradual improvement, rather it consists of large periods of stasis punctuated by rapid bursts of change. To this day, and in the absence of other hypotheses, evolution is human-kind's best explanation for how we all came to be here – and how you come to be reading this thesis.

As it happens, it is from this period that the earliest fossil remains of animal eyes have been found. While basic photo receptors are sufficient for an individual to distinguish light from dark or day from night, an eye is something rather special. Our vision systems permit us to explore and navigate through the world with a remarkable degree of understanding.

Although nobody knows for certain what triggered the Cambrian Explosion, some zoologists [2] believe it was the advent of sight that sustained it. One can only imagine the advantage of a predator with some form of sight would have in a world filled with blind prey. The survival of a species would hinge on its ability to adopt better defences or to develop such new senses in turn; an evolutionary arms race commenced. As it happens, the *Burgess Shale*, a substantial deposit of fossils dating from the Cambrian Explosion, shows evidence of the adaptations evolved by predators and prey alike. Nowadays 96%

of the world's species employ some form of vision system in order to perceive the world as illuminated by visible light[1].

Most species have what one might describe as a *vision system*, including both eye sensors and a degree of processing in the brain[2]. Indeed some species, notably birds of prey, have significantly better sensors than humans. However, humans are distinguished by the significant portion or our already generously proportioned brains dedicated to vision *processing*. In addition to being able to perceive our surroundings instantly in astonishing clarity, we can recognise almost anything, including things we haven't seen before. We can understand how some tools and machines work simply by looking at them, all seemingly without effort.

Because these "tasks" can be performed without much noticeable exertion, it is quite easy to be complacent about the quality and scope of our vision system. It is perhaps only when we try to implement vision systems in silicon that we come to realise just what an astonishing feat of engineering is to be found inside our heads. From our implementations of imperfect sensor technology, incomplete vision algorithms and underpowered hardware, one gains an appreciation of how much has been accomplished within and behind our eyes. Nonetheless, while the tasks that computer vision researchers have set themselves are sometimes ambitious, some great leaps forward have been made in recent years.

1.1 The Rise of Machine Vision

During the last decade or so, some applications of computer vision, such as the automatic recognition of car number-plates or in-camera face detection, have become ubiquitous. This is thanks both to the significant increases in computing power available in increasingly small devices and the quality of work undertaken by academic researchers and commercial organisations in labs around the world. Whether one likes or dislikes applications such as the congestion charge cameras in London, they only exist because machines can now do the bulk of the vision processing automatically and accurately.

Now that machines can demonstrably perform a limited subset of useful vision tasks, human ambition demands ever more applications for them to work on. Some are com-

[1]Of course, visible light is only a small band in the electromagnetic spectrum. Interestingly, the wavelengths within the "visible" spectrum are one of only two bands of light that can travel through water. Our continuing exclusive perception of just this small band is a throwback to the environment in which vision systems first appeared – in creatures who lived underwater.

[2]Although some species of jellyfish do not – their eyes are connected directly to motor receptors.

pletely new, such as the interactive window displays now showing on the streets of Tokyo or the foyers of Harrods department store. As machine vision breaks out from military and research circles into the mainstream, new topics of research arise, along with a desire to produce these systems more easily.

How might one go about creating vision systems more easily? Certainly many of the established, excellent techniques for extracting information from images are not easy to understand, being rooted firmly within the confines of mathematics, statistics and pattern analysis. They are often built around certain assumptions about how the world appears, which may not hold true, and they generally require the effort of an expert to select, implement and tune them. In short, it is time-consuming and difficult to produce vision software.

1.2 Machine Learning

As a concept, it would be nice to be able to develop strategies in a way that required less effort. Ideally we would have the computer do this on its own – a process broadly defined as machine learning. After all, computers can work on certain problems much more quickly than we can and, unlike humans, computers don't have a concept of impatience nor do they demand hourly wages! Modern computing power is certainly making heavy tasks such as image processing increasingly practical and so the automatic learning of computer vision is becoming more and more feasible. Indeed, some high quality vision software has already been developed using machine learning techniques, for instance the work by Viola and Jones on face detection [3].

One family of machine learning techniques takes inspiration from Charles Darwin's theory of evolution directly. The aim is that by simulating the process of natural selection, computers may be able to replicate the explosion of progress and adaptation that occurred on Earth 540 million years previously. So-called evolutionary computation was largely confined to parameter optimisation tasks until computing power permitted the invention of Genetic Programming (GP) [4], an adaption of the classic genetic algorithm [5] which evolves *programs* instead of sets of numerical coefficients. Over the last twenty or so years, researchers have been exploring the ways that programs themselves can be evolved to solve all manner of problems. As we'll see later, some equal or outperform human invention.

The value of machine learning is well understood, and accordingly many learning algorithms have been proposed, most from outside the realm of evolutionary computation.

Some, such as neural networks, can already claim a variety of established applications within computer vision, notably character recognition. Most learning techniques, however, are constrained either by assumptions or applicability. Support Vector Machines for instance, are an excellent classification technique but can't develop algorithms, *per se*. Neural networks are somewhat limited in their ability to develop truly non-parametric solutions to problems. Decision trees can make decisions based on features but cannot develop the features themselves. Genetic Programming is distinct from each: as an abstract learning system with no particular underlying assumptions about input data, it is uniquely placed to develop a whole range of different software, including all the examples above. As such, it seems appropriate that GP should be used for the automatic construction of vision systems.

1.3 Evolved Vision Software

As we'll see later, Genetic Programming has already been put to work on a wide variety of problems in computer vision. Although a number of excellent solutions have been evolved by GP researchers, they are generally developed to prove that GP can solve problems within a particular domain. However, if one is to devise and implement a separate GP learning system for every situation, then one of the main advantages of machine learning – reducing the human effort – is lost.

In contrast, this thesis explores the viability of the GP as a domain-independent generator of visual routines; more specifically, the development of more complete, working vision systems – something that isn't generally tackled by GP researchers. The author would define a working vision system as one that takes images as input and returns high-level output without requiring human intervention in the intermediate stages. The idea is to develop a framework that can learn to develop vision systems applicable to a range of different situations, with the proviso that all the task-specific information can be encapsulated within training data. The key is that the framework should not require manual tuning, adaptation or modification for individual tasks.

It is the aim of this thesis to explore the extent to which certain vision tasks can be learned automatically (or with minimal human input) by this generic architecture, and to see whether the GP-learned vision software is any good from a practical point of view. The feasibility, accuracy and capabilities of vision components developed using Genetic Programming will be described and analysed.

1.4 Challenges

Some would argue that the use of Genetic Programming is a self-imposed challenge in its own right. As we shall later see, Genetic Programming does have some significant drawbacks not least its reputation, which is not entirely undeserved, for being slow and inefficient.

A theme that runs through the entire course of this thesis concerns means by which Genetic Programming can be made more efficient at solving problems involving vision. We shall also see how many of the parameters usually required to define a GP run can be dispensed with, turning the GP system into a more flexible learning tool.

Despite the drawbacks, one cannot deny the power of evolution as evidenced by nature. The potential of Genetic Programming to discover novel solutions by means we hadn't previously considered motivates this author to produce vision systems whose task-specific components are all evolved by GP, something that has not been done before.

1.5 Contributions

The author has made a number of novel contributions during the course of undertaking this research. These are outlined briefly below:

- The author has developed a complete Genetic Programming toolkit specifically for the purpose of solving vision problems, which compares favourably to a current, widely-used toolkit, ECJ [6]. p.98

- The author has developed a classification scheme using Genetic Programming termed "Intelligent Classification System", which produces demonstrably better results on public datasets than the majority of other GP classification systems. The classification system makes pseudo-intelligent decisions, can run in parallel, and can solve problems with many classes. p.90

- The author has proposed an extension to an existing GP classification representation to include confidence estimation. This makes it more suitable to certain problems and can improve classification accuracy when used by multiple classifier systems. p.82

- The author has described how many of the parameters in GP systems can be optimised or discarded such as to render GP into a suitably generic learning engine for vision problems p.97

- The author has developed a generic feature-detection approach which is capable of evolving components for a variety of domains, to a similar standard to those previously published by GP researchers. p.127

- The author has developed a multi-stage vision framework which permits the creation of vision systems applicable to a variety of different tasks. The architecture is controlled by graphical user interface which permits end-users to define tasks and create training data visually. After the training data has been created, the interface initiates an intelligent learning process to evolve complete vision systems automatically. p.167

- The author has demonstrated the vision systems on a number of different vision tasks of a complexity not previously attempted by GP researchers. p.181

- The author's work on generating vision systems using Genetic Programming has been published in [7]. By the time you come to read this thesis, two further papers will have been submitted. The first is a more thorough description to the author's vision system generator. The second concerns the author's approaches to classification by GP. p.127

1.6 Thesis Structure

This thesis concerns two areas of study: computer vision and evolutionary computation. Accordingly it begins with two literature surveys. In the following chapter the author introduces evolutionary computation in general and Genetic Programming in particular, and covers a number of techniques and algorithms employed by researchers in the GP community. In Chapter 3 the wider field of computer vision and pattern recognition is introduced. A number of techniques commonly employed in computer vision are described and assessed; solutions developed by human intuition and machine learning are compared and contrasted.

The learning component of the author's system is based largely around the classification or labelling of feature vectors. Before discussing what features may be appropriate to describe the content of an image, Chapter 4 considers how to use Genetic Programming in order to generate classifiers in a general sense.

The field of information theory and pattern recognition is well established. There has already been mention of various machine learning techniques. In Chapter 5 the results of the author's GP classifiers are compared to other GP toolkits and other machine learning systems, in order to justify more fully the decision to use GP on a generic system. The chapter also describes in more detail the implementation of the author's GP toolkit.

Later chapters are concerned with the development of vision systems by Genetic Programming. We do not believe it possible to evolve a complete vision system during the course of a single evolutionary run; early it on it was decided to create a vision system comprising several stages. The first is concerned with low-level vision tasks, or the detection of features, and is discussed in Chapter 6. The further stages of the author's vision system framework are presented in Chapter 7, which covers how GP can be used to produce more complete, standalone vision systems. It also describes how the process of creating them can be automated to the extent that non-expert users can produce vision software. Finally, in Chapter 8 both the capabilities and limitations of the author's system are explored.

The reader is left with a question as this introduction draws to a close. How might one create a vision system, that can learn to analyse medical images, read car number-plates, recognise gestures in real-time or read sheet music, without any form of parameter tuning or expert knowledge? In this thesis we'll discover whether Genetic Programming can find an answer.

Chapter 2

An Overview of Genetic Programming

At certain points in the 20th century, people were especially interested in measuring the "redundancy" within languages – which letters are most repeated, and which words could actually be done without. While the reader may have guessed such information is particularly useful for practitioners of modern cryptography, it also reveals some interesting characteristics about the languages we speak. It turns out that most human languages are about 50% redundant.

Although computer software is rather more restricted in its basic vocabulary, the range of solutions to a particular problem that can be expressed in most programmingt languages can be quite diverse. If one were to instruct a hundred programmers to solve a task, one would probably amass a whole range of different approaches to the problem. Some approaches may be better than others.

Genetic Programming (GP) is a machine learning technique that aims to develop computer software through a similar process of producing redundant software solutions. GP also has the potential, at least, to devise solutions that none of the hundred programmers would have thought of. Fortunately, a squadron of human developers is not required for a process of evolutionary computation – GP is intended to find solutions in a reasonably automatic manner. In this chapter, the author will cover some of the key aspects of Genetic Programming in detail, starting with the general rationale underlying evolutionary learning algorithms.

2.1 The Inspiration Behind Evolutionary Algorithms

The theory of evolution by means of natural selection (“survival of the fittest”[1]) is the established scientific explanation for how intelligent life came to be on Earth. It is the process by which heritable traits that benefit individuals become more and more common within a species, and heritable traits that disadvantage individuals are rapidly discarded; those individuals with beneficial traits are more likely to pass them on to subsequent generations.

Key to the process of evolution is the variation between members of a species. Although life is filled with instances of good and bad luck, mainly outside of an individual’s control, features present in certain individuals but absent in others may cause the former to have a slightly better chance of surviving until breeding, at which point those factors may be passed on, perhaps to spread throughout the species. Although natural evolution is generally not observable during a human life span[2], the cumulative result of this selective process over thousands or millions of generations is a species that has become highly adapted for survival in its chosen environment.

The second key to evolution is the ability of members of a species to store these adaptations reliably. If alternations cannot be recorded, then they cannot easily be accumulated. This is made possible by each individual’s genetic material, or DNA, which encodes everything needed to make a perfect replica. A copy of the individual’s entire genetic sequence is present in every cell: the genetic material of any multi-cellular organism is massively redundant[3]. At the point of reproduction, however, just one set of genes from each parent is required for the creation of a new, genetically distinct, offspring. It is at this point where new variations may be introduced, either following the normal process of combining two sets of parent genes together (recombination, see p.28) or the abnormal event where errors occur during the transcription process (mutation, see p.32). In either case, the offspring may be more, less or equally capable of survival than its parents, which in turn affects its overall likelihood of breeding. And so the process of natural selection proceeds.

[1]First coined by Herbert Spencer in 1864, later integrated into Darwin’s 5th Edition of *On the Origin of Species.*

[2]Exceptions do exist: evolution of species of fruit fly introduced into North America have been observed over periods of less than 20 years [8].

[3]This is a safety mechanism that helps impede the rapid replication of mutated or otherwise damaged cells.

2.2 A Very Brief History of Evolutionary Computation

Almost as soon as the computer age dawned during the middle of the 20th Century, researchers became interested in using machines to simulate the evolutionary process. It soon became apparent that as well as providing insights into the nature of evolution itself, these evolutionary techniques had potential to solve contemporary engineering problems.

The author will shortly introduce three different evolutionary learning techniques, covered in chronological order, to give the reader some idea of evolutionary computation's own evolution. Although the techniques exhibit some significant differences, they each share a common evolutionary process, which is shown in Figure 2.1. The concept is to produce a *population* comprising a reasonably diverse pool of solutions. Each solution is evaluated against certain metrics, the better ones are selected for "breeding" in order to produce new solutions that may become part of a subsequent generation. The process continues iteratively until termination criteria are met.

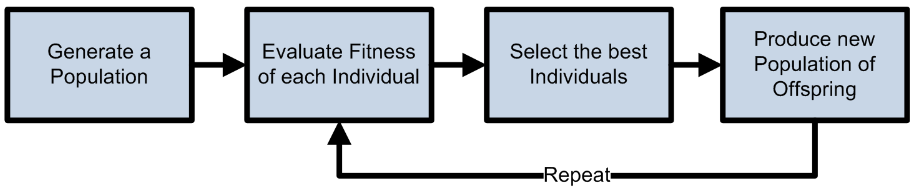

Generate a Population
Evaluate Fitness of each Individual
Select the best Individuals
Produce new Population of Offspring
Repeat

Figure 2.1: The general process of artificial evolution

2.2.1 Evolution Strategies (ES)

Once of the first attempts at artificial evolution was the development of an optimisation technique named Evolution Strategies by Ingo Rechenberg in the 1960s and 70s [9]. In Evolution Strategies, the population is composed of a number of *individuals*, each typically represented as a vector of real-valued numbers (genes). The values are used to make decisions depending upon the task. The chore of finding optimal values is left to the evolutionary process whose search is driven by a mutation operator. The operator perturbs the values of each gene by adding a normally distributed random value to each, simulating the genetic mutations which cause much of the diversity found in the natural world. The parents then compete with the children for a place in the next generation. In contemporary versions, the learning environment comprises a population of "parent" individuals. Following the selection of a number of good parents (the definition of "good" or "bad" in this sense will be discussed later), a population of offspring is generated using mutation and recombination operators; the mutation operator is the dominant means of producing variance.

Evolution Strategies has been demonstrated as a useful parameter optimisation technique for certain real-world problems, but is somewhat limited by the simple numeric representation of its individuals. It does not make use of a particularly accurate simulation of natural selection or genetic recombination. “Classic” ES uses deterministic (greedy) selection, which always selects the top n individuals to become parents. Although this approach initially appears the best way to arrive at a solution quickly, it also significantly reduces the diversity within the population, since the remaining individuals have a zero chance of reproduction. The mutation operator is used to put diversity back into subsequent generations. In general, reductions in diversity may decrease the rate of learning in later generations, leaving the ES learning system potentially prone to becoming stuck in local optima (Figure 2.2). Selection techniques, which usually aim to select better individuals while maintaining diversity, are discussed in more detail in Section 2.3.4.

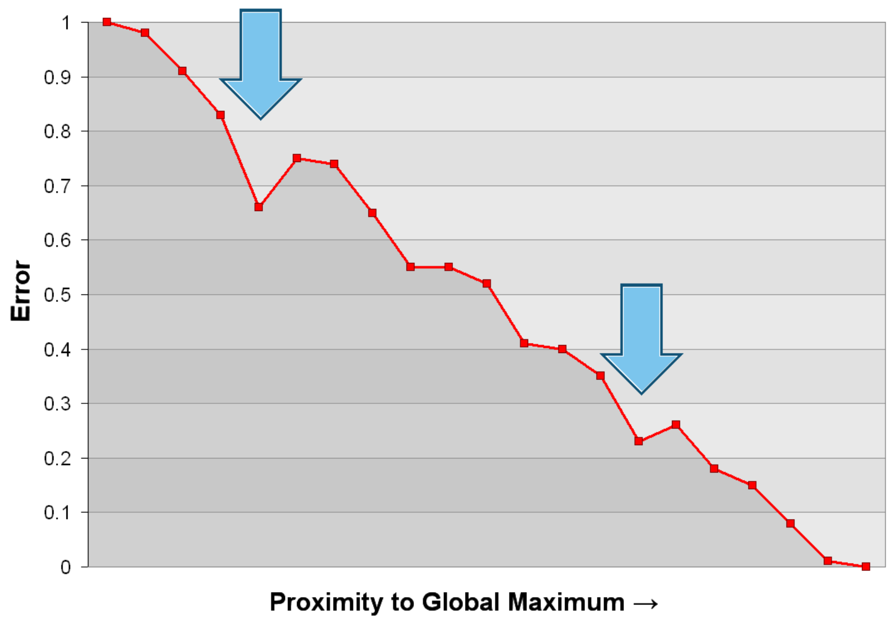


Figure 2.2: Sometimes solutions are only locally optimal. If the population concentrates its search in these areas too much, it may become unable to progress further towards the global maximum.

2.2.2 Genetic Algorithms (GA)

Following Evolution Strategies were the Genetic Algorithms (GA), popularised by the work of John Holland [5]. Holland's work was focused on symbolic as opposed to real-valued representation of individuals. This more abstract approach makes GAs applicable to a wider veriety of different domains. Indeed, Genetic Algorithms are used in a whole variety of different circumstances. Although GA and ES are broadly similar, there are a number of key differences. The most important is that Genetic Algorithms use operators more closely related to our knowledge of genetics, namely crossover (p.28), a more realistic mutation operator (p.32), and inversion[4], together with a less greedy selection implementation.

In essence Holland was more interested in the study of natural adaptation; indeed his technique was a closer simulation of biological evolution than Evolution Strategies.

2.2.3 Genetic Programming (GP)

Although Genetic Algorithms are reasonably abstract, their inherently linear nature remains a limitation (which is apparent in other systems based on GAs, such as XCS[5]). Genetic Programming is an extension of the GA algorithm such that individuals are represented not as vectors of values, but as syntax trees (programs) which can express non-linear logic in the form of computer programs. Therefore, Genetic Programming is not a parameter optimisation technique but a method for automatic programming.

Although several researchers investigated the automatic evolution of *programs* during the 1980s, the modern concept of Genetic Programming is based the work of John R. Koza who published a tome of research on the topic in the early 1990s [11].

The primary difference between Genetic Programming and Genetic Algorithms is the differentiation between *phenotype* and *genotype*. The genotype refers to the encoding of the solution (in nature this is the genetic material, the DNA) with the phenotype relating to the realisation of that solution; the eventual organism and its behaviour. In nature – and Genetic Algorithms – these are distinct: the GA's symbolic genes must be interpreted in some way when applied to domain-specific problems. In Genetic Programming they are often the same, so the programs being evolved may be used directly (although this need not necessarily be the case). Genetic Algorithms, therefore, are somewhat restricted to

[4]Inversion aims to address the need for the function of genes to be independent from their position. It worked by flipping the positions of genes in a section of the individual's chromosome. It is less commonly used in modern GA systems thanks to more sophisticated crossover implementations.

[5]XCS [10] is a classification system which uses a GA as its learning component.

discovering parameters required by certain models to solve a problem; Genetic Programming can be used both for optimisation and the discovery of the model. We shall come back to this distinction later in this thesis – sometimes a separate phenotype is beneficial.

2.3 Components

Evolutionary learning systems usually comprise a number of core components necessary to simulate the process of artificial evolution. In this section, we shall cover in detail four of the most important components with respect to Genetic Programming. This section is intended to give the reader an appreciation of the current areas of research in Genetic Programming and some of its strengths and weaknesses with regard to:

- The Representation and Creation of Individuals
- The Evaluation of an Individual's Fitness
- The Selection of Individuals
- The Choice of Genetic Operators

For a more thorough, beginner-level introduction to Genetic Programming, the author recommends the "Field Guide to Genetic Programming" by Poli, Langdon and McPhee [12].

2.3.1 The Representation of Individuals

The purpose of each GP individual is to represent program code, so that the evolutionary process can search for the program code that best solves a particular problem. GP researchers generally adopt one of three main approaches to represent code. One of which is a linear representation, in which each individual is composed of a list of commands, see Nordin [13]. These commands are usually low-level in nature, and may be used to develop machine-code programs. Linear representations are simpler to implement, as they rely on a more straightforward data structure, but also suffer some drawbacks: it may be more difficult to ensure the program performs meaningful computations (see page 16).

The more common approach in Genetic Programming is to express programming code using a tree structure (see Figure 2.3). The tree consists of a series of nodes which represent programming functions and inputs. The leaves of the tree are referred to as *terminals*, through which input values are provided to the program, either in the form of data acquired from some process or as constants. The remaining non-leaf nodes in the tree take any child nodes as input parameters and perform some processing action on them, and are referred to as *function* nodes. Execution starts at the top *root* node whose children are then traversed recursively. In many cases, the output from the root node is taken as the output of the program. The "size" of a program is usually defined as the number of nodes in its tree. The tree may be extended into a graph (also known as graph-based GP [14]) such as to represent more complex dynamics, such as loops (see Teller and Veloso [15] or Shirakawa *et al.* [16]). However, graph-based GP is less able to support strong-typing and it is more difficult to ensure that programs are efficient.

The advantage of tree-structures is that they permit the dependencies and types of each node to be taken into account more carefully (see page 16), and that they are better able to handle functions which may require different numbers or types of arguments. In this thesis, the reader may assume "genetic programming" refers to "tree-based genetic programming".

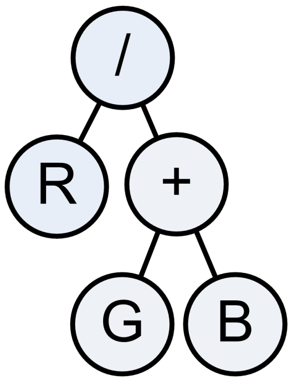


Figure 2.3: An example tree whose function is equivalent to $x = R/(G + B)$.

2.3.2 The Creation of Individuals

As was shown in Figure 2.1, the first stage in an evolutionary computation run is to produce an initial population of programs. Since the system has no means of guessing the solution to a problem, the process is largely random. The expectation is that within the randomly generated population there will be some "good ideas" which can be built upon in subsequent generations. In Genetic Algorithms, the production of random, fixed-length sets of constants or symbols is straightforward, but the nature of GP's tree representation makes the process rather more complex.

Koza proposed three methods for creating random program trees, given a set of function and terminal nodes, and a *maximum-depth* parameter which sets an upper bound on the size of the tree. The process in each case is similar: add children to the root node until its constraints are satisfied (for instance: "I require two children as arguments"), then apply the same process to each child of the root node and so forth.

GROW Builder

Given a set of function nodes and terminals, the GROW builder selects one at random to produce the root node. If the root node requires children then these are added in a similar fashion, until terminal nodes have "completed" every branch. The GROW builder is simple to implement and can run in linear time, and is commonly used in Genetic Programming toolkits. Since the tree generation process is largely unconstrained, the builder can generate a variety of regular and irregular tree structures, that is to say tree structures whose branches *all* extend to the maximum depth, or those trees whose branches do not. However, the size of trees generated by the GROW builder is dependent upon the ratio of function nodes to terminal nodes. If the function nodes have arity n, then if the proportion of function nodes to terminal nodes equals or exceeds $1/n$, then the GROW builder becomes likely to produce trees of infinite size! For this reason a maximum depth constraint D is usually imposed – if the depth is about to exceed D then the GROW builder selects a terminal to complete the branch.

FULL Builder

Koza's second tree builder [11] works similarly to GROW, with a small addition which causes it to build "full" trees in which every branch extends to the maximum permitted depth D (see Figure 2.4). This is implemented by selecting function nodes on *all* occasions except where their addition would bring the branch beyond the depth D. There are some comments that one could make about such a builder. Insisting upon a particular depth for trees appears slightly restrictive, especially considering that there are fewer full-trees than there are irregular trees. The full-ness constraint may reduce the thoroughness with which the search space is explored.

The Ramped Half-and-Half Population Builder

Although the GROW builder is able to produce both full and under-full trees, it suffers from one drawback, namely that the probability of reaching a particular tree size/depth

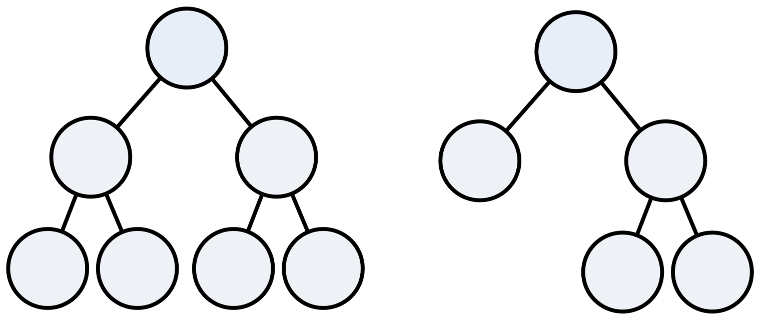


Figure 2.4: The distinction between a "full" tree (left), where all the branches extend to the maximum depth, and a tree developed by the GROW builder, in which some or all the branches may terminate before reaching the maximum depth (right). There are more "unfull" trees than there are "FULL" trees.

is linked to the proportion of terminals and non-terminals. Similarly, the FULL builder is perhaps overly restrictive for particular depths. To mitigate the disadvantages of each technique, Koza proposed a strategy to use both while generating populations. Given a minimum depth D_{min}, a maximum depth D_{max} and a required population size N, roughly $N/(D_{max}-D_{min})$ trees at each depth $D_{min}, \cdots, D_{max}$ are generated, half using the GROW builder and half using the FULL builder. However, as the author will assert later in this thesis, the ramped-half-and-half builder still suffers the drawbacks of each builder. The GROW builder's behaviour is affected by different terminal sets, which clearly will have implications for any generic system. Later in this thesis we will examine how to mitigate some of these problems.

Strong Typing

Koza's early function sets tended to use nodes of the same "type". A set composed of, for instance, the mathematical operators $+, -, \times, \div$ and a series of numeric terminals $x_1, x_2, \ldots, x_n$ are all compatible with each other, since every node returns a numeric value and every function requires numeric inputs. However, more complex programs demand extra types. If one wanted to introduce an $IF(\cdots)$ function (see Figure 2.5), for instance, then it may still return a number dependent on the branch it chose to execute, but its condition would have to be *boolean*.

Although Koza proposed a clunky solution to the problem, an early deviation from his original implementation was the addition of *strong typing* by Montana [17] which permitted the use of different data types within the same tree. Strong typing is a common feature in many programming languages: it prevents the compilation of code that may be seman-

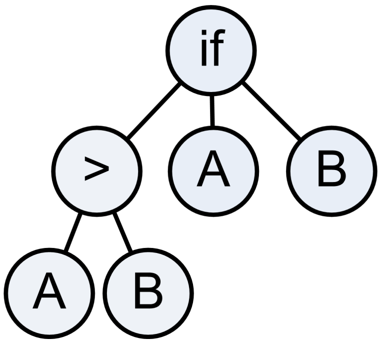


Figure 2.5: An $IF()$ Statement in GP. The first branch is the condition, which decides which of the other two branches will be returned as the node's output. Nodes in this tree use two different types: boolean and numeric. The code statement here is used to implement a $max(A, B)$ function, a useful non-linear transformation.

tically incorrect. As the search space will consist of a number of potential useful solutions but a much larger number of useless ones, the technique is also useful in preventing GP from needlessly evaluating senseless trees and sub-trees.

Strong typing is implemented in the tree building and search operators of the GP system, each of which must ensure that nodes which expect certain types are satisfied. For instance it may be specified that an *add()* node should only take numeric inputs; the GP system will then honour this requirement. The author's toolkit has extended this concept further, which will be explained in Chapter 5.

Other Tree Builders

The generation of trees is important – the algorithm employed inherently defines the extent to which the search space can be explored. Nonetheless, tree builders are a somewhat neglected aspect of GP research. Since Koza's initial work, only a few authors have proposed alternatives.

Iba [18] developed a tree builder named *RAND_tree*, which built uniform trees from the bottom up, joining nodes then sub-trees together such as to build a single tree. This type of approach permits the user some control over the *size* of trees created. However, although depth *limits* size, the average size at a given depth remains dependent on the function and terminal sets. Iba's approach is less easy to implement with strong typing, and takes longer to produce trees than does the simpler GROW method, which can execute in linear time.

Luke [19] made a study of tree builders and developed two algorithms (named PCT1 and PCT2) which could run in linear time but still offer greater control over the average tree size to the user. Luke's PCT1 algorithm starts by calculating the probability $p(t)$ of selecting a terminal (rather than a node) such that the average tree size would be s. Given F functions, each with arity a_i, that have equal probability of being selected, $p(t)$ is:

$$p(t) = \frac{1 - \frac{1}{s}}{\displaystyle\sum_{i=1}^{F} \frac{1}{F} a_i}$$

Luke's PCT algorithm is otherwise identical to the GROW algorithm, except that the functions and terminals are divided into two sets. The probability that a given node is selected from the terminal set is $p(t)$, otherwise it is chosen from the set of functions. Of course, this algorithm does not guarantee that trees will *always* be of the user's preferred size, they will only be of size s on average; indeed there may be significant variance between different trees generated by PCT1 and the likelihood remains that there will be significantly more small trees than large trees, which may account for redundancy in the population.

In order to enforce a more fixed size constraint Luke suggested another algorithm, PCT2, which could generate trees with to fit both a given average size and a given size distribution. PCT2 starts by choosing a random tree size s_r from the user's preferred probability distribution, then builds a tree to match. The tree is built by adding non-terminals at random to open branches of the tree until the tree size + the required number of terminals to "complete" the tree equals or exceeds s_r. At this point, terminals are added to the end of each branch to finish the tree.

Both PCT1 and PCT2 can be extended to support strong typing, although this may sometimes conflict with $p(t)$. As we have seen so far, there is no such thing as a perfect tree-creation algorithm, although Luke's builders would appear to solve some of the problems of GROW and FULL. Still, each can produce redundant trees – trees that are identical to others in the population, because certain tree depths have a limited number of possible tree combinations which is less than the proportion of the population assigned to a particular depth. The author will make additional contributions later in this thesis with regard to ensuring that random populations do not exhibit as much redundancy.

2.3.3 The Evaluation of an Individual's Fitness

Nature implements rather a blunt evaluation of an individual's fitness: the organism will either die before breeding (perhaps following a confrontation or another of life's unpleasantries), in which case its genes will disappear from the population, or it will survive sufficiently long as to reproduce, at which point its genetic heritage is secured for one more generation.

Essential to any *artificial* system which learns through a process of simulated natural selection is some kind of "fitness" function that describes mathematically the performance of a program. This permits direct comparisons to be made between individuals such that some may be selected for breeding. GP generally discovers solutions using a supervised learning paradigm, using a human-generated set of training data that has been marked with appropriate classes. Individuals are then measured by their ability to emulate the human decision making process.

If one is to develop a GP program for the purposes of pattern recognition, then the fitness of an individual should be proportional to the number of patterns that it can classify correctly, or rather the proportion of times with which its output agrees with the human decision. Although humans usually associate higher performance with a higher "score", in Genetic Programming the reverse is the case: fitness is usually measured by counting the number of mistakes. This means that a fitness value of zero signals that the "perfect" individual has been found. In this case the term "fitness function" is something of a misnomer: "error" or "cost" function is more accurate.

The most straightforward fitness function is defined as follows. Given a set of classes $C = \{c_1, ..., c_n\}$ and a set of samples $\{(x_1, y_1, w_1), ..., (x_n, y_n, w_n)\}$, with each sample comprising a feature vector $x \in X$, the correct class $y \in C$, and a corresponding weight w . The fitness of function $f : X \to C$ may be calculated as:

$$error_1 = \sum_{i=1}^{N} w_i[f(x_i) \neq y_i)]$$

In other words, $error_1$ sums the weights of samples mistakenly classified. In the author's experience this kind of fitness function performs well, in spite of its apparent simplicity. Since the function is calculated using only one measure, it is possible that a number of classifiers may be assigned identical fitness; in which case the selector (see page 23), when faced with two or more equally fit individuals, will choose the smallest. The training data error is thus the overriding factor in determining an individual's fitness; other desirable features are secondary. We shall see alternate means by which competing criteria may be accommodated later in this section.

Different Errors, Different Costs Some problems demand slightly more complex fitness functions which permit certain adjustments to be made, usually relating to the relative cost of making different kinds of mistake. A classifier can generally make two types of mistake: either a *false positive*, where a sample from another class is incorrectly identified as the class in question, or a *false negative*, where a genuine sample from the class in question is not identified as such[6]. In some cases, notably medical diagnosis or credit management, the cost associated of each kind of mistake is different. When diagnosing disease the false positive may be alarming to the patient but is substantially less costly than the false negative, which can leave a person unaware of a potentially serious health problem which, upon its eventual discovery, would no doubt leave the indivudal much worse off and the medical establishment facing a substantial claim.

The relative importance of different mistakes can be encoded by introducing extra coefficients α and β into the fitness function. An example, used by Roberts & Howard [20], is:

$$error_2 = \frac{\alpha TP}{\alpha N + \beta FP}$$

where TP is the number of correctly classified samples (true positives), FP is the number of incorrectly classified samples (false positives), and N is the size of the training set. This fitness function produces a one-dimensional bias in favour of either sensitivity (maximising the true positives), or specificity (minimising the false positives) depending on the ratio of α and β. This provides an additional level of control that may be useful in tuning the system for problems in which outcomes have different costs. For example False Positives are acceptable for a cancer screening application, provided that the sensitivity is at or around 100%.

Extra Embellishments Penalties or rewards may be added to the fitness function to provide incentives for certain behaviours. For multi-class problems, the fitness function may include a penalty if at least one sample from each class is not classified (see Teredesai [21]). Although the principle of “guiding” the classifier in this way may yield desirable outcomes, such a penalty also creates a large fitness-step which may be disproportionate to the improvement in ability. In Chapter 4 we’ll see some of the other approaches to multi-class classification by GP.

[6]Although the false negative in one class may be a false positive for another.

Parsimony Pressure Interestingly, the author's experiments have shown that the most effective fitness function is often $error_1$, which, as we have seen, simply sums to the number of mistakes out of the total training samples. It might well have been the preferred technique of William of Ockam, the 14th century logician, had he been born six hundred years later and with an interest in evolutionary computation. Ockam's benchmark principle, now known as Occam's Razor, states that the simplest solution is usually the best one[7]. Simple fitness functions, however, do not necessarily yield simple, parsimonious solutions. In fact Genetic Programming is rather prone to producing over-complicated solutions to problems, described as "bloat" (covered in more detail in Chapter 5). Unlike GA individuals, which generally use fixed-length chromosomes, there is no arbitrary limitation on the size of GP trees, so they can grow considerably. Parts of trees which do not contribute to fitness are described as *introns*[8]. Introns are undesirable as they can reduce the efficiency of the learning process.

For this reason, a parsimony component is sometimes added to the fitness function, the size of the penalty depending on the individual's tree size. This creates a fitness pressure towards smaller, simpler individuals. The extent to which parsimony is pressurised is determined by a parsimony coefficient c, the choice of which requires deliberation of its own. Parsimonious fitness functions are composed of a regular fitness metric f plus an additional penalty associated with the individual's size s:

$$error_{parsimony} = f + cs$$

Of course, particularly large values of c will cause the program to treat size as its main objective and ignore f altogether! Poli and McPhee [22] suggested a means by which c could be calculated automatically in order to maintain a given average population size.

A further discussion of fitness with more specific regard to machine vision problems awaits the reader in Chapter 5.

Multi-Objective Optimisation We have now seen a couple of examples where two objectives are incorporated into a unidimensional fitness function. In the case of classification $error_1$ is quite often sufficient. However, research into multi-objective optimisation (MO-O) deserves a mention here, not least because of the desire in some cases to gold-

[7]The modern interpretation is slightly different to the original principle: "*entia non sunt multiplicanda praeter necessitatem*". which roughly translates to "entities must not be multiplied beyond necessity".

[8]The term is borrowed from genetics, relating to DNA portions in genes which apparently do not perform any function during the production of proteins.

plate the definition of a useful solution with other beneficial characteristics. The crucial problem with MO-O is that it isn't possible to define "optimum" in a scalar way given two or more objectives, since different criteria will usually conflict with each other. In most cases it is only possible for individuals to represent good compromises rather than globally optimal solutions. These individuals, which represent the best possible trade-off between two or more criteria, are said to lie on a *Pareto Front*.

It is necessary, therefore, to consider the optimal solution instead as a *set* of solutions. Evolutionary algorithms are particularly suitable in this regard because they deal simultaneously with populations of possible solutions. Two issues immediately arise: how to define the local optimality of solutions, and how to ensure that the population does not become dominated by a single solution. Various approaches have been proposed which fall into two broad categories.

The main approach to multi-objective optimisation are the *Pareto-based* approaches which do not attempt to define a fitness term which can never be optimal. Rather they evaluate individuals according to their proximity to the Pareto front, which is based on the concept of *dominance*. An individual is said not to be dominated if there exists no other individual in the population that can improve a given objective without causing a simultaneous reduction in another objective. An individual is Pareto optimum if there exists no feasible individual that can do this. There many be many or an infinite number of Pareto optimal solutions; these are distributed along the Pareto Front. There are rather more solutions than necessary: popular GA toolkits such as NSGA-II try to ensure that individuals in the population are distributed over the Pareto front. The problem, as ever, is that assessing non-dominance is computationally expensive, especially for large populations or large numbers of criteria.

It has been established that Pareto optimal solutions may be discovered using a scalar, rather than multi-objective measurement of fitness, in which the objectives are all aggregated into a single criterion. There are various approaches to doing this. Syswerda [23] used an approach where the fitness with respect to each criterion is measured, then combined together in a weighted sum. The question then becomes how best to define the weights for each criterion – another optimisation problem!. So-called lexicographic ordering (Fourman [24]) is more readily definable – the objectives are placed in order of importance. In a competition between two individuals, the first objective is compared first; in the event of a tie the individuals are evaluated against the second criterion and so forth. This approach works well provided there are few objectives – objectives too far down the pecking order are unlikely to be taken into account at all. Schaffer suggested evaluating different sub-populations with different fitness functions, an idea later implemented by Fonesca [25]. The consequence of this technique is that it selects individuals that can excel with regard to one criterion, but not necessarily both.

This completes a very brief survey of multi-objective optimisation. Later in this thesis the author will touch upon a couple of multi-objective fitness functions: however in general the addition of extra criteria inevitably harms the accuracy of the solution in classification. Other criteria, such as efficiency, are desirable extra characteristics, but we will see that there are other means to address bloat without compromising the fitness function.

Generality and Validation Sets While the fitness function exists to measure the accuracy of a GP solution on a set of training samples, it does not necessarily indicate how robust or general the solution may be on data that it hasn't seen before. 1-Nearest Neighbour algorithms, for instance, are immediately capable of 100% accuracy on training data, but may fail to achieve decent results on test data[9].

Initial experiments using Genetic Programming reported the performance of the evolved programs on the same data sets that had been used to train them, a practice which gives little indication of the actual robustness or usefulness of the evolved algorithms. Later research split data sets partitioned into training and validation partitions, with the latter used to provide a better indication of the performance on unseen data. We shall take a closer look at validation data later in this thesis.

Another means of encouraging "generalists" is to evaluate the individual on a series of different datasets (or in the case of machine vision, images), and calculate the fitness individually for each. The individual's *worst performance* is then used as its overall fitness (see Ebner [27]). This removes some of pressure towards *over-fitting* the training data, which usually degrades the program's ability to generalise and is an issue with non-parametric learning methods.

2.3.4 The Selection of Individuals

One of the primary concepts of Darwin's theory of evolution by natural selection is that species become better adapted to their environment by "survival of the fittest". Artificial evolutionary learning systems, therefore, are motivated to select "better" individuals while discarding "worse" individuals. After the fitness of each member of the population has been computed, certain individuals are then selected to be "parents" of the next

[9]k-Nearest-Neighbour may also be slow to run on large data sets, although a Best-Bin-First (BBF) (see Beis and Lowe [26]) algorithm can be employed to speed up the distance matching.

generation. The characteristics of good individuals are thus more likely to re-appear in subsequent populations, at which point the genetic operators will attempt to refine them further.

In this chapter we have already encountered so-called *deterministic* selection (also referred to as *truncation* selection), used by Evolution Strategies, in which the top n individuals in the population are guaranteed to be selected as parents. Indeed, given the extremely limited computational resources available in the 1960s, this was probably the only efficient means of getting a solution. However, selection methods are guided by a well-known principle which states:

> "The rate of increase in fitness of any species is equal to its genetic variance in fitness." *R. A. Fisher, 1958 [28]*

So if deterministic selection chooses the best n individuals in the population, all other individuals have a zero probability of being involved in breeding. This means that the variation in subsequent populations is derived from only a small, fixed subset of individuals in the current population, and over time the diversity may be severely curtailed, although the genetic operators will continue to reintroduce diversity. Diversity in the population may be better maintained if each individual in the population is permitted at least some chance of reproduction.

At the heart of the selection process, therefore, is a dilemma concerning the desire both to utilise the most successful individuals immediately and to preserve the diversity of the population as a whole – which may come in useful at a later stage. Good selection algorithms should incorporate both a bias in favour of better individuals and sufficient randomness as to maintain a broad range of different solutions within the population.

Three main selection algorithms are in common usage in the GP community: Fitness Proportional Selection, Rank Based Selection and Tournament Selection, which shall be discussed in turn.

Fitness Proportional Selection As its name suggests, this technique selects individuals from the population with a probability proportional to its fitness. This may be implemented in practice using stochastic sampling ("roulette wheel" selection), in which a number of "markers" are assigned to each individual, the number proportional to its fitness. To select an individual from the population, a marker is randomly chosen and the individual associated with that marker is selected. Sampling is repeated n times until a sufficient number of parents have been selected. Dependent on the resolution of the stochastic sampling, some individuals may have such low fitness that they are not assigned a marker at all, so the technique is biased. Unless sampling is made without replacement, there is the potential for a single, very fit individual to dominate the breeding population – at the expense of diversity.

For these reasons, Stochastic Universal Sampling (SUS) was introduced by Baker [29]. In SUS, the individuals are shuffled before being placed on the roulette line with each marker a fixed distance from the last (see Figure 2.6). The markers are then chosen all at once. Each individual has a non-zero chance of selection, and the selection of individuals is fast. SUS has zero bias, and reduces the potential for a single individual to remove all the diversity.

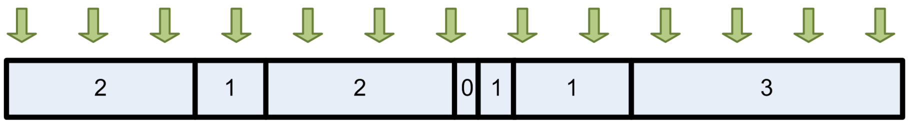


Figure 2.6: Stochastic Uniform Sampling. Each individual is allocated a given width on the line, according to its fitness. Individuals are shuffled then placed on the line. A set number of markers is placed along the line, equally distributed. The number of markers associated with each individual determines the number of times it may be involved in breeding.

Still, even the enhanced forms of fitness proportional selection suffer from two further drawbacks: an individual with an unusually high fitness remains likely to dominate the intermediate parent population at the expense of diversity; and fitness proportional selection does not provide a means of adjusting the *selective pressure*.

Rank-Based Selection This algorithm is similar to fitness proportional selection, except that the raw fitness is discarded in favour of the *rank* of an individual, following the sorting of the population by fitness. The individuals are ranked in descending order. The index j of the selected individual may be chosen according to the following formula by Whitley [30]:

$$j = \left\lfloor \frac{N}{2(c-1)} \left(c - \sqrt{c^2 - 4(c-1)r} \right) \right\rfloor$$

The variable c determines the selective pressure, or the slope of the linear distribution. N refers to the size of the population. The function converts the output of r, a uniform random number, into a linear distribution which is more likely to select lower-indexed, fitter individuals in the population.

Rank-Based selection has, therefore, two advantages over Fitness Proportional Selection: the selective pressure can be adjusted, and the selection technique is less prone to outlying fitness values. However, the sorting of the entire population introduces an additional computational overhead.

Tournament Selection Arguably the most commonly used algorithm used for selection in evolutionary learning systems is Tournament Selection, on account of its simplicity to understand, model and implement, both on a single machine and on parallel systems.

Tournament selection consists of two stages: sampling and selection. t individuals are sampled from the population, the most fit among which is selected as the winner. The selective pressure may be increased by using higher values of t. The choice of tournament size t determines the probability that an individual will be sampled at all (discussed by Poli [31]) and also affects the diversity and quality of the next generation. $t = 1$ tournament selection is akin to random search – diversity will be maintained but there is no selection pressure at all. When t approaches N (the number of individuals in the population) the best individual in the population will come to dominate the parental population, but any diversity in the subsequent generation will be severely curtailed. It is therefore necessary to choose a reasonable value of t, which (in combination with N) defines the selective pressure.

Tournament selection does produce a sampling bias. There is a chance, due to the random nature of tournament selection sampling, that an individual may never be selected. This probability can be calculated as:

$$p_{notselected} = \left(1 - \frac{t}{N}\right)^N$$

The formula assumes that the number of tournaments is equal to N, which is true in standard evolutionary breeding process where one parent produces one child via mutation and two parents produce two children via crossover. In a population of 500 individuals with a tournament size of 2, the probability of an individual not being selected is 13.5% (which is relatively constant for different values of N). Therefore 13.5% of individuals are evaluated even though the result of the evaluation, their fitness, is never used: they never compete in a tournament.

One solution, first suggested by Poli [31], is that individuals which are not included in any tournament should not be evaluated. To do this, Poli pre-calculated the memberships of a series of tournaments across a given number of generations (in the form of a graph), so that individuals not participating need not be evaluated, or indeed created. Although this is a tidy scheme which does avoid unnecessary fitness evaluations, it is perhaps slightly overkill in the case of evolved vision: the *creation* of individuals will generally take only a fraction of the time it takes to evaluate individuals on image data.

A simpler scheme was suggested by Xie, Zhang and Andreae [32], named "evaluated-just-in-time" where individuals are evaluated only once they enter a tournament. While certain individuals are still created but never evaluated, the authors pointed out that the bulk of computational expense comes from the evaluation rather than the creation of the individual.

Perhaps the ideal solution is to ensure that the problem of sampling effects leading to selection bias does not arise in the first place, this should be better able to maintain diversity and makes use of the entire population. Sokolov and Whitley [33] suggested a different form of tournament selection termed uniform tournament selection. If each individual is located according to its index i in a population of size N, where $i \in I$ and $I = \{0, 1, 2, \ldots, N - 1\}$. Each individual is allocated a tournament partner whose index p_i is selected such that $p_i \in I$ and $p_i \neq i$. The individual at index i then competes with the individual at p_i. This guarantees each individual at least two tournaments. The best individual in the population will be selected exactly twice, and the worst individual will not be selected at all. This is the same idea as suggested by Goldberg [34]. However, the value of t is then fixed at 2; it can be extended to higher values of t, but ensuring that tournaments are unique becomes messy.

Island Models Although more applicable to the evaluation of fitness than to selection, it is certainly the case that evolutionary algorithms require more computational resources than other learning systems owing to the wasteful nature of the search process itself. Fortunately, evolutionary algorithms may also be executed in parallel with relative ease as the time-consuming evaluation of individuals can be delegated to other machines. The relatively fast process of selection and recombination can be left to a single machine.

A different form of parallelism, implemented using a so-called island model (see Pettey *et al.* [35]), may also be implemented in software, this time to increase diversity and decrease premature divergence. Again inspired by observations from nature[10], the island model presumes that if the population is split into sub-populations then each may tread a different path through the search space. Allowing the sub-populations to inter-breed occa-

[10]It is a common observation that distinct species arise following geographic separation, for instance the Galapágos Islands Iguana or, closer to home, the four-horned Loghton sheep, which is unique to the Isle of Man.

sionally may help share the expertise of each sub-region while maintaining more diversity that might otherwise be the case in a single population.

2.3.5 The Genetic Operators

Following the evaluation and selection of individuals, new programs are generated by adjusting or combining the parents in order to find new and better solutions. In Genetic Programming, these adjustments are usually performed by two key operators: crossover and mutation.

Crossover

The main genetic search operator employed in Genetic Algorithms, and subsequently Genetic Programming, is crossover, a process inspired by genetic recombination in the natural world. In essence, two parent genomes are combined in such a way as to create a new individual that is similar to but genetically distinct from both parents (see Figure 2.7).

"Standard" crossover is a simulation (and gross simplification) of the operation that happens during sexual reproduction. It works by selecting one sub-tree from each of two "parent" trees at random[11]. The sub-trees are then exchanged between the parents to produce either one or two new individuals. These "children" are subsequently inserted into the new population.

The reasoning behind crossover stems from the idea that fit individuals are composed of useful "building blocks". By exchanging building blocks *between* individuals, the system may find new and improved building block combinations which yield fitter individuals. In most Genetic Programming systems, crossover is the dominant search operator, the main driver behind the search process. The advantages of crossover were analysed by MacKay [36], who showed that it permits useful sub-trees to spread rapidly through the population while rapidly removing unhelpful ones.

Common Genetic Programming wisdom, inherited from Genetic Algorithms, suggests that crossover is the best operator for genetic search. However, Genetic Programming and GAs differ in various significant ways, chief among them being that the nodes in a GP tree are not independent: the output of a parent node is dependent on the output of its children[12]. Therefore, a single change lower down the tree can "break" the entire indi-

[11] If the trees are strongly typed, constraints may be applied to ensure that both trees are compatible.

[12] The author acknowledges that genes in Genetic Algorithms can also be dependent in certain circumstances.

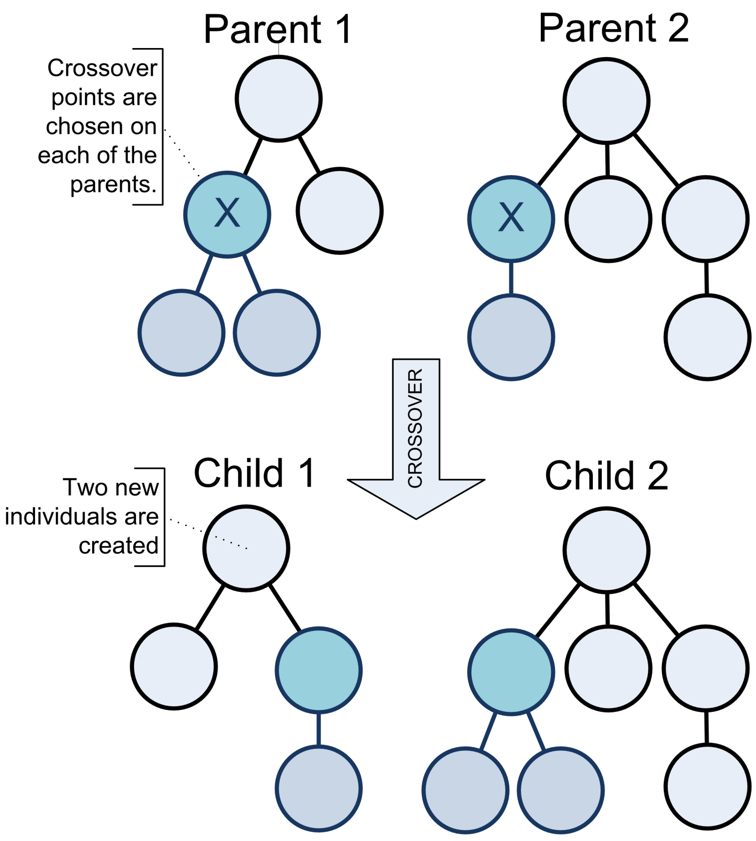

Parent 1
Parent 2
Crossover points are chosen on each of the parents.
X
X
CROSSOVER
Child 1
Child 2
Two new individuals are created

Figure 2.7: The crossover operation. Given two parents, a point on each is selected as the crossover location. The sub-trees are then swapped between parents to yield two distinct offspring.

vidual. Crossover, therefore, is rather a blunt instrument, since it exhibits no intelligent strategy for selecting *suitable* crossover points. The random means by which sub-trees are selected may result in otherwise fit programs being severely disrupted. Experiments by Nordin *et al.* [37] showed that the average fitness of child programs is less than half the fitness of the parents following about 75% of crossover events for certain tasks: standard crossover often has decidedly injurious effects! Furthermore, if the mean population fitness is reduced, then the few remaining fit individuals will be selected more and more often, leading to a reduction in population diversity.

A second consequence of this harmful behaviour is that individuals with higher proportions of useless code (that is to say code that does not contribute towards the individual's fitness), are more likely survive crossover intact. This is one explanation of code *bloat* in Genetic Programming.

The main explanation of bloat in GP is the so-called crossover bias theory. Although crossover should not change the average size of a population, it is observed the distribution of trees produced by crossover is characterised by substantial numbers of very small programs, due to the bias of the operator to choose nodes from near the leaves of the tree. For most non-trivial problems it is unlikely that very small individuals will be able to solve the task satisfactorily: the larger individuals will be fitter, on average. The theory states, therefore, that there arises a selective pressure in favour of larger programs.

Crossover also provides the *mechanism* by which bloat may be introduced: in GAs the two alleles swapped are always the same size; but in GP they may be between sub-trees of different sizes, which provides a means for programs to grow and grow. Approaches to controlling code bloat will be described in further detail in Chapter 5.

Poli and Langdon [38] demonstrated through modelling and empirical investigations that standard crossover was also local and biased. Local search generally yields offspring which are not very different from their parents, and may not provide a thorough exploration of the search space. Standard crossover is biased towards smaller sub-trees, since it is more likely to make adjustments near to the leaves of the tree (there are more of them). In Genetic Algorithms, by contrast, each individual is usually of the same length and each position has the same probability of being selected, so recombination is inherently less biased. For various reasons, therefore, it can be concluded that GA crossover and GP crossover are not quite equivalent. Poli and Langdon suggested versions of GP crossover that are generalisations of corresponding GA operators, including "one-point crossover" and "uniform crossover", which are intended to reduce the bias and local nature of standard crossover.

A substantial amount of research has been devoted to addressing the shortcomings of crossover in the GP environment; the proposed refinements take many forms. One approach aims to "undo" any destructive crossover events by only permitting recombined individuals into the next generation if they are at least as fit as their parents. If crossover does produce poor quality offspring, then they are discarded. This is generally referred to as "non-destructive crossover". The crossover operation may have to be repeated several times until a successful outcome is reached. Usually it is tried only a fixed number of times to prevent the evolutionary process from stalling.

Following a similar train of thought, Tackett [39] noted that animals often produce large broods owing to the likelihood that a number of their offspring would not survive[13]. He proposed that crossover between parents should be repeated n times to pro-

[13]This is especially the case with fish. Wild salmon, for instance, usually produce broods of thousands of offspring – following their epic journey through river and ocean, on average fewer than ten make it back to

duce a "brood" of $2n$ individuals (instead of the usual two), from which the best pair would be chosen. This would increase the probability that some offspring would survive crossover[14]. Although this helps to ensure that the crossover operator is less destructive while still permitting an exploration of the search space, it does impose a significant amount of additional processing.

The author proposes that another way to make crossover less destructive is to use a classification procedure that is not dependent on a fixed output range or threshold, which reduces to some extent the dependencies that make GP individuals fragile. We shall cover one technique, so-called dynamic range selection, in Chapter 4.

Iba *et al.* [40] attempted to adjust non-destructive crossover to remove its local bias and make it concentrate on larger, more influential sub-trees. They did this by choosing first a depth d at which to cross over, then selecting a node at depth d to create the crossover point. Curiously they assigned weights to each depth i such that the weight associated with each depth is $1/2^i$. This would appear to generate a bias in the other direction.

Lang [41] proposed another crossover strategy, dubbed "headless chicken" crossover, where only one parent was selected from the population and then crossed over with a randomly generated individual. Crossover was repeated until the offspring was better than the parent – a form of hill climbing. While Lang showed that his method could outperform standard crossover on a multiplexer problem, Zhang *et al.* [42] showed that while it could still get better results than standard crossover, it was also significantly slower. Crossover with a random individual is analogous with point mutation, with the exception that it is more wasteful.

If humans were in charge of the crossover operator, we might make more effort to reach an intelligent decision as to how to swap sub-trees. In programming, we might like to change a piece of code, but we usually need to ensure the replacement is still compatible with the rest of the program. D'haeseleer [43] invented so-called *context-aware* crossover, which aims to do something similar. The context of each node is encoded as an n coordinate array $A = p_1, p_2, ..., p_n$, where n is the depth of the node in the tree and p_i is the position in the tree at depth i. D'haeseleer suggested operators where sub-trees could only be exchanged if they had identical contexts. The problem with this approach is that it does not readily permit new, possibly better contexts to be discovered by crossover.

spawn the following year.

[14]Tackett's analogy is not strictly accurate, of course: offspring in nature are likely to die as a result of factors relating to their environment, not because of their individual genetic predispositions.

The author would further submit that the *position* of a node in a tree is not necessarily an ideal measure of its context.

Another approach is to evaluate the usefulness of each sub-tree, so that they may be protected. Hengraprohm and Chonstitvatna [44] proposed that the “value” of a particular sub-trees could be identified my measuring the difference in an individual’s fitness if the sub-tree were removed and replaced with a function-less node. This procedure is used to identify the “best” and “worst” nodes in the tree. A single offspring is created by replacing the worst node in one parent with the best node from the other, a process termed *selective crossover*. Hengraprohm and Chonstitvatna demonstrated that their operator could converge on solutions with around 20% less computational effort. The reader should note that computational effort is a measure of how many individuals are evaluated before the population converges on a solution. Although the “computational effort” was lower, the amount of processing per-individual would have been significantly higher, so Hengraprohm ’s and Chonstitvatna’s algorithm would not have yielded individuals more quickly.

In summary, one can see that there is a number of strategies for dealing with the side-effects of crossover. It is perhaps possible to guide the crossover operator to more intelligent decisions, but doing so requires a lot of additional computation. Unfortunately there appears to be no analytical means of estimating the fitness or usefulness of a particular sub-tree: in all cases, the addition or subtraction of a sub-tree from a program can only be measured empirically, which is computationally expensive. If GP is used to construct vision algorithms, which are already computationally intensive, the addition of certain hungry crossover operators is likely to make the process excessively slow.

Mutation

The second genetic operator in common use in Genetic Programming is mutation. In nature, mutation is the name given to the rare event during which parental DNA is incorrectly transcribed or copied. Although mutations occur in our bodies with surprising frequency, their effect is usually confined to just one cell in the 50–75 *trillion* cells in an average human being. Still, we all start off from a single cell, so any transcription error during the production of that cell will become part of every subsequent cell in the offspring’s body. Despite our perceptions to the contrary, mutation is usually benign in the natural world, thanks to the redundancy within DNA; other times it is harmful and occasionally it is beneficial. It is mutation more than crossover which is believed to bring about the “unusual” changes that can trigger leaps forward in the evolution of a species.

Mutation is commonly used in artificial evolutionary systems. In Evolution Strategies, mutation was simulated by perturbing the numeric values of genes. In Genetic Programming, mutation works by manipulating the tree. The GP mutation operator works by taking a single parent, discarding a sub-tree at random and replacing it with a new one, commonly generated using the GROW tree builder. The adjusted parent is then inserted into the new population.

As we’ve seen, the consequence of fitness-based selection techniques and crossover is to make certain sub-trees more abundant in the population, while discarding others. This leads to a reduction in diversity in the population, which may cause the search rate to decrease. The rationale is that mutation can help maintain the diversity in the population by occasionally adding new sub-trees or re-introducing previously discarded ones that may yet be useful.

The mutation rate in GP populations is usually set much lower than crossover: between 10–20% of the population are generated by the mutation operator, in recognition of the fact that mutation makes completely new additions to an individual, rather than swapping sub-trees known to make up reasonably successful individuals.

In common with crossover, mutation may have lethal effects on individuals. Indeed, many of the issues associated with crossover also apply to mutation. Accordingly, issues like local bias and destructiveness may be dealt with in a similar fashion as discussed earlier.

Elitism

The final means by which individuals may pass into subsequent generations is though *reproduction*. Reproduction refers to the insertion of an individual directly into the next generation without subjecting it to either crossover or mutation. Since reproduction doesn’t involve any further exploration of the search space, it is usually left at a low level (around 5%), or reserved for so-called *elite* individuals.

Elitism is a protective mechanism for situations where the selection mechanism fails to select the best individuals in a population, or where crossover produces offspring that are worse than their parents. Elitism guarantees that the n best individuals will be copied into the next generation. These individuals are not affected by crossover or mutation and therefore will continue to exist until better individuals take over the elite set. This is distinct from deterministic selection since each elite is only copied to the next generation *once*.

As we have seen, the genetic operators are designed to discover further refinements given a number of selected parents, but can also produce offspring with significantly worse fitness. We have also discussed code bloat, manifested in GP programs as useless code fragments or *introns*. Introns provide a form of protection against the effects of the genetic operators: introns are expendable. The *Baldwin effect* is an observable phenomenon, in which the genes (and introns) of successful individuals become more and more common in the population through their increased breeding potential. Introns bloom in GP populations because, although they don't make an individual any better, they might help it to survive.

In addition to preventing the population's maximum fitness from regressing and helping to ensure a higher end-of-run fitness, elitism also appears to reduce code bloat [45, 46], as it reduces some of the advantage that introns offer to the survivability of individuals.

One of the problems of GP, exemplified by issues with crossover and mutation, is that crossover lacks a mechanism for preserving the useful information it has discovered so far, beyond the hope that natural selection will cause useful sub-trees to become more abundant within the population. Andre [47] suggested the use of Automatically Defined Functions (ADFs) in which the tree builder could take advantage of whole sub-trees as well as using the basic nodes. It is intended that this technique would allow the GP process to generate and keep those building blocks in order produce complex programs more easily. However, as we've seen in the case of crossover and mutation, it is difficult to assess the usefulness of sub-trees in a computationally efficient manner, so the identification of ADFs is not straightforward. We shall consider another approach by which useful code fragments can be preserved in Chapter 4.

2.3.6 Generation Gap Methods

Having covered selection as a means for selecting good parents for breeding, it is also worthwhile considering the natural consequence of such an action – the removal of another individual in order to maintain the population at the same size. We saw that in the classic version of Evolution Strategies, the parents compete with their offspring for a place in subsequent generations, a scheme referred to as *plus* selection, often represented as $(\mu + \lambda)$, where μ is the size of the parent population and λ the size of the new offspring population. Plus selection bears much in common with a different paradigm, referred to as steady-state evolution, in which the concept of separate “generations” is discarded in favour of producing a much smaller number of offspring following each process of evaluation. In plus reproduction and steady state reproduction a *deletion schemes* is necessary in order to remove the *worst* individuals from the combined population (see GENITOR [30]).

A more common approach, alluded to throughout this chapter, is referred to as *comma* reproduction, denoted by (μ, λ), in which the offspring population completely replaces the parent population – there is no overlap or competition between the two, so all individual “lifetimes” are of a fixed length. The deletion scheme in comma reproduction is therefore trivial – simply discard all the parents.

At first glance, it appears that plus reproduction is the more effective means of going about evolution. It bears something in common with non-destructive-crossover, which helps ensure that the genetic operators do not cause population fitness to regress and it does not discard parents in an arbitrary fashion. Plus selection is an elitist strategy: parents will remain in the population until such a point as offspring can outperform them. However, if elitism is applied to half the population instead of just a few individuals, it can cause the population to become stuck in local minima; it isn’t possible for individuals to regress and explore different avenues of exploration. Substantial amounts of elitism will cause the same parents to reproduce repeatedly, which compromises population diversity.

Wakunda and Zell [48] proposed a different selection scheme intended to help maintain diversity within plus and steady state populations. They also showed that comma schemes required more computational effort – more evaluations – in order to achieve a desired degree of fitness on certain problems. Holland and Reitman [49] found that generational reproduction was not suitable for *classifier systems* where each classifier is to some extent dependent on the others as decisions are made by the population as a whole (see Wilson [50] for a more definitive version). They found that replacing only a small part of the population at a time was more beneficial than replacing the whole population in one go. Eggermont *et al.* [51] performed a comparative study that appeared to show that generational GP yields slightly better results, on average, than does steady state GP on certain classification problems. This may be because the steady state approach does not encourage learning to the same degree. De Jong [52] stated that the advantages of overlapping populations were offset by the effects of genetic drift (loss of diversity). Like many areas of research in Genetic Programming, generation gap methods exemplify the different approaches taken to solving problems; each is applicable to certain situations.

2.4 Conclusion

In this chapter, the author has covered in detail the main components of evolutionary systems in general and Genetic Programming in particular. An issue throughout, highlighted in the section on multi-objective optimisation, is that there are often conflicting demands for each component. Indeed, it is quite difficult to describe any of these components as being optimal in all circumstances; we've seen that most have particular advantages and particular disadvantages. Consequently it is difficult to write down *the* definitive set of parameters, although this is desirable if GP is to be rendered into a "black-box" learning system for evolving vision systems in general. Later in this thesis the author will go about describing suitable choices that make GP suitable for learning vision components in particular. We shall now turn our attention to those vision system components – the major challenges and areas of study in computer vision, and the quality and extent of vision software evolved by GP researchers thus far.

Chapter 3

A Brief Overview of Computer Vision

In 1932 a small group of students at Harvard Business School commenced work on an ambitious project. They were lead by a man named Wallace Flint, whose idea was to develop a system by which products in a catalogue could be identified by a unique code that was readable by machine. Given a set of punch-cards from the customer, the system could then go about producing an invoice and updating the stock count automatically. Unfortunately for Flint and his students, the world was in the midst of the worst economic depression of the 20th century and his idea failed to gain momentum. It was another twenty years before the birth of the modern barcode and a further two decades before it finally caught on in a mainstream sense in the mid-1970s. It could be argued that barcode readers are now the most ubiquitous machine vision systems in use throughout the world – indeed it would be inconceivable for most modern stores to operate without them. The use of barcodes has extended beyond retail too – they are used for ticket validation in sports venues, provide mobile-phone-readable hyperlinks to web pages, and have been employed in miniature form in order to identify and track the movement of individual bees. Like most machine vision systems, barcode readers automate and simplify complex or tedious aspects of modern life; such systems are both desirable and profitable. Unfortunately vision in a truer sense cannot be confined only to the highly-constrained patterns apparent within barcode patterns, but should be applicable to images in general.

The reader will probably be familiar with the well-known saying that “a picture is worth a thousand words”, which somewhat *understates* the complexity inherent in imagery. Indeed if it were possible to summarise any image in a mere thousand-word précis, the work of the computer vision developer would be easier! Still, most of us are familiar with the ability of computers to process complex data much more rapidly, accurately and patiently than can we, but software is typically dependent upon the data being well organised or carefully encoded. Images, by contrast, are usually characterised by large amounts of unstructured, redundant or irrelevant information that hamper the machine's ability to perceive that which we can see so easily. Computer vision is the area of study concerned with making sense of images in such a general sense. It incorporates a broad range of different sub-topics including signal processing, segmentation, object detection and localisation, image matching, motion detection and tracking, classification, recognition, 3D reconstruction and many others. Rather than attempting a comprehensive review of the field, the author shall instead concentrate this brief survey of the vision literature to a selection of topics, some of which will be discussed in more detail later on in this thesis. For more substantive introductions to computer vision, the reader is recommended to read textbooks such as those by Parker [53] or Davies [54].

In this chapter the author will compare the work of conventional machine vision researchers to equivalent solutions discovered by Genetic Programming and machine learning techniques in general. Where possible, the relative quality, applicability and advantages of machine vision systems shall be discussed; towards the end of this chapter the application of Genetic Programming to vision problems shall be assessed in a more general sense, setting the scene for the author's work.

There are various ways in which one can organise a survey of computer vision; in this chapter we shall take a "bottom-up" approach and start with the lowest level image operations, working upwards to higher-level algorithms.

3.1 Image Acquisition

All vision systems require one or more images as input, which may be captured through a digital still camera, video camera, or scanner. Whatever the device, a characteristic of any electromagnetic signal captured by electronic sensors is its liability to a degree of disruption or noise. It is preferable that this be minimised before image processing can start in earnest.

Noise arises from three sources. Two of these, *readout* and *thermal* noise, can be minimised through better sensor circuitry, but the dominant source of noise in images collected by digital sensors – photon or *shot* noise – can not. This is caused by the quantum nature of photons themselves and the non-regular rate at which they are emitted and subsequently detected. Although it is easy to think of photons as being so tiny that their number must be vast, the human observer can in fact detect light at a rate of as little as 10 photons/sec at certain wavelengths, and advanced sensors can detect single photons at a time. Although we would ideally like to measure the *average* number of photons reflected from the scene onto a particular CCD/CMOS element, if the number of photons actually captured during a given interval is relatively small then the number may deviate significantly from the average, leading to a noisy image. Noise is usually considered to be signal independent and becomes most apparent when the signal itself is relatively weak, for instance when taking photographs at night.

Perhaps the most effective way to remove noise from images, therefore, is to accumulate a single image during the course of a number of separate exposures, which generally causes the noise to average itself out while leaving the signal intact. However, one needs a reasonably large number of images to improve the signal to noise ratio, which may be computationally expensive. Furthermore the camera and scene need to remain static during the exposures to avoid smeary blurs appearing, which is not very often the case! Although some commercial vision systems use sophisticated hardware to achieve a desirable and consistent image quality, this cannot be expected for vision systems in general. Indeed, our own eyes are still able to recognise objects in poorly captured images; artificial vision systems should be capable of the same. Accordingly there is a number of software techniques designed for the purpose of active noise removal. Although these techniques cannot uncover information from the image that wasn't there to begin with, their purpose is to achieve a more consistent image quality which should make subsequent processing more robust.

Low-pass filtering, implemented using a Gaussian convolution mask, is a simple noise reduction technique. The mask re-computes each pixel's value as a weighted average of its original intensity and those of its neighbours. This reduces differences in intensity due to noise. However, legitimate differences in the signal level, such as those found at edges in the image, are also diminished causing a loss of definition. A better, non-linear technique instead chooses the *median* intensity of a pixel and its neighbours. As this technique redistributes existing intensities rather than creating new ones it is less susceptible to producing a blurring effect than the Gaussian filter and therefore maintains definition more effectively. Although median filtering requires more computation (to calculate the median it is necessary to sort the values), it is a preferred technique in computer vision applications, although it will fail if too many of the pixel's neighbours have also significantly deviated from their "true" values. The *mode* is perhaps the most useful average to take, since it selects the value with highest probability. Although the mode is poorly defined when choosing among a sparse distribution of data points, Davies [55] proposed a method to approximate the mode given the position of the median. A consequence of using the mode is that the image becomes slightly sharper, which indeed can be a desirable feature. Some results by these different techniques are shown in Figure 3.1; there are relatively few examples of GP being used explicitly for noise reduction so the figure also includes a result evolved by the author's software.

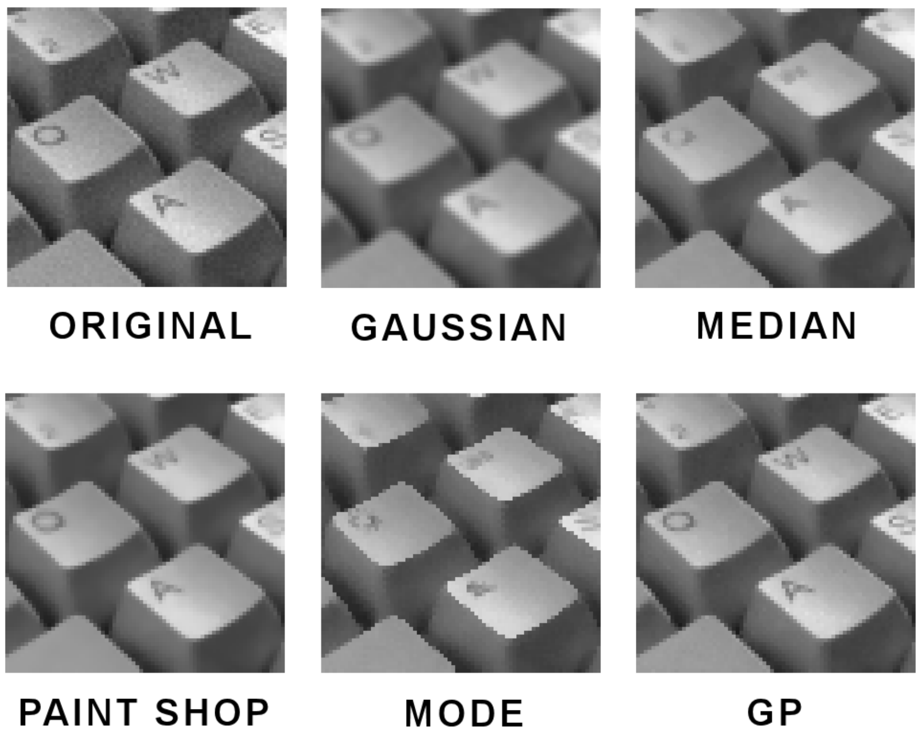


Figure 3.1: Different noise reduction techniques working on the original, noisy image. Gaussian filtering removes noise effectively but is too indiscriminate. Median filtering preserves more definition (although some is still lost). "Paint Shop" refers to the edge preserving smooth option in Corel Paint Shop Pro X2®. The author's implementation of a Mode noise filter causes a slight sharpening effect. Finally, the results from a noise filter evolved by the author's GP toolkit.

The drawback of all the above noise filtering techniques, whether linear or nonlinear, is that they apply the same procedure across the whole image, so there is always a chance that non-noisy pixels will also be disturbed. A sensible addition, therefore, is to integrate some form of *detection* which establishes whether a given pixel is noisy or not; if the pixel is noisy then it can be processed using an appropriate filtering technique, commonly the median filter; others are left untouched. This is often implemented using one or more thresholds to compare a pixel with its neighbours, such as the work by Chen *et al.* [56]. Petrovic and Crnojevic [57] used Genetic Programming to evolve a noise detector and stated that it yielded comparable or better peak-signal-to-noise ratios on test images when compared to a wide range of similar techniques.

Rather than evolving separate noise filters directly, other researchers have instead investigated the robustness of GP approaches on noisy data. The general consensus is that noise in training data can help the GP system to develop robust solutions, which otherwise may become over-fitted to unblemished data. Reynolds [58] injected noise into training data to investigate the idea for a robot obstacle avoidance task. He stated that this appeared to encourage the evolution of robust solutions, although the robustness of the individuals was not quantified on test data or otherwise. In a slightly more recent comparative work by Nopsuwanchai [59], the robustness of a robot controller was measured on a test set and compared between controllers trained on data with varying levels of perturbation. In one of the less surprising discoveries of recent times, it was shown that the controllers evolved with the highest level of perturbation were most robust. The key point is that Genetic Programming has the potential to incorporate noise handling into its choice and usage of features. Since noise does not always follow expected profiles, it is worthwhile using a learning system that can implicitly take observed noise characteristics into account.

Of course, noise is but one of many issues affecting the quality of captured images. Colour balance and exposure are examples of other image characteristics that current sensor hardware finds difficult to reproduce reliably. Exposure is limited by the dynamic range of current sensors, while colour balance is particularly troublesome owing to the subjective and relative nature of "colour" itself.

Ebner [27] used Genetic Programming to evolve an artificial retina capable of recognising the true colour of materials. His retina consisted of a 2D array of elements, each of which had red, green and blue sub-elements and could access the components in immediately adjacent neighbours. Each element corresponded to one pixel on the image, a similar approach to that used by neural networks which are also used in colour constancy. The colour of each pixel was computed by iterating an update process on the image 100 times, using update equations evolved by GP. Training was conducted on a series of Mondrian-like pictures[1]. Ebner reported good results in using his program to restore differently illuminated images to their original colours. In Chapter 6 we shall consider a few basic

[1]Mondrian's famous, "pure abstract" paintings often consisted of black lines and coloured rectangles, one of the most famous of which hangs in the Tate Gallery, London. On a pedantic note, the collages of coloured rectangles without black borders used by computer vision researchers often bear more similarity to some by Theo Van Doesburg, a peer of Mondrian's.

software approaches to colour constancy and exposure control, and see whether they can render the author's vision software more robust.

3.2 Low-Level Vision

For now, let us assume that the image has been acquired and is of adequate overall quality, and consider some of the fundamental ways by which machines may extract image content. The "content" of an image, however, is not easy to quantify. Any scene may contain hundreds of objects of all shapes, sizes, colours and textures. The difficulty is confounded by the sheer number of pixels within any image: thousands at the very least. A popular approach is to start by identifying those pixels that seem to indicate certain generic points of interest. Many of the early approaches, perhaps typified by the convolution with masks of the 1970s and early 1980s, did little more than emphasise features for subsequent human processing. They were later refined into techniques such as edge, corner and other interest point detectors which are more suited to further interrogation by machine. More recent detectors such as SIFT [60] and SURF [61], used to discover those points within an image which are most invariant to transformations, are direct descendants. In this section we shall consider various approaches designed to assist machines in identifying patterns that may be indicative of higher level content.

3.2.1 Edge Detection

It is often the case that different objects are constructed from different materials, or that their positions relative to fixed light sources will cause them to be subjected to different levels of illumination. Therefore, breaks in pixel intensities provide a useful cue for the detection of object boundaries and an essential first step for a variety of higher-level algorithms, for instance edge-based 3D template matching or some corner detectors. Accordingly *edge detection* is one of the most thoroughly researched areas in computer vision.

The purpose of edge detectors is to identify whether local changes in image intensity indicate the presence an edge – or not. Since any point where two neighbouring pixels have different values is a potential edge, the edge detector's decision is not straightforward; the process is rather subjective. Furthermore, there are various different "types" of edge (sharp edge, gradual edge, roof, trough etc), each of which have different intensity gradient profiles. It is difficult for one algorithm to be able to detect them all. Nonetheless, a wide variety of different approaches have been proposed.

An intuitive method of discovering edges is to compute a number of convolution masks that correspond to edges at different orientations and then choose the maximum response among them (see Wang [62]). However, this approach can only be rendered scale/rotation invariant by deploying a large number of masks. As a relatively "brute force" approach, it may be rather computationally expensive. Nevertheless, "template matching" is, in general, a flexible technique which has a variety of applications, such as character recognition, face detection or crater detection.

A more general approach to edge detection is to look at the second order derivative of the image; zero crossings in the local intensity gradient are indicative of edges. The second order derivative of the image may be approximated using the Laplacian convolution operator[2]:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

An early edge detector, invented by Marr and Hildreth [63], used Laplacian of Gaussian convolution in order to search for edges, a technique also used for blob detection. The Gaussian convolution is intended to remove noise that would otherwise yield false edges. As is apparent from the mask, the Laplacian operator responds highly to fine detail, so it is often used as a sharpening operator. Consequently the Marr Hildreth detector may be more adequately described as one that finds regions of high intensity contrast. This includes both edges and other features, giving rise to false edge responses.

A different approach aims to target edges more specifically. Differential gradient methods estimate the magnitude of edges perpendicular to the x and y axes using the first order derivative, which again may be approximated by convolution operators such as those of Prewitt ($m = 1$) or Sobel ($m = 2$):

$$s_x = \begin{bmatrix} -1 & -m & -1 \\ 0 & 0 & 0 \\ 1 & m & 1 \end{bmatrix} s_y = \begin{bmatrix} 1 & 0 & -1 \\ m & 0 & -m \\ 1 & 0 & -1 \end{bmatrix}$$

These masks approximate more succinctly the nature of an edge and therefore do not respond to high-contrast non-edges as strongly as the Laplacian operator. Each mask may be used to estimate the extent of the edge in each direction; the edge's *magnitude* may be calculated either using Pythagoras' theorem, or by the approximation

[2]The Laplacian response is believed to play a role in mammalian vision systems.

$$m = |s_x| + |s_y|$$

The magnitude of each edge candidate is then thresholded to determine the presence or absence of an edge. The Sobel operators are able to work reasonably well using small masks (which motivates the author to use them as a feature when using Genetic Programming). Given two perpendicular measurements, a little trigonometry can yield a rough estimate of the edge's direction. However, in comparison with the Laplacian operator, the differential gradient yields thicker edge responses.

The well-known algorithm by Canny [64] extends this approach with several stages of additional processing in order to ensure thin edge responses are obtained and to mitigate some of the problems associated with thresholding edge from non-edge. In Canny's algorithm, the image is first smoothed using a Gaussian convolution mask to remove noise that might otherwise cause false edge responses. The intensity gradient of the image is then measured in order to find candidate edges. A search for the local edge intensity maximum is used to thin each edge down to single-pixel thickness, so-called non-maximal suppression. The final identification of edges is performed by an algorithm called hysteresis, which makes use of two thresholds. Edges below the first threshold are flatly rejected; all edges above the second are accepted as *bona fide* edges. Those in between are only marked as edges if they connect two such genuine edges. This reduces the number of interrupted edges in the final image.

Other, more complex techniques include methods that identify edges by viewing the image at different scales. The algorithm by Bergolm [65] first blurs the image and locates intensity gradients in excess of a particular threshold. Progressively less blurred images are examined in order to localise the edge accurately. Rothwell *et al.* [66] proposed a few modifications to Canny's framework, doing away with the process of hysteresis and replacing it with an adaptive threshold, stating that the contrast of an edge should not be a primary indicator of its semantic usefulness. Numerous other approaches to edge detection continue to be published, although it becomes progressively more difficult to differentiate between them!

Harris and Buxton [67] used Genetic Programming to evolve functions which could identify the discontinuities in a one dimensional signal. Their work was inspired by the theoretical aspects of the work of Canny; they assessed the fitness of evolved individuals using Canny's suggestions that a good edge detector should have a good signal-to-noise-ratio, localise each edge accurately and respond only once per edge. Their evolved function performed better than Canny's on real world signals. They stated that GP permitted the discovery of near-optimal detectors for edges in signals which fell outside of the assumptions inherent in a theoretically "optimal" detector. The authors later extended their software to render it capable of working on real images [68] and once again showed that it could outperform theoretical operators in applied situations. They stated that it was better to tailor an edge detector for a particular "class" of images sharing similar noise characteristics than to use a generic edge detector. Given appropriate training data, techniques such as GP naturally permit the construction of tailored solutions easily and may provide an advantage in delivering robust solutions to work in particular applications. All the generic edge detectors are dependent to some degree on parameters (Canny's detector typically requires three) which have to be considered carefully. This raises once again an important motivation for the use of machine learning techniques in computer vision – data in some problem domains do not necessarily fit with assumptions made by some algorithms; a solution learned for a specific application can sometimes produce better results. Incidentally, later in this thesis we will show how GP can be used to evolve a detector for finding edges specific to a particular problem.

Chao [69] used a neural network to find edges in an image, using each pixel as an input for the net, with the weight of connections between pixels being determined by their difference in intensities. Following learning of the networks, the outputs on edges would tend towards a value of one; everywhere else to zero. As the net was essentially learning the weights for a convolution mask, results were found to be comparable with a Sobel operator. Another approach, this time using an evolutionary learning technique, was demonstrated by Bhandarkar [70] who used genetic algorithms. The individuals' chromosomes were laid out in 2D so that the GA could evolve "edge images" for locating edges in grey scale images: again the GA produced, in essence, an optimised convolution mask.

However, Srinivasan *et al.* [71] used neural networks to devise a two-stage edge detection approach that yielded a detector comparable with Canny's. Edge detection is, therefore, one instance where learned techniques may produce equivalent or better results than "standard" vision algorithms for certain domains of applicability. This is due to the inherent flexibility of machine learning techniques to tailor specific solutions to specific problems.

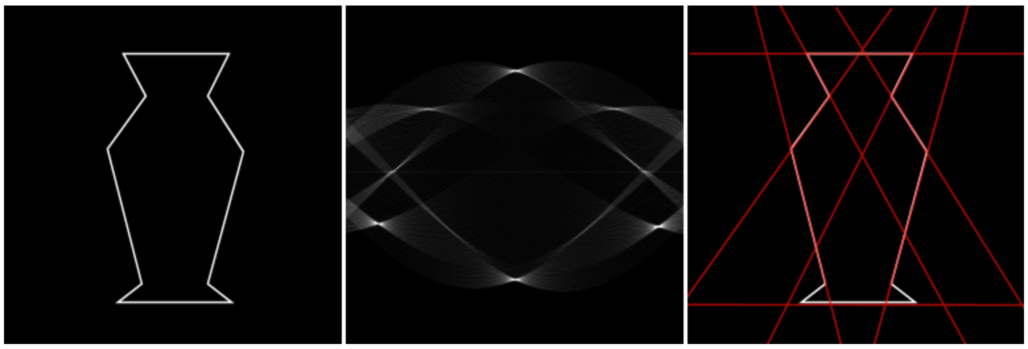


Figure 3.2: The Hough Transform. From left to right we see the original image (usually acquired following a process of edge detection), the contents of the accumulation array, and the lines discovered by the algorithm.

3.2.2 Line Detection

An edge detector should be able to determine whether a given pixel is an edge or not. If the detector is applied to the whole image, then the edge detector yields: another image. As we have seen, computers find it difficult to understand the context in images, so edge detection in itself is usually not enough to yield useful information. The long-established Hough transform is a useful technique for determining whether a given edge pixel is part of a larger linear edge, which itself may be more useful information. Depending on a threshold value, the Hough algorithm can be employed to detect lines of a particular minimum length. More usefully, the Hough transform can discover the coefficients that define the line, so the computer can gain access to more quantifiable information. If we choose to represent a line in terms of any x, y point's distance r from the origin, given an angle θ:

$$x \sin(\theta) + y \cos(\theta) = r$$

then for a set of x, y points all values of r can be calculated by iterating through all possible values of θ, or rather a discretised subset of θ's possible values. Plotting the values of r against θ yields a sinusoidal curve. If all the curves are accumulated on the same graph then the curves for a number of x, y points on one line will all cross at a single point. This point represents the common values of r and θ, which define the line. These points are identified by creating a two dimensional "accumulation" array. The Hough algorithm searches for peaks in the array to find all the lines in the image.

As the Hough transform is at heart a parameter discovery mechanism, other geometric primitives that can be parameterised, such as circles, ellipses and curves, can also be detected in images using a similar process. For these reasons, and because the algorithm is suitably rotation invariant, the Hough transform is used extensively in computer vision. It is a preferred technique for applications such as road detection from satellite images. When the points are taken following the tracking of particular objects, it can be used to detect (and thus predict) the direction of moving objects, for example the movement of cars on motorways or satellites passing through the night sky. The search can be made into a local one if we assume r to be zero and measure coordinates relative to a point of interest. This reduces the search to one dimension and can be performed quite quickly, making it a useful feature to be used by Genetic Programming.

3.2.3 Other Points of Interest

Besides edges and lines, there are many other useful areas of interest in a scene, for which other detectors have been developed. Some interest points are particularly generic, and may be discovered mainly according to their invariance to certain transformations, making them particularly useful for image matching. In popular use is the SIFT algorithm created by David Lowe [60]. SIFT works by locating a number of points in the image (not necessarily edges) using a difference-of-Gaussians (DoG) blob-finding technique. A "pyramid" of scaled DoGs is used to find those blobs that are scale invariant. Each key point is then defined by a feature vector descriptor which helps allow it to be uniquely and robustly matched. A later, slightly more robust, algorithm, SURF [61], is based loosely on SIFT, and implements sufficient performance enhancements in order to work in real time. SIFT and SURF features are detected reasonably consistently; they can be reliably identified between different frames, making them ideal features for image matching, panorama stitching, and 3D scene reconstruction. Robots using trinocular cameras can use common features in each image for distance estimation. SURF and SIFT are also suitable for object recognition if the descriptors can be matched to an item in a feature database.

Trujillo [72] used Genetic Programming to evolve similar features of interest. The fitness function for the features was primarily defined by the repeatability of the feature in successive frames. GP was able to evolve an improvement over a rotationally invariant operator proposed by Beaudet[73]; this is regarded as one instance where Genetic Programming was able to develop a human-competitive result.

We have already mentioned in passing that the Laplacian-of-Gaussian operation is quite good at detecting blobs, or circular points in a scene. Other interest points include ridges and corners; various algorithms have been proposed for the detection of each.

The intuitive Wang and Brady corner detector, for instance, tracks edges and detects the points at which the direction changes suddenly, indicating a corner. The SUSAN [74] family of algorithms is able to integrate both edge detection and corner detection into one approach which requires neither image differentiation nor noise reduction. The principle underlying SUSAN is to pass a circular window over the image, comparing the value of the central "nucleus" to other pixels within the window. The set of pixels with similar value to the nucleus is described as the USAN[3]. Different USAN sizes can be used for the detection of different features. A USAN which completely fills the circular area indicates the region is largely self-similar, so no edges or corners should be present. A USAN size of approximately half the region size indicates that the area is hovering over an edge, and smaller USAN sizes are more indicative of corners.

3.3 Segmentation

The concept of self-similarity used by the SUSAN detectors leads us towards a new topic, in which images are partitioned into regions based on their homogeneity. Given that many items are composed of a single material, the process of identifying self-similar regions or *segments* within images can play a crucial first step in object detection and recognition. Segmentation generally works by grouping pixels according some common characteristic, quite often based on grey-level intensity. Since the boundaries of segments implicitly determine edges (although perhaps less accurately), segmentation has the potential to kill two birds with one stone, as it were. Like edge detection, segmentation has received a great deal of attention from vision researchers over the years; a few of the more popular schools of thought are studied here.

One of the most straightforward approaches is to threshold each individual pixel by its intensity, yielding a binary image. Since the processing here is trivial, segmentation can readily be performed in real time. Discarding either the "dark" or "light" areas will yield the regions of interest, and is suitable for applications such as Optical Character Recognition (OCR) or various industrial inspection tasks. Although there is generally a substantial contrast between printed text and paper (in the case of OCR), gradual changes in illumination across the entire image usually prevent global thresholds from being suitable. More robust are so-called variable thresholding techniques which analyse local intensity histograms and choose automatically the threshold that divides the classes most completely, such as the technique by Otsu [75]. Ohlander [76] proposed a non-binary technique,

[3]or "Univalue Segment Assimilating Nucleus".

in which thresholds are chosen at the peak of intensity and colour histograms iteratively until no significant peaks remain. We shall consider a couple of automatic thresholding techniques, including Otsu's, in our discussion of classification in Chapter 4.

Such thresholding techniques are to a large extent dependent on the modality of the histogram as a result of the relative occurrences of the "object" and "background" classes. If a scene is dominated by the background class, then any peak in the histogram may be too small for histogram analysis techniques to detect it. Although techniques such as gradient relaxation (see Bhanu and Parvin [77]) attempt to take this into account, there are various other means too.

One approach is region growing. A set of "seed" regions are progressively dilated until the whole image is filled. An example is the *watershed algorithm* [78], for which there are many implementations. If one imagines the image as a height-map derived from pixels on grey-levels, "water" may be allowed to flood the terrain from the lowest points, or basins, until the entire image has been covered. The different regions are then defined by the boundaries between distinct water sources. The seed points in this case are selected automatically, but may also be chosen manually, or by some other *ad hoc* method.

Figure 3.3 shows three stages of a recursive watershed algorithm progressively discovering boundaries on a flag image. As can be seen, the algorithm is rather prone to over-segmentation, finding too many regions, although segments can be merged together at a later stage. Another technique applies hints or "markers" to indicate which parts of the image are definitely contiguous and should not be over-segmented. Of course, such markers must be manually drawn or discovered automatically using other approaches.

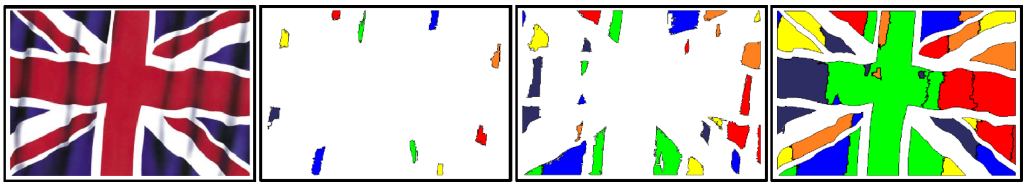


Figure 3.3: A basic watershed algorithm working on the image on the left.

There are several other techniques which are not mentioned here – segmentation, like edge detection, is not well-defined so attracts numerous different approaches, each with different scope and applicability. The problem is apparent within the Berkeley Segmentation Benchmark [79], a growing set of human-segmented images. Each image is segmented by several people – and no two human responses are the same! Indeed, the *evaluation* of segmentation algorithms is a field of study in its own right.

As evidenced by the above examples, the research within the mainstream computer vision community is often focused on unsupervised segmentation. However there are no automatic algorithms that are sufficiently general to work satisfactorily on all images. Once again this motivates the author towards supervised machine learning techniques. A nice semi-hybrid approach was proposed by Campbell [80] who used a segmentation technique that incorporated both supervised and unsupervised components. A Self-Organising Map (a type of unsupervised neural network) was used to produce the initial set of regions. The regions themselves were then classified into different categories using another neural network. This combines the advantages of unsupervised neural learning with the benefits of knowledge provided by domain-specific training data.

Other machine learning techniques, especially optimisation techniques such as Genetic Algorithms, have been used quite extensively in image segmentation. Genetic Algorithms may also be used to optimise parameters for many of the algorithms described above. Brumby *et al.* [81] introduced a different approach using Genetic Algorithms where each GA "gene" could invoke a different image operator primitive (included were logical, spatial, thresholding and spectral operators). The gene additionally specified certain parameters, and the input and output "scratch panes" which would be affected by the operation. Their approach essentially permitted the vision system to experiment with different image filters and operations as might a computer vision researcher using MatLab or other software.

Attempts at segmentation by Genetic Programming proceed along similar lines, in which the program more closely resembles a filter which transforms each individual pixel into a particular class. In fact the process bears most in common with classification; each pixel is described by certain features with programs evolved to identify whether or not the pixel's feature vector is representative of an "object". The description of a pixel in terms of multiple features is a distinct advantage. As one can see from the folds in the flag in Figure 3.3, a continuous region can have both dark and light areas – so in this case the *hue* of pixels would probably result in more accurate segmentation, although our perception of "colour" is itself rather subjective from a machine's point of view. The key is that intensity alone is not always enough information in order to make a judgement about a particular pixel.

Of course Genetic Programming is not alone in making use of pixel features beyond grey-value. Another family of algorithms cluster image pixels by their apparent similarity, a process which is less dependent upon the size of individual classes but instead the differences between different classes. Clustering may take various forms, including k-means-clustering, where clusters are formed by iteratively moving instances between cluster centroids. The major issue with k-means clustering techniques, however, is that the number of clusters must be defined *a priori*.

A different approach, hierarchical clustering, takes an initial set of clusters and progressively combines them together based on similarity. An approach by Chang and Li [82], named “Fast Adaptive Segmentation”, bears much in common with this technique, in which the image is partitioned into a large number of small areas. These are progressively combined together to grow larger and larger regions. Adjacent areas are permitted to merge if they are sufficiently similar, and provided that there isn’t a significant edge along their shared boundary. The accuracy of the eventual segments, however, is somewhat dependent on all previous decisions.

3.3.1 Applications and Image Features

We shall now look more closely at the efforts of genetic programmers to segment or otherwise detect objects in images. Of particular interest is the kind of image features used as pixel descriptors.

Poli [83] used Genetic Programming to evolve a custom filter capable of segmenting magnetic-resonance (MR) images of the human brain. Although his terminal set was comparatively modest, consisting of different scale image intensity averages, GP was able to invent filters that were marginally more successful those that of a neural network.

Roberts and Claridge [84] proposed a system for segmenting skin lesions, some of which are indicative of certain types of skin cancer, a problem made difficult by the natural variability in skin tone and texture. Their evolved programs were given access to a large range of image features, consisting of thresholds, morphological operators, logical operators, region statistics etc. One of the nice features of Genetic Programming is that the function set can be suffused with a reasonably large number of operators: GP’s selective nature will cause the appropriate ones to be chosen more readily. The evolved operator performed reasonably well, with a sensitivity of 97% and specificity of 81%.

Among the metrics that may be used to describe a pixel for the purposes of segmentation are descriptors of its surrounding texture. Texture is crucially important for the accurate segmentation of certain image types, including satellite, medical or multi-spectral images. Although any feature that combines the output of more than one pixel will encode some detail about the surrounding context, the computer vision literature suggests a number of specific approaches for describing texture.

A popular tool in texture analysis is the grey level co-occurrence matrix (GLCM, also referred to as a spatial dependence matrix, see Haralick [85]), which encodes the distribution of co-occurring pixel intensity values at different offsets from a central point for different levels of adjacency. A variety of different statistical features can then be calculated from the matrix. Song and Ciesielski [86] evolved binary texture classifiers that could distinguish a given black-and-white texture from a set of 47 others. Song and Ciesielski tried two function sets, one using 195 Haralick texture features and another using only the raw pixel data. They stated that GP was able to outperform the C4.5 Decision Tree algorithm marginally when differentiating each texture from 47 others using the Haralick function set, although both classifiers frequently attained 100% accuracy, on account of the training set's simplicity. More interesting was that evolved programs using the raw data set could still achieve $> 90\%$ accuracy, indicating that GP can construct useful statistics itself as well as glue existing ones together.

Laws [87] proposed a different algorithm by which textures could be described. He suggested using a series of 5-pixel, one-dimensional masks based on Gaussian, Gradient, Laplacian of Gaussian and Gabor features, intended to measure image level, edges, spots and ripples respectively. Different combinations of these were combined into 5×5 convolution masks that could be run on the image, with the expectation that among the different combinations would be one able to distinguish between different classes of texture. Although not particularly invariant, these can be useful descriptors of texture, and are ideal for use by Genetic Programs, although they don't appear to have been used by GP researchers before.

Quintana *et al.* [88] evolved programs to detect domain-specific areas of interest in images using Mathematical Morphology (MM) features[4]. One of their experiments concerned the extraction of different musical symbols from sheet music: the notes, hooks connecting quavers and clef lines. While the heads and lines could be isolated well, the GP method was unable to segment satisfactorily the clef lines from the musical notation. It may be that MM is perhaps not capable of removing large objects without eroding smaller objects such as the clef lines.

[4]Mathematical Morphology [89] is a long established technique used for various image processing tasks. MM considers sets of pixels, rather than the pixels themselves, and offers two basic operations whereby the set may be expanded outwards or contracted inwards through dilation or erosion respectively. Although Mathematical Morphology was initially conceived to work on binary images it was later extended to work on grey-scale images, permitting new applications, such as the watershed algorithm.

3.4 Object Detection

The reader may have noticed that the applications of GP described so far have drifted from image segmentation to the detection of features of an increasingly specific nature. In this chapter we have already seen some techniques concerned with the detection of particularly generic features such as circles, corners or blobs, but here we shall cover the detection of more particular patterns. We shall start with a discussion of object detection in mainstream computer vision, taking face detection as an example.

Despite the innate ability of humans to detect faces robustly and effortlessly, face detection is difficult to implement in silicon. Nonetheless, in recent years face recognition algorithms have become relatively ubiquitous, working in real time in small devices such as digital cameras. Since faces cannot be modelled with the mathematical precision of, say, a circular blob, machine learning techniques feature prominently in mainstream face detection algorithms and object detection in general.

Various strategies have been employed in order to generate face detection algorithms, including geometric approaches where the face is fitted to a 3D model, or more knowledge-based approach where the common features of human faces (eyes, mouth, nose) are identified and faces recognised according to the known relationships between them. This latter approach, while apparently logical, is rather dependent upon the assumption that the aforementioned features are indeed the most robust means of identifying human faces. Among the most influential papers on the area is the work by Pentland and Turk [90] who developed one of the first successful face detection systems. They used principal component analysis to construct useful face templates, so-called eigenfaces (some of which bear little resemblance to our concept of a face!). Weighted sums of various eigenfaces can be used to represent a variety of actual faces. Since faces are matched based on intensity, face recognition algorithms using this technique can be quite sensitive to changes in the position of the illuminant and are not necessarily invariant to changes in rotation.

While a common strategy for making detection scale or rotation invariant is to use different sized templates, or to rotate the image itself through various angles, Rowley *et al.* [91] used a neural network to develop a face detection system that implemented rotation invariance in a more efficient manner. They first trained a detector to identify only the angle of faces within a given window. As each angle was identified, the window was then "de-rotated"; a second, upright-only face detector then decided whether a face was present or not. The detector was trained on an initially small training set, with additional "false" images added during the learning process to reduce the likelihood of the net yielding excessive false positive results. One of the big advantages of neural networks is that it can be incrementally trained using this kind of "bootstrap" approach. An issue in face detection, indeed all detection tasks, is that there are many more examples of non-faces than there are examples of faces: the training data is always insufficient to some degree. The problem with face-detection algorithms, therefore, is not the sensitivity but the specificity – the ability of the algorithm to work without producing too many false positives. Rowley suggested using multiple nets and voting schemes to develop a consensus that may suppress some of these errors, however they noticed this was sometimes at the expense of sensitivity. Such schemes may also fail because there is no guarantee that separately learned algorithms will not make the same mistakes.

One technique used to develop a solution to these competing criteria is to use a boosting framework, such as the AdaBoost algorithm by Freund and Schapire [92], which develops a more accurate classifier from several sub classifiers, each of which is trained on the same training data, whose weights are adjusted between learning runs. A weight is assigned to each item of training data, which is iteratively adjusted after the learning of each sub classifier in order to encourage high levels of accuracy and to ensure that each classifier is different. Viola and Jones [3] made use of AdaBoost in the development of their own face detection technique to combine various such "weak learners". The weak learners themselves were Haar-like features, defined by parameters discovered by a Genetic Algorithm. Haar-like features can benefit from the use of an "integral image" which permits the calculation of mean and standard deviations of arbitrary image areas in constant time. Viola and Jones showed that their approach was slightly more robust than that of Rowley.

Winkeler and Manjunath [93] demonstrated the use of Genetic Programming for face detection using a multiple-scale windowed approach. Within each window, fifty-two pixel-based features were made available to the GP system, as well as Gabor features and other descriptors developed by mainstream computer vision. The authors suggested that island selection would avoid premature convergence, a problem found in many machine learning techniques. Code bloat is another common problem in Genetic Programming; it is worth mentioning it here because image processing in particular demands that algorithms are reasonably parsimonious, and it would be unfair to paint Genetic Programming in an exclusively positive light! One of Winkeler and Manjunath's evolved solutions was composed of over 3000 nodes: a severe performance bottleneck. In a second experiment, a program was evolved to distinguish roughly areas that contained faces from areas which did not. When combined into a multi-stage approach, the processing cost was reduced by 75% and the false negative rate was also cut significantly (although the effect on the detection rate was not reported). The reader should be reassured, however, that it is probably possible to evolve a reasonable classifier of a much smaller size. Various means by which code bloat in GP programs can be curtailed shall be discussed throughout the course of this thesis.

We shall now move away from face detection in order to cover some of the other ways in which Genetic Programming has been applied to object detection in general. Tackett [94], who has the distinction of publishing the first significant GP paper concerned with computer vision in 1993, evolved an automatic target recognition system for recognising targets in military situations[5]. He used straightforward spatial features consisting of areas' intensity means and standard deviations to evolve tank detectors. Tackett compared his detectors with an MLP neural network and found that GP delivered the more accurate solution. The creative freedom afforded by the fitness function paradigm in GP makes it ideal for solving difficult problems. Given a clear set of objectives, Tackett knew his program needed a detection rate of only 96% and adjusted his fitness function to deliver no extra reward to individuals whose accuracy exceeded that value, causing the system to favour more specific individuals instead. The MLP neural network, whose learning algorithm aspired to solve 100% of the "true" training data samples, was found to accommodate difficult samples at the expense of lower specificity. As Tackett observed, this caused an unacceptable level of false alarms for the neural network.

Roberts and Howard [95, 96], used Genetic Programming to evolve detectors for objects in complex environments – vehicles and ships in satellite images. The authors used single-pixel-thickness circular statistics as features which are reasonably invariant against rotation and noise. Four circles of different diameters were used, with the average intensity, standard deviation, edge count and an edge distribution measure calculated for each. Again, the authors encountered problems with false positives, and solved the problem in a similar manner to Winkeler and Manjunath: a "rough" preliminary detector was followed by finer, second stage detectors. The authors used a bias ratio in their fitness function (see page 19) to subjectively determine the sensitivity/specificity ratio that would determine whether to evolve a rough or fine detector. Roberts and Howard did not expect each secondary detector to provide a perfect detection rate, but rather suggested that different second stage detectors might be good at detecting different kinds of vehicle, although this was not presented in evidence. The evolved detector was able to detect 89% of the

[5]Despite the processing burden imposed by images in general and GP learning in particular, Tackett trained his system on 2000 samples, an impressive feat at the time!

vehicles with a 14% false positive rate. However there was no testing on unseen training data, and relatively few images were trained.

Other GP researchers have also used a variety of other basic features for vision tasks. Roberts and Claridge [84] used GP to develop detectors using a generous selection of different features including statistics about rectangular, circular and ring shaped windows, convolved results, differences between 3×3 areas and various other features. Combinations of these basic features can produce higher level operators capable of solving more subtle tasks, with the essential goal being to find features specific to particular problem domains, including shape recognition and pasta detection (a perennial favourite). In later work [97], the authors went on to describe how Genetic Programming may construct and use imaging features simultaneously using a process of co-evolution, where the operators and GP programs using them are evolved simultaneously and cooperatively. The results published appear to show that co-evolution in this manner can solve problems to a high degree of accuracy, although unfortunately no comparison with "standard" GP was provided.

Although detection has its own specific issues (we've already seen the sensitivity specificity tradeoff), it may be seen as the binary case of classification or recognition in general. In computer vision classification is dominated by machine learning techniques, including support vector machines (SVMs), decision tree builders, and neural networks. We shall look at classification in depth in chapters 4 and 5. For now, we shall briefly look at multi-class object detection/classification by GP.

Zhang and Ciesielski [98] investigated the detection and classification of multiple kinds of object. They conducted several experiments with different image types: square/-circle classification, coin classification, and haemorrhage/aneurism detection in retinal images. Detection proceeded using a windowing approach, using various circular and rectangular statistics. Post-processing was deployed to ensure that only the centre of each object was detected once and only once. Good detection rates on the (difficult) retinal images were achieved but at the expense of relatively high positive alarm rates, despite the post-processing.

Zhang, Andreae and Pritchard [99] went on to explore the usefulness of positional pixel statistics. In an experiment using moving windows to detect basic shapes and different coins, they investigated the effects of just using statistics of the whole window (mean, variance and moments) in comparison to using just the mean in sub-windows. In another of the less surprising discoveries of recent times, they found that whole window statistics were unable to deliver sufficient information about shape, and the programs where sub-window statistics were made available were able to produce better results more rapidly.

Any discussion of multiple-class object recognition would be remiss if it were to omit a discussion of optical character recognition (OCR) which, like barcode readers, has been around for a surprisingly long time[6]. Accurate recognition of characters is largely regarded as a solved problem, at least for printed Latin-based alphabets. Nonetheless, it is by no means a straightforward task; the offline recognition of handwritten alphanumeric characters or indeed character-based written languages remains challenging.

Like face detection, character recognition is dominated by machine learning techniques. Commercial OCR systems commonly use neural networks for recognition, since they are able incorporate new knowledge during application. The key to most OCR systems is the choice of features, which have a substantial impact on robustness. Assuming that a single character can be detected and cut out from the image, it can be scaled to a fixed size. Each pixel within the scaled image can then be used as input for the neural network. This was the approach taken by Koza himself in GP, and later by Spivak [100], who used binary features ("is the pixel at i, j black?") and a standard set of functions to develop classifiers for the hexadecimal characters (0–9,a–f). Classifiers were trained on one font and then tested on others. The problem with such an approach is that the binary features are not particularly robust, indeed the decision of some of the programs was based on the values of as few as four pixels – classification may proceed based on "coindidence" detection rather than semantic understanding.

Andre [101] made use of a more robust feature set in which the character is processed to discover its boundaries. The shape is divided into four segments, with bounding boxes measured for each one, as well as for the character as a whole. Andre showed that his system could distinguish the letter 'C' from all others on about 96% of occasions. He also proposed a clunky method by which a hand-written algorithm could be converted into a GP tree by hand, and then evolved further to take into account new training samples. The author acknowledges that neural networks can adapt to new situations in an altogether more elegant manner.

A slightly more serious attempt at OCR was undertaken by Teredesai *et al.* [102] in order to recognise the characters 0–9 from the NIST handwritten digit set. Rather than deploying a so-called "One Model Fits All" approach, they used a hierarchical feature set starting from coarse overall features but progressing to increasingly finer distinctions within sub-regions of the character image. Nine features were extracted at each stage,

[6]OCR has been used in British post sorting offices since 1965.

four based on gradient and five based on moments. The authors found that the best individuals tended to use features at a range of different depths in order to make accurate distinctions. A binary classifier was evolved for each digit, results between 95.9% and 97.6% were reported. The authors did not report the overall accuracy of the multi-class classifier, so its overall capabilities could not be assessed.

There are many applications of binary shape analysis beyond character recognition. Johnson *et al.*[103] used GP to locate features *within* shapes. For part of an interactive system, they evolved a program which could locate the positions of the right and left hands of a silhouette of a person (taken against a blue screen background). Instead of using pixel statistics, they used a basic “point” data type which could hold x, y coordinates. Various functions were provided to find important points around the silhouette, such as the corners of the bounding box, the centroid and the right-most point, and other functions could perform point operations, such as adding them together or finding the point between them. Their results were quite successful as their best program could find 93% of left hands, although their sinister programs didn’t perform so well on validation data.

GP is otherwise rarely used for shape classification in images, despite a significant body of work performed in mainstream computer vision. Over the years countless descriptors for shape have been proposed. The most straightforward descriptor of a segment or shape is its size. Morphological erosion followed by dilation may be used to discard small shapes that do not convey much meaningful context. This is a somewhat computationally inefficient means of filtering shapes by size, but can also be used to smooth the outlines of the larger shapes, rendering them more suitable for further processing.

It is useful to identify the perimeter of a shape by tracking its edges. From the perimeter, corners and other features may be identified[7]. Other simple features can detect the roundness or rectangularity of the shape, and its aspect ratio and orientation. Other algorithms label the shape according to each pixel’s distance from the edge providing descriptors that reveal whether a shape is rounded or mostly thin. Conversely, shapes may be “skeletonised”, or progressively thinned to the point where the shape resembles a tree. The number of nodes, connections and leaves in the tree can then be used as descriptors of its structure.

One family of shape descriptors use statistical moments [104] which are traditionally used to describe the shape of probability distributions. Common statistics include the mean, variance, skew and kurtosis. Another popular approach is to compute a complex

[7]The author spent an enjoyable month crafting algorithms for these ideas in his first year of study, although it was mainly a process of wheel reinvention!

hull, which wraps around the shape as would an elastic band. From the hull can be calculated so-called concavity trees (Sklansky [105]) which store information about the hierarchy of convex areas of the shape, and unlike the examples seen so far can be used to reconstruct the shape completely. Metadata regarding the tree structure itself may be used as descriptors for object recognition, or different concavity trees may be matched more directly in order to compare two shapes (see Badawyl and Kamel, [106]. The advantage of many of these features is that they are rotation and scale invariant.

Montes and Wyatt [107] used graph-based Genetic Programming to locate of the centre of mass of an object. While not a particularly difficult problem to solve in practice (given that the object had already been segmented), it was shown that GP could evolve its own, reasonably accurate solution using only a single imaging operator (the mean intensity of a square region).

There are myriad other ways of extracting image features: the discussion so far only scratches the surface. More features will be described in Chapter 6 when the author comes to apply them for the purposes of generic feature detection.

3.5 Vision Systems

As we approach the end of this chapter we shall turn our attention from isolated vision components to vision *systems*, which perform as many levels of processing as necessary as to produce high-level output from images. As we have seen so far, Genetic Programming has been applied to a number of fields of endeavour within the computer vision domain; solutions have been evolved with varying degrees of accuracy and robustness. Here is a brief summary of the evolved vision algorithms covered so far:

Task	Examples
Preprocessing	Noise Suppression [57]
	Edge Detection [68]
Feature Detection	Texture [86]
	Musical Notation [88]
	Interest Points [72]
Segmentation	Multi-spectral Images
	Medical Imaging [84, 83]
Object Detection	Faces [93]
	Vehicles [94, 95]
	Hands [103]
	Robot Vision [58, 59]
Object Classification	OCR [100, 101, 102]
	Shapes/Coins [99]

A common feature in all of these papers is the rather specific nature of the problems being solved. There is, of course, nothing inherently wrong with this, although to concentrate on a single domain diminishes one of the key advantages machine learning techniques offer over the development of software by hand — the potential to learn solutions to more than one kind of task and thus save time. If one spends as much time tuning and preparing the machine learning technique as it would take to solve the problem using traditional computer vision methods, then the research has little usefulness beyond satisfying a particular curiosity, especially if the problem in question is one that has already been solved to a satisfactory degree. However, if one can "generify" the learning system in such a way that two or more problems can be solved, then one has a system with some added value.

The GAMERA framework [108] is a programming interface designed for the production of document analysis systems in a general sense, and has been used for a variety of applications involving the processing of historical documents and recognition of other printed characters. The framework does not imply the use of a particular learning algorithm, but facilitates the production of domain-specific vision systems such that they can be generated more quickly.

Later in this thesis, the author will present a software interface designed to provide a unified interface for the production of vision system training data.

The definition of "vision system" is rather loose, not least because the architecture of such systems is dependent on the application. Indeed, many authors describe their work as a vision system when it is perhaps more appropriate to call it an image processing component. For the purposes of this work the author defines a vision system as follows:

> "A system which takes unprocessed images as input, then processes them in such a way as to return high-level output without requiring further user interaction."

There are many examples of working vision systems, both within and without the confines of purely academic research. Machine vision systems are often used for the purposes of industrial inspection, for instance the inspection of products or foodstuffs on conveyor belts, for detecting or counting items, or defect detection, such as cracks in railway lines or aviation components. Security applications include the detection of aircraft, of people or unusual behaviour, or indeed face recognition. Consumer level applications include gesture recognition for human computer interaction or augmented reality applications. Applied vision is commonly used in academic research as part of robotics projects. The iCub [109], currently the most advanced humanoid robot in Europe, makes use of a number of different vision algorithms, including saliency detection, object recognition and online learning.

In practice most vision systems will deploy several of the stages of processing covered in this chapter. Given an image capture device, the vision system will go about acquiring an image, pre-processing it, performing low-level vision tasks to emphasise the objects or areas of interest, followed by higher-level processing, often classification. Indeed, there may be several further stages not covered in this brief overview.

In this chapter we have seen most of the stages necessary to go about creating a reasonably complete OCR application. Other systems, such as automatic number plate recognition, which is essentially OCR in a less constrained environment, require additional stages of detection and greater invariance to transformation. The key point in any such system is that the human should not have to perform any part of the processing manually, such as telling the system the location of the number plates or manually segmenting individual characters.

All of the GP research described so far has considered evolved vision in terms of a single stage of processing within a larger context, implying further stages that must take place before or after the evolved component's execution. For instance, research on OCR assumes in each case that the characters have already been cut-out from the image, binarised and scaled before the GP operator performs its task. Likewise the skin lesion research performs segmentation on individual images but does not suggest what to do next. In each case these papers relate to the proof of particular concepts rather than working on producing something complete in itself. The author has not encountered any papers which use more than one evolved stage in order to develop a “vision system”. To put multiple stages of evolved vision together would validate such components in a more complete sense – it is, of course, worthwhile to present quantitative results regarding the quality of a given component, but its utility is best tested when its output is used as input to another stage of processing.

We have seen that some Genetic Programming researchers do occasionally employ a multi-stage approach, but this is usually restricted to two or more stages of processing by programs that each perform the same task, though they may have been evolved using different fitness functions. A common example is the evolution first of a “rough” detector to discard as much of the background as quickly as possible, followed by a secondary detector to make finer distinctions.

3.6 Conclusions

This chapter has described a number of different tasks in the field of computer vision. There exist a wide range of approaches designed to tackle each, although it is not always straightforward to identify the most appropriate algorithm for a particular task. Usually the choice of appropriate techniques and operators is undertaken by computer vision experts. We’ve also seen various examples where machine learning techniques have been used to “glue” certain imaging operators together in order to generate task specific features that can’t readily be modeled mathematically. Some interesting solutions have been evolved by GP researchers. A common thread running through many of the tasks is the need to take decisions: whether a pixel is noisy or not, whether it sits on an edge or corner, is within a particular region, or is part of a larger feature. The first step in the author’s framework, therefore, is to investigate means by which Genetic Programming can be applied to making decisions in a general sense, which is the topic of the next chapter.

Chapter 4

Classification by Genetic Programming

Computers are excellent at crunching numbers and can do so substantially faster than we can. Accordingly they are also very good at taking measurements, for instance from images. After taking these measurements, however, the machine becomes rather less useful: commanding a computer to perform more subjective tasks, such as recognising patterns, is less straightforward. Of course we humans have no difficulty in recognising the patterns, shapes and objects in the world around us – it is something that our human brains can perform astonishingly well. If challenged, most people would find it extremely difficult to objectify how or why certain things are so instantly understandable. It is so difficult, in fact, to describe why our brain works the way it does that a word was invented for it: "intuition", which in turn is difficult to define! Although intuition will probably remain a trait reserved for living organisms for some time to come, in this chapter we shall discuss how machines may learn to make subjective decisions based on objective information.

Classification is the process by which a series of measurements may be transformed into a decision regarding the object from which the measurements were taken. Decision making in general, and classification in particular, is not a topic confined to computer vision. Technology now permits information to be collected and stored faster than ever before, so finding algorithms to make sense of the piles of data we've amassed has been of interest to data-miners and information theorists for some time. Many problems in computer vision too, such as detection, recognition and segmentation, may be dependent upon the classification or matching of patterns. In later chapters we shall consider exactly how one might take useful measurements from an image, but here we shall confine ourselves to making decisions based on arbitrary sets of data. We shall study the ways in which Genetic Programming can be employed to evolve classifiers and investigate means to make the process more accurate and generic.

We shall start by briefly considering the definition of a classifier. Almost any "object" can be expressed in terms of its measurable characteristics. If the object is an animal, for instance, we might describe it in terms of its pigmentation, number of legs, eyes or relative size. If we take n measurements from each animal, then each individual beast may be represented as a tuple $\{x_0, x_1, \ldots, x_{n-1}\}$. A classifier may be defined as any function that takes such an n-dimensional feature vector as input and describes it using a one-dimensional class label. In general, classification is about discovering groups and patterns in data and the boundaries that define them.

As we will see in Chapter 5, there is a whole range of different classification algorithms, since satisfying the above definition of a classifier is an ill-defined problem. There are various parametric and non-parametric algorithms developed for the purposes of classification, where parametric methods are typically linear transformations and non-parametric methods generally permit non-linear processing. The latter group of techniques includes decision tree builders, neural networks and support vector machines (SVMs). Decision tree builders work by iteratively partitioning the feature space into progressively smaller, more uniform portions that can each be allocated a particular class. Trained neural networks are weighted in such a way that particular combinations of features will lead to certain output neurons. SVMs aim to identify the decision plane that best separates two classes. We shall discuss the pros and cons of each in the next chapter, but first we shall consider the various means by which Genetic Programming can be applied to classification.

4.1 Representing Classifiers in GP

Genetic Programming is an abstract means by which to go about automatic programming. As such its use does not imply any particular approach towards the task of classification any more than would a standard programming language. In general terms, the GP system is used to learn the relationship between a set of inputs and expected output(s). The reader will recall that we can insert the inputs via *terminals* at the bottom of the GP tree; processing takes place further up with a final, usually numeric, output expressed from the root node. Unlike other learning algorithms, such as multi-layer perceptron neural networks, whose outputs map to specific classes almost by definition, it is necessary to embellish our genetic programs with certain operators or other additions to ensure the process yields suitable classifiers. While some of these additions may add preconceptions or assumptions of their own, the purpose is to interpret or translate the logic encoded within the genetic program into the all-important class label. There are various means by which GP can be used to develop classifiers, a few of which are summarised below.

4.1.1 Evolving Decision Trees

Many school children will be familiar with classification in the form of dichotomous decision trees, which allow non-experts to identify particular species or types by answering a series of straightforward "yes or no" questions: a favourite in biology field trips. This sort of approach is one of the most intuitive means of making a classification, not least because it can actually be written down on paper! Accordingly there is a number of decision tree algorithms, which aim to develop classification keys automatically. Algorithms such as such as the C4.5 algorithm by Quinlan [110] (and previously ID3 [111] and CART [112]) have been used for classification for some time. The goal, to divide the feature space into a series of increasingly homogenous areas, is implemented by representing a hierarchy of decisions in a tree structure. As a non-parametric method, decision trees can develop arbitrarily complex models to fit data and thus can handle multi-class problems with relative ease.

Since Genetic Programming is at heart a tree building and optimisation technique, it is also well suited for developing decision trees, albeit in a more *ad hoc* manner. Still, a little work must be undertaken first in order to render GP capable of producing such trees. The key function node is an $IF(\cdots)$ statement, which permits conditional logic. As we saw in Chapter 2, it is useful to implement strong typing to help ensure the conditions make semantic sense. It is also necessary to develop a new type of terminal node to act as a "return" function, returning the decision up to the top of the tree. An example of such a tree is displayed in Figure 4.1.

Bot and Langdon [113] used GP to evolve decision trees capable of solving a variety of different problems, using a method similar to that described above. They were able to use their system to solve some multi-class problems involving up to seven different classes. Bot and Langdon's evolved solutions were not always competitive with specialist decision tree algorithms. Why might this be? The principal problem here is the mixing of genotype and phenotype, of structure and behaviour: since the tree is responsible for both the logic and its translation into class outputs, there is a high degree of dependency between

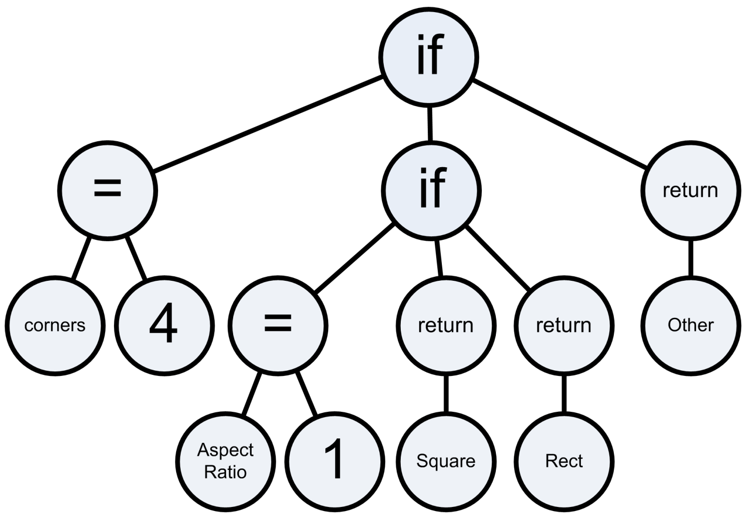


Figure 4.1: An Example of a Decision Tree Classifier for Distinguishing Shapes.

parts of the tree. For a branch to work properly, it must not only include the right logic but also terminate with the correct return class. Disruption to either aspect may cause the tree to stop functioning, so the crossover operator has even more scope for mischief! Because of this dependency, making incremental improvements without breaking existing functions becomes difficult. The author's classification system will be compared to Bot and Langdon's results in Chapter 5.

4.1.2 $f(X)$ Representations

An alternative approach is to have the GP process compute a value from some or all of the features, then interpret the meaning of the value afterward. Indeed, the learning process can be made more straightforward if the interpretation is performed by a separate algorithm, leaving Genetic Programming to learn only the discriminant function: the logic that underpins the eventual decision. The evolved function generally takes inputs $X = \{x_0, x_1, \ldots, x_n\}$, processes them in some way then returns a single, one-dimensional numeric output:

$$output = f(X)$$

As a minimum, the function set is composed of basic arithmetic operators. Many GP researchers leave it at that. Other operators, for instance logical operators, trigonometric functions or statistical methods can also be added to render GP capable of evolving more complex non-linear transformations.

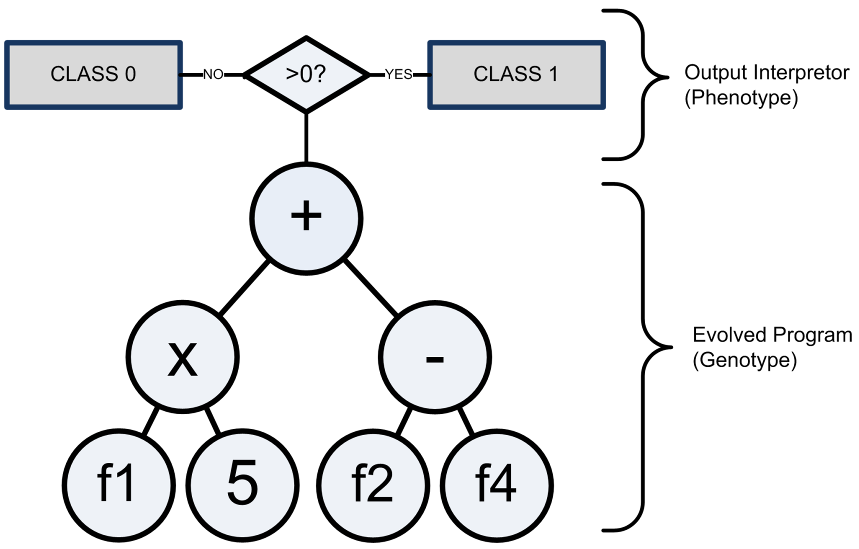


Figure 4.2: The $f(x)$ Classifier Representation, producing a floating output which is translated by a threshold operator

The simplest interpretation of the output's meaning is achieved using a threshold, which yields a binary classifier (see Figure 4.2). Placing the threshold at 0 is a tidy choice: all negative outputs are attributed to one class and all zero-valued or positive outputs are attributed to the other; the number space is equally divided. Over the course of the evolution, the programs should become better and better at producing meaningful output that straddles the zero origin. A disadvantage of this approach is that it is limited to binary discrimination between exactly two classes. While binary classification is sufficient for certain tasks, such as detection algorithms, we shall see later in this thesis various examples of classification problems which involve more than two classes. By definition, these problems require more than one threshold; yet there is no longer a set of thresholds that divides the number space equally.

The key problem with this technique, however, is that it doesn't fully disassociate the interpretation of a class from the logic. $f()$ still needs to ensure that its outputs fall around the zero-crossing in order to attain a reasonable fitness score. Although this kind of representation is quite restrictive, it is nonetheless often used in GP classification.

4.1.3 Binary Decomposition

Having mentioned *intuition* at the very start of this chapter, one may take inspiration from human approaches to problem solving. One natural, intuitive, strategy is to break up problems into smaller, more straightforward pieces. We've already seen how this is done using decision trees. If the training data are composed of examples from more than two classes, then it may be more reasonable to learn a solution for each class. The individual solutions can be chained together at the end to produce software that can work for all classes. If the representation is limited to binary decisions, then this is the only approach to generate multi-class classifiers.

There are two ways by which a multi-class problem can be decomposed into smaller pieces. The first may be referred to as "complete", or "one-vs-one" binary decomposition in which a classifier is created to distinguish every class from each other class on an individual basis, breaking the problem down into small pieces. For an n class problem the complexity is $O(n^2)$, which soon becomes a limitation: it may take substantially longer to evolve the solution and the classifier may take longer to execute. The second is by class ("one-vs-others"), where a classifier is evolved to distinguish each class from all others; an n class problem therefore has complexity $O(n)$.

Binary decomposition is widely used in the pattern recognition community. It has also been used by GP researchers, such as Spivak [100] who used GP to develop solutions to a character recognition task.

Since each class now requires a separate process of evolution, the total learning time may well be longer. Smart and Zhang [114] proposed communal binary decomposition in which a set of binary classifiers is evolved at the same time, allaying some of the speed problems of binary decomposition. They reported some success after testing their system on problems consisting of four classes (which is reasonably modest compared to some of the problems considered in later chapters). Regardless of the evolution time, however, the most important consideration is whether binary decomposition can yield more effective classifiers. The author shall present some empirical investigations into binary decomposition later in this chapter. For now, we shall continue looking at different classifier representations in GP.

4.1.4 One Individual, Multiple Trees

We have seen two techniques so far. In the first, decision trees, it is difficult to ensure that each branch of the tree's logic associates itself with the appropriate return class. This issue does not apply to the second approach, binary classification. The latter is, of course, limited to two-class problems unless one splits up the problem into pieces.

A compromise was suggested by Muni *et al.* [115], who proposed the evolution of a single individual that incorporated several sub-trees, one allocated to each class. The process of binary decomposition could therefore be attempted during the course of a single GP run. In order to ensure that most attention was paid to the weakest parts, Muni suggested applying the Genetic Operators probabilistically based on the *unfitness* of certain trees, aiming to improve poor performers while protecting good solutions. Other post processing methods (OR-ing) and heuristic rules for conflict resolution were used to boost the classifiers effectiveness. His results were broadly competitive with a number of other machine learning methods on benchmark tests (IRIS, WBC, BUPA, Vehicle, RS-Data), which tend to be dominated by non-GP learning techniques. Muni's trees used a zero threshold to determine their response, so his representation was still limited by an incomplete separation of genotype and phenotype. Muni's results shall be compared with the author's in the next chapter.

4.1.5 Modi Program Structure

Zhang and Zhang [116] proposed a different ("modi") program structure , in which any node may affect a vector of different outputs, each corresponding to a class. Each node in the tree is processed as usual, but if bound to a particular class, it passes its value to the vector upon execution. The class with the highest value assigned to it after the tree is executed is determined to be the chosen class in winner-takes-all fashion.

The modi approach permits a single tree to behave as though it were several separate trees, although the modi sub-trees may not be independent of each other. One advantage is that branches further down the tree can work as completely separate units, or building blocks, for a particular class and thus may not be so disrupted by the genetic operators. The modi method was able to make modest improvements over "standard" GP[1], increasing performance by up to 11% on a small set of datasets. However, the effectiveness of each sub-tree is dependent on which class it chooses to contribute towards. As the number of classes increases, so too does the chance that the node will choose the wrong class! The modi approach again does not permit the logic and the translation to be truly disassociated.

4.1.6 Range Selection

Arguably none of the techniques so far have completely separated the individual's genotype from the phenotype, the logic from its interpretation. Loveard [117] suggested interpretation using a program classification map (PCM), which divides the number range into

[1]The authors did not, however, define what "standard" GP was.

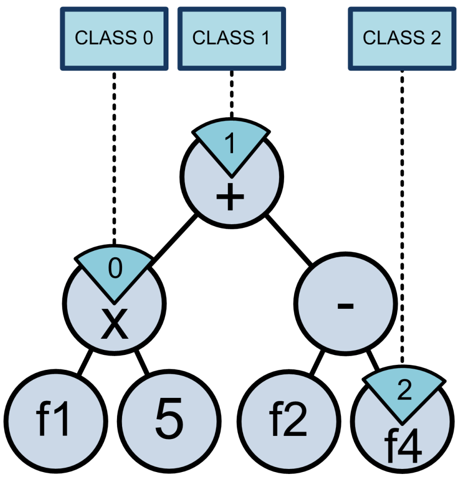


Figure 4.3: A Modi representation. Some nodes can affect an output vector from which the class is then chosen using a winner-takes-all strategy.

a series of slots, each of which is assigned a particular class (see Figure 4.4). The output from the individual is matched to the relevant slot and the slot's class is returned.

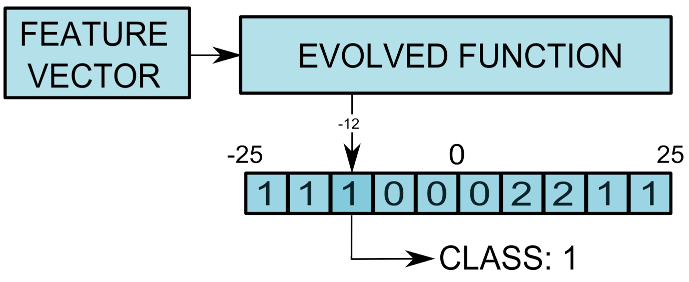


Figure 4.4: A Program Classification Map divides a number range into a series of slots and associates a class with each slot. The class ID associated with each slot may be predefined (static range selection), or discovered at run-time for each individual (dynamic range selection).

The class labels associated with each slot may be assigned class labels before evolution begins (*Static Range Selection*), in which case each class is usually allocated an equal number of slots. This has the significant disadvantage that the classes are arbitrarily ordered and the ranges chosen may not suit the clustering of different classes. An improved technique is to choose the class ID for each slot *at runtime* to fit with each individual, referred to by the authors as *Dynamic Range Selection* (DRS). Classes are allocated as follows:

1. For each slot i

2. Find the tuples of training data M which, following processing by the GP program,

are allocated to slot i.

3. Among those tuples in M, select the most common class label and set this as the class label of slot i.

The advantage of Dynamic Range Selection is that the GP programs can produce output within a larger range and are not as restricted to arbitrary thresholds or orderings. Appropriate thresholds are automatically allocated after the individual has been developed, so the GP system does not have to evolve a specific translation component, beyond ensuring the output is within the range of values.

Among the design choices is how to set the thresholds that define the range and the slots within. Usually the map occupies a range of values, say from $-n$ to n, with s slots, such that each slot occupies a range of size $2n/s$. The authors appear to have chosen the values of n and s relatively arbitrarily. They stated that the output from the GP program should be rounded to the nearest integer and that the map should occupy the range -250 to $+250$, creating 501 separate slots.

The process of determining class memberships for the map on a per-individual basis does impose additional processing and memory requirements. This second learning step may once again be thought of as analagous to the difference between genotype (the DNA of an individual) and phenotype (its eventual behaviour) in natural life. To put it another way: although the purpose of our genes is to build a body and a brain, the learning of language, or how to play chess for instance, is not hard-wired – it comes through practice during the individual's lifetime. As was mentioned in Chapter 2, this distinction is not very often apparent in GP, which tends to concentrate on the genotype. The second step, of learning by experience, is often missed out – this may account for why GP programs do not always perform as well as other algorithms; it is equivalent to asking a newborn baby to start working on an algebra problem! In any case, once initialised, the class mapping for a particular value can be extracted rapidly. Zhang *et al.* [118] were able to use DRS with some success on three different problems, although the largest multiclass problem studied consisted of only four classes.

A similar solution aims to map the output of a program given a certain class onto a probability density function which can then be used to identify the classes; see Smart & Zhang [119]. This has the advantage of providing a level of confidence estimation, provided that the chosen probability density function models the distribution of the data accurately. However, it imposes a heavier burden of computation on the system. We shall see later in this chapter an improvement to DRS by the author which permits an indication of confidence to be calculated in a more efficient manner.

4.1.7 Computer Vision Inspired Techniques

By now the reader may have the impression that GP researchers are quite good at contriving new ways to solve particular problems. As we have seen, one of these problems is how to select thresholds automatically. Although Dynamic Range Selection is one flexible solution, the computer vision literature also suggests established techniques for tackling this kind of task, usually for the purposes of grey-value segmentation. A couple are introduced briefly below.

Variance Based Thresholding

Assume that the output of the classifier, when plotted, yields a bimodal distribution. The first peak in the ditribution belongs to one class, and the second belonds to another. It is necessary to find a threshold that divides the distribution into two. One simple means of automatic theshold search uses the variance between classes compared to the overall variance of both classes in order to establish a reasonable threshold position, proposed by Otsu [75]. If the output of the solution for every sample is collected, then a histogram of its output may be computed. As the histogram forms a discrete probability distribution of N samples over a range of different values i, then a threshold t will divide the distribution into two classes whose values of i are either less than t or greater-than-or-equal to t. The idea is choose the value of t that maximises the between-class variance σ_b^2, an indicator of the "distance" between two classes:

$$\underset{t \in \{0,1,2,...255\}}{\arg\max} \left(\sigma_b^2\right)$$

where the between-class variance is calculated as:

$$\sigma_b^2 = p_0 \left(\mu_0 - \mu_t\right)^2 + p_1 \left(\mu_1 - \mu_t\right)^2$$

where p_n is the total probability of class n, μ_n is the mean of class n and μ_t is the global mean. Since the total variance is equal to:

$$\sigma_T^2 = \sigma_b^2 + \sigma_w^2$$

then by maximising the *between*-class variance σ_b^2 , the *within* class variance σ_w^2 is minimised. Once the threshold has been discovered, the output of the GP program for each value may be decided and the fitness calculated as before. Since the process of finding the threshold is unsupervised, GP is compelled to search only for useful discriminant functions.

Entropy Minimisation

Another approach, first proposed by Kapur [120], aims to identify an optimum value for t by minimizing the entropy within the two class distributions defined by t. Distributions whose entropy is low tend to be clustered together neatly. Again, t is chosen by evaluating through a series of different values and choosing the value that yields the lowest entropy:

$$\underset{t\in\{0,1,2,\ldots 255\}}{\arg\min}\;(H_0 + H_1)$$

Where H_0 is the entropy of the distribution to the left of t, and H_1 is the entropy of the distribution to the right of t. They are calculated using the standard entropy equations:

$$H_0 = -\sum_{i=0}^{t} \frac{p_i}{p_0} \log \frac{p_i}{p_0}$$

$$H_1 = -\sum_{i=t+1}^{N} \frac{p_i}{p_1} \log \frac{p_i}{p_1}$$

Both of these techniques are well suited to interpreting the output of a binary GP classifier, and do not appear to have been used for this purpose before. Having described a number of representations for classifiers, it is now necessary to decide upon which is most appropriate for the author's classification system.

4.1.8 Choosing a Representation

Loveard and Ciesielski [117] published results comparing five GP representations: decision trees, evidence accumulation (very similar to the "modi" program structure), binary decomposition and static and dynamic range selection (DRS), on a series of classification benchmarks. Their results showed that DRS often yielded the best results when classifying a number of datasets; these results are confirmed by the author's own experiments, which are not shown here. Other comparative investigations by GP authors are relatively few and far between. In this section the author shall make some of his own, slightly different, investigations, starting with a comparison between DRS and the threshold choosers borrowed from computer vision. Although DRS is also applicable to multi-class classification, in the binary case it is worthwhile investigating its performance relative to the variance or entropy thresholding techniques, which are each based on a theoretical framework for identifying an optimal threshold.

At this point, the author must beg forgiveness from the reader for presenting experiments using a Genetic Programming system which has not yet been described! More details about the GP set-up itself, including parameters and the function set, await in the following chapter. For now let us assume that we have access to a suitable GP system, which simply allows us to measure the *difference* in fitness between different classifier representations.

In the first experiment, GP was used to discover classifiers for five different public classification datasets concerned with binary classification. The datasets are described at the beginning of this thesis, and will be used throughout the following chapters. Fifty classifiers were evolved for each dataset using three different classification representations: DRS, Variance and Entropy Thresholding. The average results on test data are shown in Figure 4.5. The results show that the DRS representation chosen by Loveard can itself be outperformed by both thresholding techniques on each of the five datasets. The differences between DRS and the thresholding algorithms were statistically significant in each case. Although classifiers using thresholding techniques need to check many different values of t in order to decide upon the optimum threshold, they do not take substantially longer to evolve than with the DRS technique. In any case, they will execute slightly more quickly following evolution[2]. Of the two thresholding techniques, entropy thresholding appears to outperform variance thresholding in most cases, although the difference is not always significant. Interestingly, the two thresholding techniques typically achieved worse *training* fitness than DRS, but were found to be more accurate on test data: an indication that the classification map was over-fitting the training samples.

Our quest for a suitable GP classifier would appear to be one step closer: for binary problems entropy thresholding seems to produce the most accurate classifiers. Problems in vision such as detection or background subtraction may be well suited to this kind of representation.

However, it has already been stated that many (perhaps most) other tasks in vision are multi-class in nature. Situations involving shape, gesture, digit, vehicle or face recognition each require various different classes to be discriminated from each other. In order to develop a classifier that is sufficiently generic to solve a variety of vision problems, the

[2]The variance and entropy methods require only a single threshold comparison to make decisions, dynamic range selection requires division and an array access to choose the class.

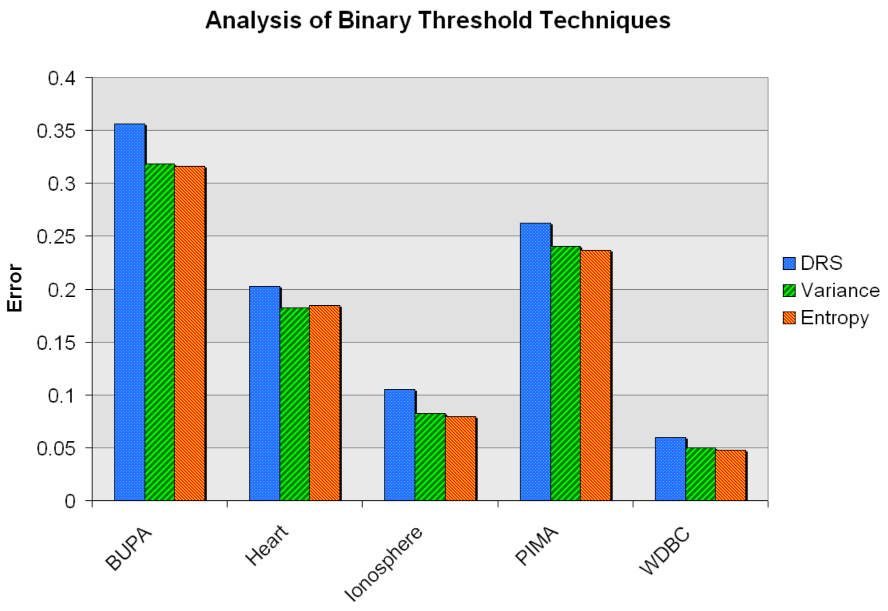


Figure 4.5: The average errors of three different automatic thresholding techniques on five public datasets concerned with binary classification.

representation must be capable of solving multi-class problems – which suggests the use of DRS. Although we have seen that multi-class problems may be decomposed into a series of binary problems, the thresholding techniques shown here suffer from limitations even in the binary situation (for the same reason that Genetic Programming was chosen as the author's preferred classification technique). A single threshold contains an inherent assumption that two classes within the data can inhabit clusters that can be distinguished by a single plane (if $f(x)$ is linear). In fact, it may well be that the members within one arbitrarily-defined class actually belong to two or more distinct clusters, or that the range is defined by two thresholds – in either case the thresholding technique would produce worse results.

For these reasons the author's classification *system* shall use both entropy thresholding and DRS, dependent on the type of problem to be solved. Before discussing the classification system as a whole, the author shall introduce various enhancements that improve the accuracy, genericity and functionality of DRS.

4.2 Investigating Range Selection

Previous research has shown that DRS is the one of the more suitable classification representation for solving multi-class datasets. Nonetheless, the author has dedicated some time to analysis and improvements to this technique. This section covers enhancements to DRS and presents supporting results.

4.2.1 DRS Filling

The reader may have been wondering what becomes of those slots that are not touched by any training data and are therefore not assigned a class label during training. Such slots are incapable of producing a classification, so may reduce the ability of the DRS classifier to generalise, or indeed may cause the classifier to behave unreliably on seemingly straightforward test data.

Using a synthetic dataset whose points are defined by two attributes, one can plot the positions of training data in two dimensions and visualise the problem (see Figure 4.6). The figure shows that GP can find a decision plane that divides the two classes, but while all the training data can be classified correctly, some slots remain unassigned. The gaps left by unassigned slots may cause an unseen sample to be classified unreliably, even if it is well within the normal range for its class. The author's original solution was, for data falling into an unassigned slot, to perform a local search until it found a neighbouring slot that was assigned a class, then to return that class. As well as being inefficient, this method does not preserve any margin of error between classes that may exist: returning an unassigned class may be a genuine indication of ambiguity, which could be useful.

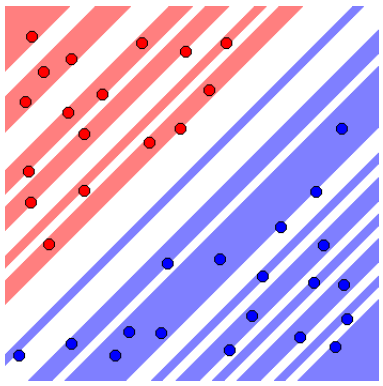


Figure 4.6: Each sample in the 2D dataset is plotted, along with the decision of the classifier at every point in 2D space (red for one class and blue for another). This figure shows that the DRS map needs to be filled in to ensure classification is reliable.

This author's solution is to invoke a separate algorithm once after training which deals with unassigned slots by "filling them in", while retaining any margin between the classes, a technique hereafter referred to as *fDRS*. The author developed the following algorithm for this purpose:

```
for each slot i in slots[] {
    if (slot is unassigned) {
        // see whether this slot can be "filled in" to make the DRS translator more reliable
        before = first unassigned class before this slot
        after = first unassigned class after this slot
        // if both before and after are the same, then "fill in" the gap
        if (before == after) {
            slots[i] = before;
        }
    }
}
```

An important characteristic of a good classifier is its ability to *generalise*, or demonstrate good performance on data that it has not encountered during training. The effect of this algorithm is shown in Figure 4.7. By filling in the slots, the generalisation capabilities of the map are increased, especially in circumstances where the training data are sparsely distributed or the number of slots is high. The algorithm maintains the margin between the two classes (if there is one), so that the unassigned result comes to indicate genuine ambiguity.

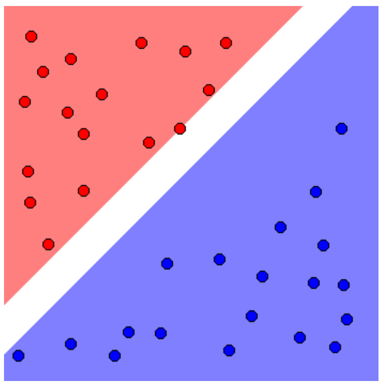


Figure 4.7: The result of the filling algorithm on the same program classification map

Although the algorithm appears to work quite well on our simple 2D example, does it improve the generalisation capabilities of GP-based classifiers on real multidimensional data? The algorithm was put to work on ten different commonly-used datasets. The results are shown in Table 4.1, which shows the average test error both with and without the filling algorithm running and the difference between the two expressed as a percentage.

Dataset	Test Error % (DRS)	Test Error % (fDRS)	Percentage Improvement
BUPA	0.361	0.356	+1.24
Glass	0.423	0.415	+1.98
Heart	0.228	0.214	+6.07
Ionosphere	0.119	0.107	+10.57
Iris	0.127	0.065	+48.29
Pima	0.250	0.247	+1.29
SatImage	0.251	0.250	+0.11
Thyroid	0.027	0.026	+3.28
Vehicle	0.405	0.404	+0.45
WDBC	0.053	0.041	+23.61

Table 4.1: DRS Generalisation Improvement by using the DRS filling algorithm

On datasets without separate test sets the appropriate k-fold cross validation technique was used to estimate the error. Each result is the average over 25 runs.

The results in Table 4.1 show that the filling algorithm in fDRS has an exclusively positive effect all the unseen portions of the datasets. Of course, this is not particularly surprising but it does confirm that the choices made by the algorithm improve the program's generality (the training data fitness would not be affected). Although the difference is slight in some cases, a paired T-test nonetheless shows the difference to be statistically significant in each case. While the filling algorithm can be executed in a single pass on any program classification map quickly, it can have a significant effect on the generalisation capability of the classifier, which in the case of the Iris dataset, demonstrated a substantial improvement. The algorithm has most effect on datasets whose samples are few or sparsely arranged.

4.2.2 Parameter Free DRS – "DRS2"

Since the fitness function is one dimensional, it is difficult to establish whether an individual's performance is hampered by its logic or by its inability to translate its decision onto the expected class boundaries. Dynamic range selection partly removes the need to evolve the latter, which reduces the complexity of the problem so that accurate individuals

Dataset	Test Error % (fDRS)	Test Error % (DRS2)	Result
BUPA	0.356	0.339	Not Significant
Glass	0.415	0.377	DRS2 Better
Heart	0.214	0.216	Not Significant
Ionosphere	0.107	0.104	Not Significant
Iris	0.068	0.064	Not Significant
Pima	0.247	0.245	Not Significant
SatImage	0.250	0.242	DRS2 Better
Thyroid	0.026	0.024	Not Significant
Vehicle	0.404	0.376	DRS2 Better
WDBC	0.043	0.038	DRS2 Better

Table 4.2: DRS Method Comparison, including the result of an unpaired samples T Test.

may be evolved faster. However, “basic” dynamic range selection still assigns slots within a specific range, for instance from -250 to $+250$. Since the output of the individual is to some extent dependent on the magnitude of the feature vector values, the range may be restrictive: the function may still need to incorporate a multiplier to ensure that values generally fall within the given range. Even if the classifier’s output is within the range, it may not occupy the *whole* range, so the number of effective slots may be smaller and the position of the thresholds may be less accurate.

For these reasons the author implemented an extension upon fDRS, hereafter referred to as *DRS2*, where the range itself is also dynamic, ranging from the lowest known output of the individual to the highest. The removal of arbitrary thresholds may increase the genericity of this approach, provided it can perform as well as the normal DRS method. This enables all the available slots to play a part in the classification process and saves the individual having to scale its output up or down to fall within an arbitrary range.

Results of experiments with the two DRS techniques on ten public data sets are displayed in Table 4.2. The results show that the updated DRS technique, DRS2, is either equivalent to with or superior to the previous algorithm on all occasions. It was also observed that the DRS technique yields programs of a slightly smaller size, which may be explained by the individual not needing to scale to an arbitrary range. As well as helping to ensure all slots have the chance to be used, DRS2 does away with two parameters, which helps make it more readily applicable to unknown tasks. In the following chapter the author shall present further attempts to rid GP of various parameters.

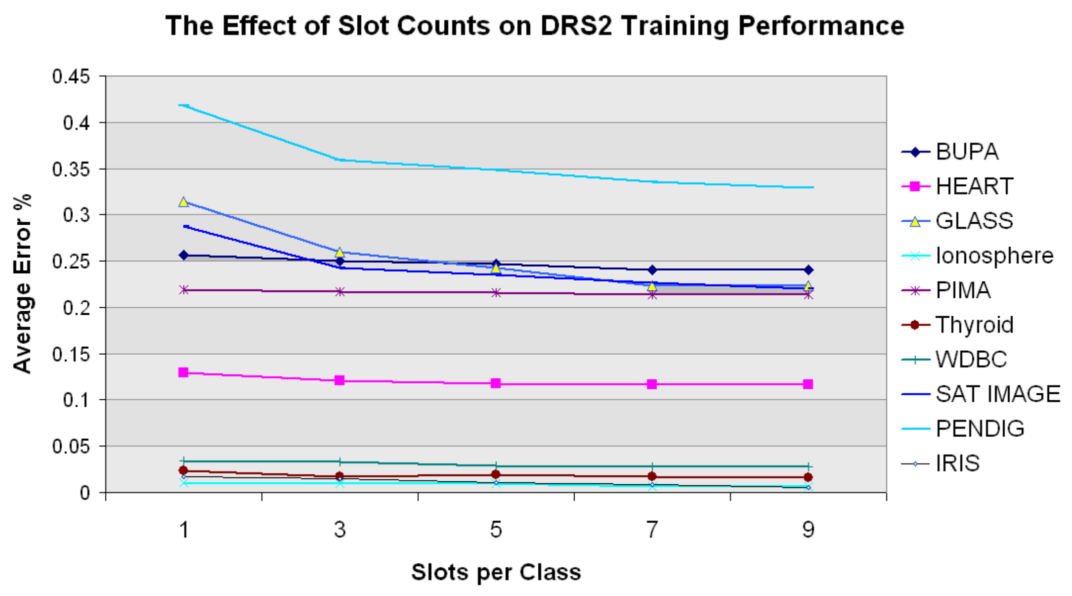
4.2.3 Number of Slots

The final arbitrary parameter associated with a program classification map is the number of slots to be used. Larger numbers of slots will permit the map to place thresholds in more exact locations and fit the training data better. Excessively large numbers of slots may result in only one unique instance fitting into each slot, at which point the DRS map would begin to resemble a 1-nearest-neighbour algorithm and may become less able to generalise. Smaller numbers of slots may improve the generalisation capabilities of the classifier, although too few slots may not be enough to divide the classes adequately. It is therefore worthwhile to investigate what constitutes “large” and “small” numbers of slots and see how slot numbers affect the classifier’s performance and generalisation ability.

The original number of slots, 501, appears to be excessive for most situations. Providing such a number of independently assignable slots may tempt the GP process to simply evolve some means of *dispersing* the data samples across the whole range. This would yield increasingly good results on training data, but poor results on testing data. The author’s experiments with more modest numbers of slots indicate this trend – that increases in training fitness correspond to decreases in unseen test fitness, a symptom of over-fitting.

The author’s first deed was to define the slot count not as an arbitrary value but as a multiplier of the number of classes, a crude indicator of the problem’s complexity. An experiment was devised to examine the differences between classifiers that use DRS for different slot count multipliers. DRS2 was used as it guarantees that every slot has a chance of being used. Again, each classifier was evaluated against a number of benchmark datasets, with each experiment repeated 25 times to determine the average error on each. The results of the experiments are presented in Figure 4.8, in which the average fitness values are plotted against the slot count for each dataset.

The figure shows that increased slot count ratios typically brought about a modest reduction in error on training data, although in some cases the difference was more substantial. On the equivalent test data, by contrast, the error remains roughly at the same level for many datasets, indicating that the classifier is over-fitting the training data. However, for some datasets, the improvement on test data is substantial for increased values of the ratio. A casual analysis of the graph shows that a ratio of 7 appears to satisfy the



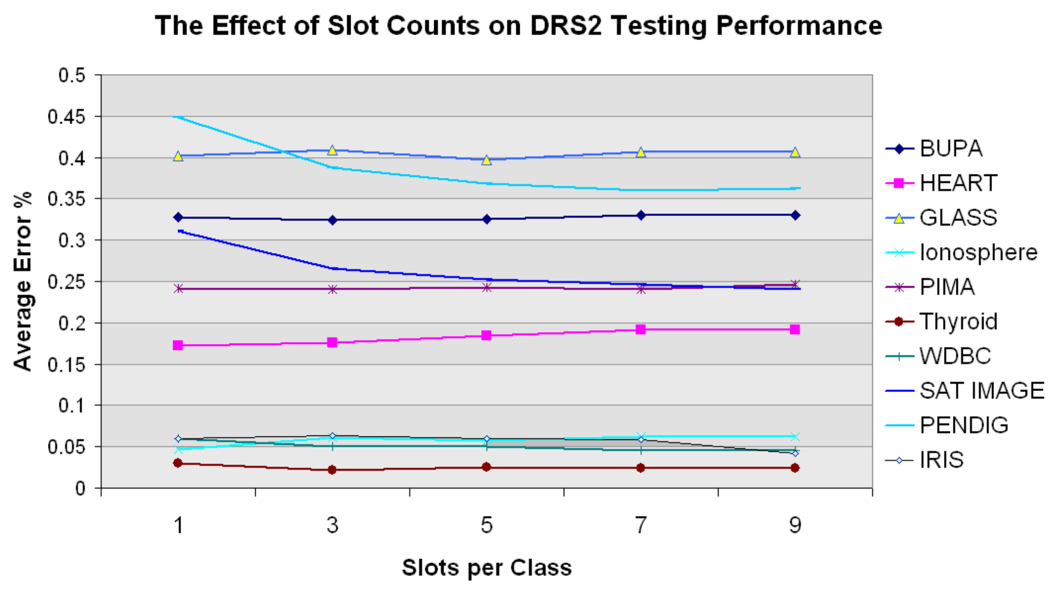


Figure 4.8: Fitness results for classifiers evolved using 5 different DRS slot ratios.

majority of datasets. Having investigated a reasonable way of determining the slot size and chosen a value that seems to deliver a good balance, the DRS technique may be regarded as parameter-free, which is a positive step towards developing a generic learning component.

4.2.4 Confidence Estimation

Moving on towards more substantial adaptations of the DRS technique, there are certain additions that can be made to increase its functionality, including giving some measure of confidence. As we have seen, the majority of classification representations by GP produce "hard" classifications, where a feature vector is assigned a class label without any indication of the classifier's certainty on the matter. This is adequate for some straightforward classification problems, where there is a reasonable margin between classes, but is not so helpful in other tasks where there is some degree of overlap between classes. Indeed, in some cases it may be more appropriate to suggest more than one potential class for a given feature vector. One example, to be tackled later in this thesis, is optical character recognition (OCR), which is characterised by a reasonably large number of classes, some of which appear quite similar to others. For such a problem, it is difficult to differentiate every class with a high degree of certainty. In these situations it is more helpful to provide the likelihoods that the feature vector belongs to *each* class. If, for some reason, the most likely character seems inappropriate, the system can fall back to the next most likely and so on. This would be the case if the characters represented the postcode of an address, for example.

As we've seen so far, dynamic range selection returns only "hard" classifications. Interestingly though, the algorithm *does* have access to information that can be used to compute some measure of confidence. The reader will recall from page 70 that the class of a particular slot is chosen by first identifying all the items of training data M that are allocated to the slot by the GP program, then choosing among them the most popular class using a winner-takes-all strategy. If M is entirely homogenous, that is to say that all members have the same class c_0, then the classifier can be reasonably certain that c_0 is the most appropriate label to return. However, if M is composed of, say, 6 instances of class c_0 and 5 instances of class c_1, then the slot should still choose to label itself with c_0 but it should not be so certain. If the slot can express its confidence, it might state that, for an output within its range, there is a $6/11$ chance that it is c_0 and a $5/11$ chance that it is c_1. Therefore, confidence estimates for each class can be computed according to the proportion of members from the winning class out of the total number of hits the slot received. This is the basis behind the author's second DRS variant, referred to henceforth as *DRS2C*.

It is rather difficult to measure the effect of confidence estimation directly: it can not be used explicitly but instead is made use of by subsequent processes, as was alluded to in the postcode OCR example above. The system may be interested in the n most significant classes for a particular feature, or may choose to consider only data points that can be classified with a particular level of certainty. Each case may be task specific. In the next section we shall investigate some post-classification techniques using GP that permit a quantitative assessment of confidence estimates.

4.3 Classification Frameworks

Although the representation of evolved programs plays a key role in the learning of accurate classifiers, there are nonetheless various different means of improving classification solutions, which may include additional processing before, during or after the course of evolution. In this section we shall look at three techniques intended to improve the accuracy and speed with which classifiers may be evolved using GP.

4.3.1 Classifier Fusion

Depending on the course that evolution takes, different individual classifiers may devise completely different ways of solving a problem – much the same as humans! Genetic Programming may be particularly prone to this, owing both to the relatively random manner in which the search is conducted and the potentially large size of the solution space. Thus, while all individuals may have equivalent overall fitnesses, they may make different sorts of mistakes on different samples of training data. Although the accuracy of individual classifiers cannot be improved after the evolutionary process has completed, if we ask several classifiers to identify each data point, then we may reach a consensus opinion which is, on average, more accurate than that of any of the individuals. This is the motivation behind *multiple classifier systems* (and parliamentary democracies!) which aim to produce results which are more robust than those of an individual classifier. Of course, it is important that individual classifiers are *independent* from each other, so that their "fusion" together is meaningful: if each of the classifiers always agree, then the technique is adding a computational burden without its consensus providing offering any improvement in accuracy. A straightforward way of ensuring sub classifiers are independent is to train each on different sections of the training data, or to weight the training data differently for each classifier; [121].

There exist various approaches for combining classifiers. A couple of straightforward ones are described briefly below.

Majority Vote

Given three or more binary "sub-classifiers", each makes a decision with regard to a particular input vector. Each classifier is allocated one "vote" with which to support its chosen class. The class which receives the most votes overall is chosen and returned. This approach does not require an estimate of confidence. As such, for the consensus to be meaningful, all sub-classifiers should be equally certain, and all sub-classifiers should have roughly equivalent accuracy.

A more rigorous approach extends the fusion technique to weight the overall vote of any sub-classifier by its own fitness, such that the fitter classifiers have more say in the decision making process.

Committee Voting

The majority technique is quite democratic, implementing as it does the ideal of "one decision one vote". However, in the world of classification not all decisions are created equal! Giving equal weight to both certain and uncertain classifications may incorrectly reward the wrong class. The author's *DRS2C* technique makes it possible to quantify how certain a particular vote is. Using the concept of confidence, it may be possible to reach fairer decisions using so-called *committee voting*.

In the committee approach every class may receive some credit from every sub-classifier. In contrast to majority voting, each vote is weighted according to the classifier's confidence. Thus each sub-classifier still contributes the same amount, but is able to hedge its bets over different classes when appropriate. This appears to be fairer, although it is reliant upon the confidence estimates themselves being accurate.

Experiments

The two techniques provide a means of assessing whether confidence estimation can bring about improvements in accuracy. It is also necessary to measure any improvement in accuracy that ensemble classification techniques in general can yield over single classifiers.

Since multiple classifier systems are inevitably more computationally expensive than single classifiers, it is worthwhile to assess whether they improve accuracy sufficiently as to justify their increased computational "cost".

In order for the comparison to be fair, the fusion classifier should take the same amount of time to evolve as a "single" classifier. In order to ensure this, the author devised a strategy that makes use of *island selection* (see page 27), a technique which divides the population into separate sub-populations or "islands" (in this case 7 islands), within which evolution can proceed partly or entirely independently. In this situation the islands were entirely independent, permitting several classifiers to be evolved simultaneously. Each island was allocated training data which were partitioned in a manner similar to that used in cross-validation – randomly, but maintaining class distributions in each[3]. Thus each island would concentrate on a different part of the training set, helping to ensure that the classifiers evolved were independent. In fact, this approach requires *less* computation, since it uses the same number of individuals but each individual is evaluated against a smaller number of samples. The single classifier was trained on all the data for the equivalent amount of time.

At the end of evolution, the best individual from each of the 7 islands was chosen, yielding 7 potential members of the multiple classifier system. The same individuals were tested both in majority voting "mode" and committee voting "mode', permitting direct comparisons to be made. As before, the experiment was repeated 25 times; results are averages of performance on test data. The results on test data are summarised in Figure 4.9.

The results show that fusion techniques broadly outperformed equivalent single classifiers, sometimes by a substantial margin; particularly so on some of the larger multi-class datasets (PenDigits and SatImage). The difference between confidence and majority is most marked on the PenDigits dataset, which is concerned with the recognition of 10 digits and is the kind of task that motivated the development of a DRS variant that could produce confidence estimates. On the 8 occasions where the difference between the fusion techniques were significantly different, the committee voting scheme produced the best results on 5. This would appear to show that, for some datasets, the additional information provided by the confidence estimate can make a significant contribution to the

[3]This was done by allocating all the training samples into bins, one for each class. Each bin was then shuffled, with one sample taken out of each bin for each partition in round robin fashion, until all bins were empty. This is a fair way of ensuring that each partition has a roughly equal class distribution, and have as similar sizes as possible.

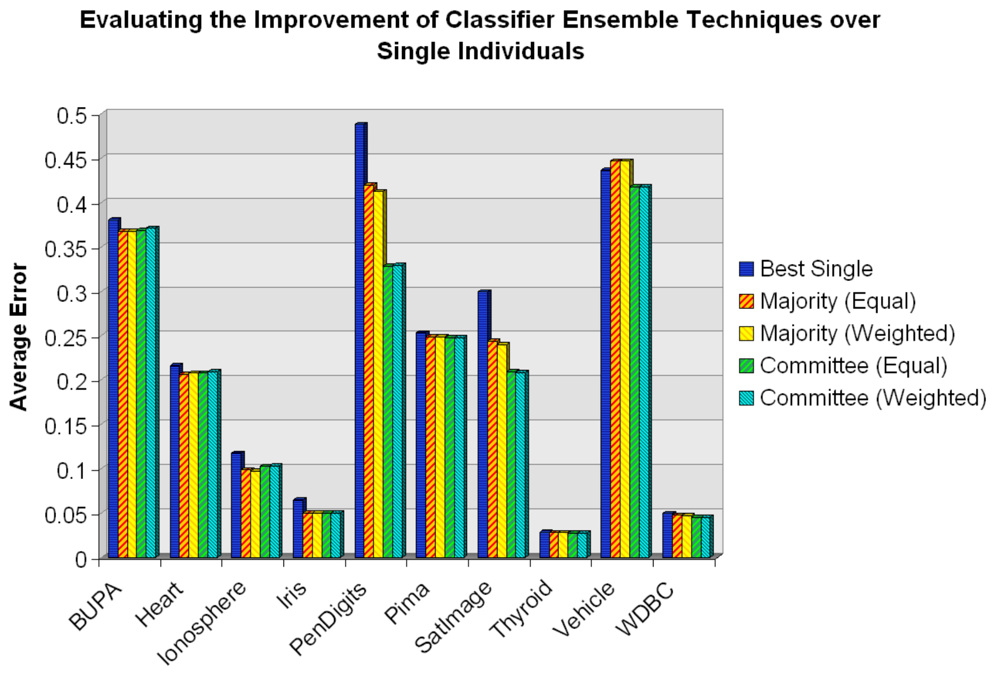


Figure 4.9: Fitness results for classifiers evolved by regular GP, then with those combined in Majority and Committee Voting scenarios.

accuracy of the system.

The figure also shows the results when the majority and committee classifiers were executed in two sub-modes: one giving equal weight to each classifier and the other weighting the classifier's votes by its fitness, so that fitter classifiers would have a proportionally larger vote than less fit classifiers. The chart shows that this addition can make a modest improvement to results from the majority fusion technique, but generally yields little difference for the committee ensemble.

It is difficult to assess which committee technique is "better" overall from these results. Langdon [122] pointed out that[4]:

> "There is no single, best combination scheme nor any unequivocal relationship between the accuracy of a multiple classifier system and the individual constituent classifiers."
>
> W. B. Langdon

Fortunately it is not necessary to make such an assessment – since the author's multiple classifier system can operate in either mode, at the end of the run both can be tried in

[4]In fact Langdon used Genetic Programming to develop the fusion technique itself.

order to find the better one for a given problem[5]. The reader may also be wondering what the best committee size might be on each occasion – the author's software implements an approach whereby the seven classifiers are used in all different permutations in order to discover the most appropriate "sub-committee". This process takes a couple of seconds to complete, but need only be performed once per evolutionary run. The author would claim to have produced a classifier fusion technique which is sufficiently flexible as to try different classification techniques, while requiring no more evolution time than a single GP classifier.

4.3.2 Classifier Evolution by Partial Solutions (PS)

The author's second approach aimed to address the time taken to evolve classifiers in a general sense. Inspired by the boosting of weak classifiers in [92] and classifier systems in general, the author developed a scheme in which GP programs are evolved as simple expressions which solve only part of the overall solution and are thus termed *partial solutions*. As each partial solution is not expected to correctly classify every sample, the programs are small and so may be discovered and evolved quickly.

As useful partial solutions are identified they are added to a separate, "strong" classifier whose scope exists outside of the GP system (see Figure 4.10). A useful partial solution is one that can correctly classify parts of the training data that have not been solved already. Following the discovery of such a solution, the problem is subsequently redefined to include only the remaining training data. This removes the need for already-solved data to be continuously re-evaluated, as is the case with standard GP, and also ensures that the "knowledge" of the partial solution is protected. This prevents regression in a more granular way than does elitism (page 33) The system continues to evolve partial solutions until the strong classifier is capable of a certain level of accuracy.

Fitness Criteria The fitness of a partial solution with regard to a particular class c is calculated by dividing TP_c, the number of times it correctly labelled a sample with class c, by the total number of training samples N, added to the number of missed classifications of c samples, FP_c, as shown below:

$$\text{fitness}_c = \frac{\alpha TP_c}{N + \beta FP_c}$$

[5]However, it is acknowledged by the author that there are *other* classifier fusion techniques in addition to these.

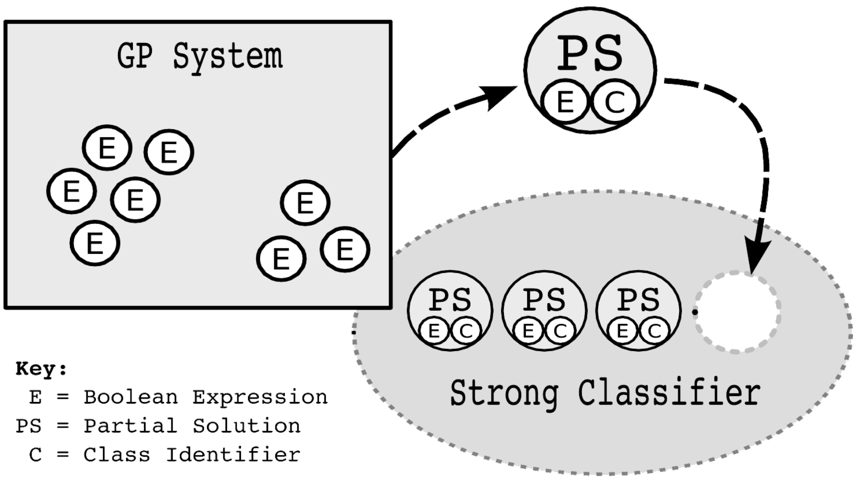


Figure 4.10: GP by Partial Solutions creates a strong classifier from a number of smaller solutions.

The factors α and β allows the fitness measure to be adjusted to affect the individuals' sensitivity or specificity.

But what is c? Each partial solution is treated as a binary classifier with respect to a class X, which is chosen by evaluating the partial solution with respect to the set of classes C in the task, and choosing the class for which it has best fitness:

$$X = \arg\max_{c \in C} \text{fitness}_c$$

This is a brute force approach but ensures that the logic and interpretation are kept separate.

Adding to the Strong Classifier Fitness is used to drive the evolution of partial solutions. However, in order to decide whether a partial solution should be added to the strong classifier – that is to say that it is able to solve a new part of the overall solution – four separate criteria were devised:

1. *Does the partial solution discriminate?* If the solution returns only 'true' or only 'false' it is not capable of making decisions. GP often evolves this kind of lazy solution in response to training data that are weighted in favour of one particular class. GP by partial solutions avoids this form of code bloat.

2. *Is the partial solution unique?* If the classifier returns the same results for every instance as another partial solution that has already been chosen, then this classifier

is usually discarded[6].

3. *Does the partial solution return 'true' for data that have not yet been solved?* Each item of training data has a field indicating whether it has already been solved. Each candidate partial solution has to identify at least one instance of training data that has not yet been solved in order to be used.

4. *Does the partial solution return 'false' for data in other classes?* If a binary classifier returns 'true' for instances of one class, it should return 'false' for all instances of other classes. However, it *may* return 'true' mistakenly for other classes provided that they have already been completely solved by other partial solutions. We describe this solution as being *dependent* on the other classes.

If the partial solution matches all these criteria, it is added to the "strong" classifier. Several partial solutions may be discovered during the course of a single generation – another characteristic which makes GP by partial solutions faster.

A version of island selection is used with crossover only occurring between parents with the same value of X, which are more likely to share some similarity.

The strong classifier produces classifications by evaluating each of its partial solutions in turn (starting with the first added). If the partial solution returns 'true' for a given input vector, the class C associated with the partial solution is returned, otherwise the next partial solution is executed, and so forth.

Experiments

A series of experiments were conducted by the author in order to explore the practicality and effectiveness of the partial solutions approach. It was found that for certain problems, partial solutions could deliver more accurate solutions an order of magnitude more quickly than could conventional GP.

However, it was observed that, for certain problems with large amounts of training data, the system generated *too many* partial solutions. Furthermore, the author was uneasy about the manner in which partial solutions could be added to the strong classifier but not changed or refined. As each partial solution is selected, the data that it solves correctly are removed from the training data, so the system is incapable of replacing it with a later, better individual which may be able to solve more instances of the same kind data and thus offer better generalisation capabilities.

[6]Unless it is smaller or less dependent than the existing one, in which case the existing solution is replaced.

There is nothing to say that partial solutions cannot find a small set of generic solutions instead of a large set of over-fitted solutions, but there is an inherent randomness in the process.

4.3.3 Intelligent Classification System (ICS)

We have seen that as individuals get better, it becomes more tricky to improve them without breaking something, particularly when the crossover/mutation operators have essentially no idea what parts of the tree are useful and which parts are under-performing. This is a characteristic of the so-called stability-plasticity dilemma, which has been demonstrated empirically by Banzhaf [123]. In essence, learning by GP becomes more difficult as evolution progresses. The concept of Partial Solutions is to protect useful components as much as possible to mitigate this problem. Still, the randomness of Partial Solutions caused the author some degree of unease. For this reason, a more "careful" classification scheme was developed which addresses this issue while still taking advantage of the underlying concept of partial solutions. It also makes use of the other innovations described in this chapter.

Rather than split the problem into pieces in a haphazard way, the author's third classification system, termed "Intelligent Classification System" (ICS), divides the problem in a repeatable and logical way. "Intelligent" refers to the way in which the system attempts to replicate the way a human would approach a problem – it performs experiments along the way and otherwise attempts to choose the best parameters and approaches for particular problems.

In general, ICS initiates a straightforward process of "one-vs-all" binary decomposition on the training data in order to split it into sub-tasks. Each sub-task is evaluated in order to assess its difficulty. A classifier for each sub-task is then learned, starting with the easiest. The classifiers are then assembled into a single multi-class classifier, of the type shown in Figure 4.11. A process of refinement then initiates in order to improve the classifier where possible. The means by which ICS goes about these tasks will now be covered in more detail.

The Choice of Binary Decomposition

Before continuing further, it is worthwhile considering whether binary decomposition is a better means of producing accurate classifiers than evolving a single multi-class classifier. An experiment was run to assess the difference between the two techniques, given an

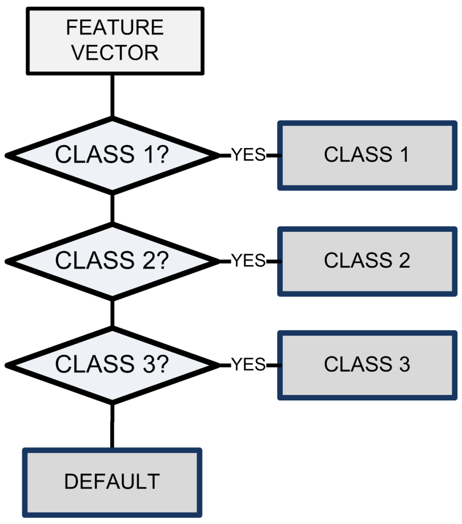


Figure 4.11: Binary Decomposition of a multi-class Problem. Each class may be returned by one sub-classifier. If the classifier returns a negative result, the next classifier in the list is tried and so on.

equal time to develop a classifier in each case. The results are shown in Figure 4.12, which shows that the binary decomposition approach yields classifiers with lower error than those from the single multi-class classifier, on average. Although the difference was not statistically significant on the Iris dataset, for the other 4 cases, the binary decomposition approach produced significantly more accurate results.

Of course, if the problem is already binary it cannot be split up further. In this case ICS uses the entropy and variance thresholding representations (discussed earlier in Section 4.1.7), which were shown to be superior to DRS in binary situations. For multi-class problems, the algorithm must develop one classifier per class, with each classifier evolved to use the author's DRS2C representation. Although these classifiers are also binary in nature, experiments reveal that range selection outperforms entropy and variance thresholding here, as the classifier may need to delimit a particular class with more than one threshold.

If provided with more than one CPU, ICS typically attempts several GP runs per class in its search for a classifier; variance and entropy thresholding are used alternately.

Estimating Difficulty

The reader is referred once more to Figure 4.11. The multi-class classifier consists of a chain of binary classifiers. If the "A" classifier returns true then "A" is returned, otherwise

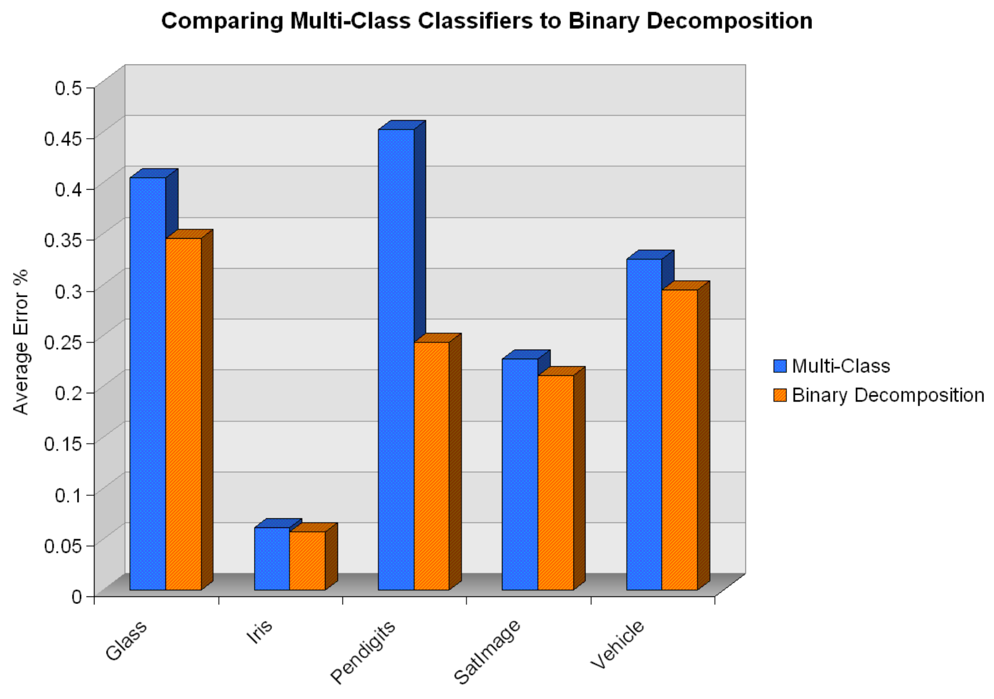


Figure 4.12: Binary Decomposition vs Multi-class Classification.

the "B" classifier has an opportunity to classify the data and so forth. There are two observations that one can make from this architecture. The first is that the first classifier in the chain should be the most accurate, as errors will carry forward. The author submits that the eventual accuracy of a classifier is proportional to the difficulty of the task. Thus, if we can estimate or measure the difficulty of classifying each class beforehand, we can organise the binary chain in a manner that yields fewer errors.

The second observation of this kind of architecture is that if classifier "A" returns false for whatever reason, there is no further opportunity for samples of "A" to be identified. Following the evolution of a classifier for "A", we can remove all the "A" samples from training data. This has two effects: the training time for subsequent processes of evolution is reduced and later problems become marginally easier to solve. This is the essential motivation behind partial solutions, except in this instance the partitions are chosen in a more systematic manner.

The first stage in ICS, therefore, is to estimate the "difficulty" of each component binary problem, so that the order of evolution can be established. This is performed by running a k-nearest-neighbour classifier on each problem and assessing its output on a subset of training data reserved for validation. Since k-nearest-neighbour requires no training as such, this step can be performed relatively quickly. The error of the k-NN classifier on each class is assumed to be indicative of class confusion. Classes for which k-NN achieved the lowest error are executed first, with the expectation that some of the class confusion will be alleviated as the training data is reduced.

The reader may be wondering whether k-nearest neighbour is an ideal predictor of GP performance. Indeed, the k-NN algorithm is an example of "lazy learning", where there is essentially no training procedure beyond committing sample data to memory. GP, by contrast, evolves an analytical solution to the problem, so the data that confuse k-NN may not be a problem for GP and vice versa. In order to assess the impact of class ordering, the author devised a series of experiments, in which the classes were ordered by the predicted "easiest", then "hardest" and randomly. These were each compared to "standard" binary decomposition, which did not implement any class removal. The hypothesis is that if the k-NN classifier's performance is a predictor of GP performance, then the runs for which classes were ordered by easiest tasks should return the lowest error. Ordering classes the opposite way, by the *hardest*, should yield worse performance. Finally the classes were ordered randomly to make an additional comparison. The average results over the course of 25 runs each are shown in Figure 4.13.

The results show that the ordering of classes does indeed have a significant effect and that ordering by the hardest as opposed to easiest has a negative effect, indicating that the ordering derived from the k-NN algorithm is not arbitrary. There may, of course, be ordering methods which yield even better results than this method[7]. It is also apparent that class removal yields more accurate solutions, since each GP classifier can concentrate on a more specific set of training data without being confused by irrelevant data. This also results the most beneficial aspect of this classification system: in contrast to most GP systems the rate of learning accelerates during evolution. Solutions to vision problems can generally be discovered within minutes, rather than hours.

Other Features

To return to the description of ICS, following the estimation of class difficulties, a solution to each component problem is then evolved in turn. Each run is repeated several times; the best classifier is chosen. Different parameter combinations are tried for different runs

[7]Indeed it would be interesting for the computer to learn the best orderings itself. This is not as intractable as it seems, if the classifier accuracy for every ordering combination is computed (which would admittedly take some time), then a set of training data could be established correlating the orderings and their observed fitness. Another GP process could then be run to find rules that lead to the most appropriate ordering.

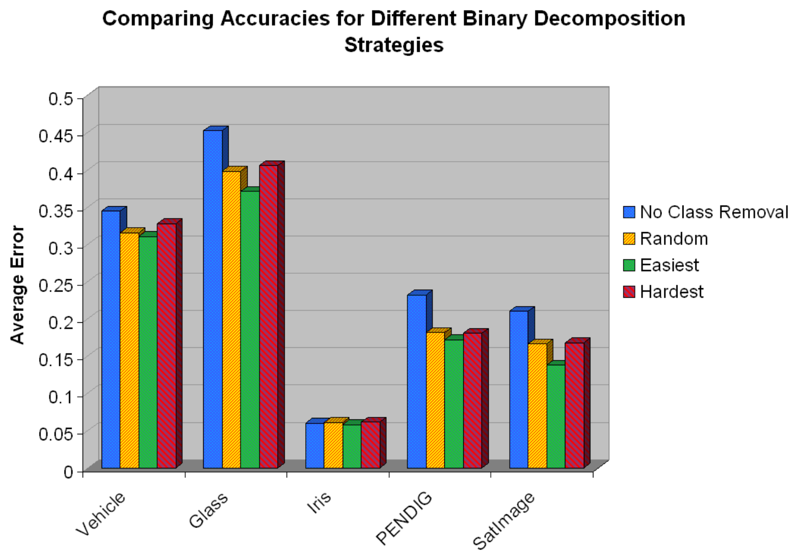
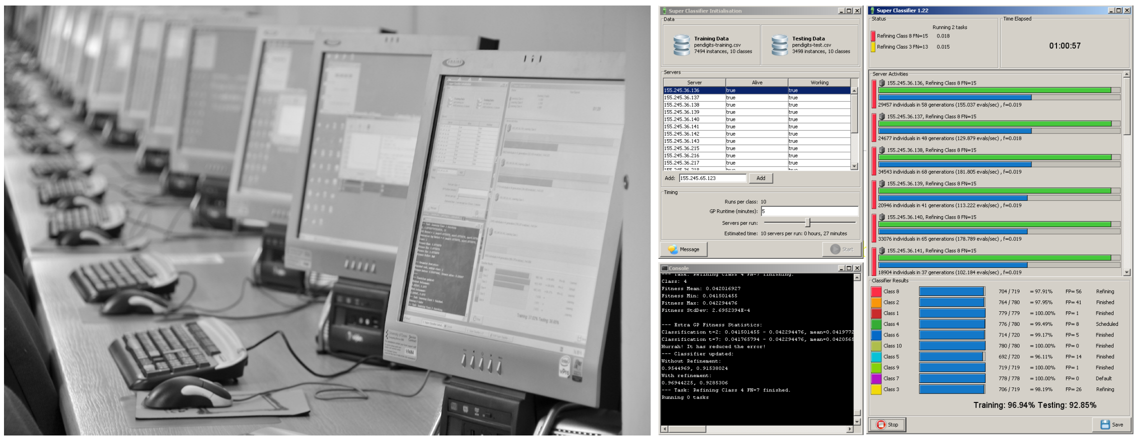


Figure 4.13: The effect of the author's ICS technique vs standard binary decomposition. "No class removal" indicates binary decomposition in which the training set is not adjusted in any way following evolution of classifiers. In Random/Easiest/Hardest, "spare" classes are removed – this appears to have an exclusively positive impact on training data. Of the three ordering techniques, ordering by easiest yields the lowest error.



Training Data
pendigits-training.csv
7494 instances, 10 classes

Testing Data
pendigits-test.csv
3498 instances, 10 classes

Servers

Server	Alive	Working
155.245.36.136	true	true
155.245.36.137	true	true
155.245.36.138	true	true
155.245.36.139	true	true
155.245.36.140	true	true
155.245.36.141	true	true
155.245.36.142	true	true
155.245.36.143	true	true
155.245.36.215	true	true
155.245.36.216	true	true
155.245.36.217	true	true

Add: 155.245.65.123

Timing
Runs per class: 10
GP Runtime (minutes): 5
Servers per run:
Estimated time: 10 servers per run: 0 hours, 27 minutes

```
Class: 4
Fitness Mean: 0.042016927
Fitness Min: 0.041501455
Fitness Max: 0.042294476
Fitness StdDev: 2.6952394E-4

--- Extra GP Fitness Statistics:
Classification t=2: 0.041501455 - 0.042294476, mean=0.0419772
Classification t=7: 0.041765794 - 0.042294476, mean=0.0420565
Hurrah! It has reduced the error!
--- Classifier updated:
Without Refinement:
0.9544969, 0.91538024
With refinement:
0.96944225, 0.9285306
--- Task: Refining Class 4 FN=7 finished.
Running 0 tasks
```

Status
Running 2 tasks
Refining Class 8 FN=15 0.018
Refining Class 3 FN=13 0.015

Time Elapsed
01:00:57

Server Activities
155.245.36.136, Refining Class 8 FN=15
29457 individuals in 58 generations (155.037 evals/sec) , f=0.019
155.245.36.137, Refining Class 8 FN=15
24677 individuals in 48 generations (129.879 evals/sec) , f=0.018
155.245.36.138, Refining Class 8 FN=15
34543 individuals in 68 generations (181.805 evals/sec) , f=0.019
155.245.36.139, Refining Class 8 FN=15
20946 individuals in 41 generations (113.222 evals/sec) , f=0.019
155.245.36.140, Refining Class 8 FN=15
33076 individuals in 65 generations (178.789 evals/sec) , f=0.019
155.245.36.141, Refining Class 8 FN=15
18904 individuals in 37 generations (102.184 evals/sec) , f=0.019

Classifier Results

Class 8	704 / 719	= 97.91%	FP= 56	Refining
Class 2	764 / 780	= 97.95%	FP= 41	Finished
Class 1	779 / 779	= 100.00%	FP= 1	Finished
Class 4	776 / 780	= 99.49%	FP= 8	Scheduled
Class 6	714 / 720	= 99.17%	FP= 5	Finished
Class 10	780 / 780	= 100.00%	FP= 0	Finished
Class 5	692 / 720	= 96.11%	FP= 14	Finished
Class 9	719 / 719	= 100.00%	FP= 1	Finished
Class 7	778 / 778	= 100.00%	FP= 0	Default
Class 3	706 / 719	= 98.19%	FP= 26	Refining

Training: 96.94% Testing: 92.85%

Figure 4.14: ICS tries a number of different ideas for each problem. It is able to make use of an arbitrary number of computers to do so. Here is ICS preparing some of the results in Chapter 5.

(see Chapter 5) in the cases where the "best" parameter is difficult to establish. Such experimentation is a bit indulgent and can lengthen the evolutionary process. To permit classifiers to be evolved in a timely fashion, the author's GP toolkit is able to distribute processing over an arbitrary number of computers on the local network or Internet, each of which can be readily converted into a GP server by downloading and running a small program. ICS evolves both single and ensemble classifiers and chooses the one which solves the problem most effectively. The author's mechanism permits different ensembles using different sub-committees and strategies to be discovered automatically.

Refinement

After all of the binary classifiers have been evolved, ICS evaluates each one and generates a class confusion matrix in order to find which kinds of classes are most easily confused (such as "0" and "O" in an OCR-like task) and initiates the evolution of "refinements", which help improve accuracy, either by reducing false positives or false negatives. Refinements are executed in order of their potential improvement to fitness. The refinement procedure continues until the user tires, or further GP runs fail to deliver any more improvements, at which point the system completes its task.

ICS nearly qualifies as a one-click classifier learning system. The user needs only to specify the training and testing data for the classifier to be evolved, select how many CPUs to use and suggest how long the classifier has to develop a solution. The classification then proceeds in a completely automated fashion.

4.4 Conclusions

This chapter has described a number of representations for performing classification using Genetic Programming and justified the authorŠs usage of two: entropy thresholding and dynamic range selection. The author has discussed the relative pros and cons of each and presented a number of ideas intended to improve the accuracy and applicability of DRS in particular. Two different approaches have been proposed: the faster partial solutions method and the slightly slower (but possibly more accurate approach) which attempts to evolve classifiers in a pseudo-intelligent manner. Although the concept behind both of these methods is to break the problem down into smaller pieces (as humans do), they go about it in different ways. Partial solutions is concerned mainly with speed while ICS concentrates on accuracy, replicability and careful refinement. Partial Solutions is a good approach for Şrapid prototypingŤ, to see whether a set of data can be solved using GP. ICS produces the equivalent of a final product, where accuracy is of the most importance.

The following chapter will describe the authorŠs Genetic Programming toolkit in more detail and explain some of the other means by which GP can be rendered into a parameter-free learning tool. The next chapter shall also seek to validate the authorŠs classification system by comparing it to an established evolutionary computation toolkit, to results obtained by other GP researchers and the best works reported by other pattern recognition researchers.

Chapter 5

Validating the Evolved Learning System

“Name the greatest of all inventors. Accidents.”
Mark Twain

Rubber in its natural form is not particularly useful. In cold conditions it is brittle and cracks, yet in hotter climates it readily melts. It was the accidental spillage of sulphur onto raw rubber by Charles Goodyear in 1839 that led to the discovery of vulcanisation, the process by which rubber can be rendered into a stable consistency[1]. Similarly, the recipe for Coca Cola was formulated inadvertently by John Pemberton while he tried to concoct a pain relief remedy[2]. In 1897, Ernest Duchesne submitted his thesis describing his discovery, again by accident, that certain moulds would kill bacteria. Although his research went largely unnoticed at the time, it was the first documented discovery of antibiotics[3]. While science certainly does not progress through accidental discovery alone, chance happenings like these can suggest avenues of exploration that had not previously been considered. The inherent randomness and abstract nature of Genetic Programming (GP) would appear to make it quite good at simulating creativity, but is this enough for it to compete with the best of human-written pattern recognition algorithms? The goal of this chapter is to make several empirical comparisons in order to answer this question.

GP has already been recognised as an automatic innovator in fields outside of pattern recognition. According to John Koza [124], whose seminal books on GP continue to dominate the literature, there are at least thirty six recorded instances where GP has produced

[1]Goodyear was unable to patent his invention and died penniless.
[2]Pemberton later sold the recipe for $900 and died penniless.
[3]Duchesne later died penniless, ironically from Tuberculosis, which his own discovery could have treated.

human competitive solutions, mainly within the domain of evolved electronics. Twenty one of these instances either infringe or duplicate functionality of previously patented 20th/21st Century inventions, and two are sufficiently novel as to be patentable in their own right.

In the latter part of this chapter, the author's GP toolkit will be compared to a standard GP implementation, then to other results reported by GP researchers in general, and finally to the results of a range of other popular classification techniques. This chapter begins, however, with a more thorough description of the author's GP system, an enumeration of its novel features, and a discussion of how a genetic learning system, often plagued by the need to discover problem-specific parameters, can be adapted into a "black-box" generic problem solver.

5.1 Building a GP System

Although the author's initial experiments made use of ECJ [6], a popular but cumbersome open-source GP toolkit, it became apparent that in order to experiment with changes to the GP system itself, it would be worthwhile to develop the GP system from scratch[4]. This toolkit (named SXGP, after its *alma mater*) is specifically designed for evolving solutions to problems involving vision.

Being practically minded, the author also sought to develop a toolkit whose evolved solutions could easily be put to work within applications. Since the objective is to develop vision systems using two or more evolved components, it is crucial that each component be easily deployed since the system cannot be evaluated purely within the confines of the GP environment. ECJ is rather difficult to integrate into other software (since each run is usually defined by a configuration file, rather than programmatically) and does not readily output individuals as for use in other programs[5]. The author's toolkit, SXGP, can be instantiated though a straightforward programming interface, and can deploy programs in various different ways so that they may be put to work within applications.

The various components that make up a Genetic Programming toolkit were described in detail in Chapter 2. There are numerous choices that one can make for each component, so two Genetic Programming implementations can actually differ quite significantly. Thus, before the comparisons commence, it is necessary to describe SXGP in more detail.

[4]Of course, it is also more satisfying than to use other peoples' code!

[5]ECJ outputs individuals using LISP syntax. The author has written an extension to output ECJ individuals in Java [125].

5.1.1 Specification

The main components of the author's system are specified in Table 5.1.

Component	Method	Where Described
Representation	Tree	Page 13
Population Builder	Ramped Half and Half	Page 15
Selection Method	Tournament Selection	Page 26
Generation Gap	Generational	Page 34
Fitness Function	Variable	Page 114
Elitism	Enabled	Page 33
Genetic Operators	Crossover and Mutation	Pages 28, 32
Population Size	Variable	Page 105
Generations	Variable	Page 105
Function Nodes	Task Dependent	Page 112
Terminal Nodes	Problem Dependent	
Learning Paradigm	Supervised	
Island Selection	Utilised	Page 27

Table 5.1: Specification of SXGP, the author's Genetic Programming Toolkit

5.1.2 Implementation Notes

SXGP is implemented in Java, a language well suited to the development of abstract tools. Java also offers excellent libraries for image acquisition, networking and multi-threading. The cross platform nature of Java makes it particularly suitable for distributing the Genetic Programming process over a number of CPUs, a feature also implemented in SXGP.

Results by a performance profiler reveal a significant amount of computational time expended by the Genetic Programming process is actually devoted to the copying of individuals, which is necessary to ensure that offspring are separate instances from their parents[6]. Although the author originally used a flexible copying operator which serialises an object into a stream and then re-instantiates it as a new instance, this was found to be

[6]Otherwise the later actions of genetic operators on parents would also affect the offspring.

a significant bottleneck. Instead, each object implements its own cloning method which permits the process to proceed much more quickly, while maintaining the requirement to perform "deep cloning"[7].

We shall look at other means of enhancing performance throughout the course of this thesis.

5.1.3 Novel Features

SXGP comprises a number of features that are novel. Each is discussed briefly below.

Strong-er Typing Depending on the set of functions used and the maximum tree size permitted, the program search space can be very large. While the essential purpose of any genetic learning system is to discover good solutions to a problem without resorting to an exhaustive evaluation of the search space, it is nonetheless desirable to ensure the search space itself is as small as possible.

One way of reducing the program space is through the use of *strong typing*, where the tree builder, which is responsible for constructing the programs (and new mutations), is constrained to connect only nodes which make semantic sense. For instance, the *LessThan* operator, which returns a boolean value, should not be supplied as an argument to the *division* operator – execution could lead to divide-by-zero errors and a rather inconsistent logic. As was mentioned in Chapter 3, Montana[17] implemented a means of strong typing for GP systems by introducing the notion of a *return type*, which would have to be matched with each node's argument types, as is the case in the strongly-typed programming languages. As an example the *add* operator should insist that both its arguments have a *numeric* return type; the operator itself will also return a *numeric* value.

However, as Montana pointed out, types such as *numeric* or *boolean* have the disadvantage of being rather *too* generic. If one is looking for a count for a loop, for instance, a floating point number is not necessarily appropriate – it is better to involve an integer. However, if one defines "integer" as a new type to accommodate this requirement, then the arithmetic operators would no longer accept it, since it cannot be both "numeric" and an "integer". Of course this shouldn't be the case. Essentially a node should be able to return more than one type, for instance to say that it returns a number that is also an integer: indicating a hierarchy of inheritance among different types. SXGP includes this

[7]Deep cloning refers to copying both an instance and all of its class members; shallow cloning will clone the instance, but will not copy class members. In GP, individuals must be completely separate instances, otherwise the genetic operators may affect more than one individual at a time.

flexibility, where each node may implement *multiple* return types. The overall return type of a node is instead defined as an *array* of different individual types. If any of these match the required, specific argument of a node, then it is deemed acceptable for addition to the tree by the tree builder. This permits the system to make use of more complex structures while producing fewer trees that don't make semantic sense.

Tree Checking Here the author's stronger-typing framework extends beyond Montana's work. Although GP is capable of evolving many means of procrastination, the stronger-typing approach still permits the creation of "useless" code that performs no function. Code may be considered useless if it is never executed, or if it always returns the same value regardless of input. The former is usually to be found in branches of if-statements whose condition is accidentally constant.

The tree builder in the author's GP toolkit attempts to avoid creating such code by following additional criteria, which may be created for specific nodes. For instance comparative nodes (*lessThan, moreThan, between, equals*) additionally insist that at least one of the child nodes in the sub-tree beneath each be some kind of image feature (a separate sub-type, based on the numeric type). This helps ensure that every sub-tree has the potential to perform some useful processing[8]. This restriction also reduces the size of the program search space.

The system also discards those programs that do not make use of features at all. GP programs, if left unchecked, may take advantage of the *a priori* probabilities of solutions and simply return the most popular!

Automatic Optimisation The author's GP system automatically optimises the best individual at the end of evolution, in order to make it more suitable for deployment in the computer vision task for which efficiency is always a criterion. After running the individual again on the training data, the system collects statistics about each node, then removes nodes that are never used and replaces nodes that always return the same value with constants.

Instant Deployment Solutions evolved by SXGP can be deployed for use in other applications immediately, in two modes:

[8]Although it is still not perfect: the lessThan function could compare a feature whose return values are in the range 0-255 to a value of 1000, which would again always return the same value. However, SXGP's automatic optimisation mechanism will remove this kind of code.

GP Tree The GP tree is used as-is, so the program can be used immediately. The SXGP tree class itself is *serializable*, which means it can take advantage of the Java mechanism to be readily saved to disk and reinstantiated later, or on different machines. All important information, including the status of the individual's program classification map, are saved. Wrapper classes permit different GP trees to be applied to different tasks in vision, such as feature detection or object classification. However, since the program is executed in an interpreted manner, the program will not run as efficiently as possible.

Compiled Java The GP tree is first converted into Java source code, then compiled automatically. This kind of program will run significantly faster (see page 157), but requires the right libraries to be linked, and for the user to have a Java Development Kit installed on their computer for compilation to proceed. The process of deploying individuals for use in vision systems is discussed in greater depth in Chapter 6.

Live Evaluation As well as providing a graphical interface to display the status of the evolutionary system, the effect of the current best GP program on images is displayed live by the user during evolution. This enables the user to assess how well the evolved solution is working and generalising, and provides an insight into how difficult a feat of image processing is: the user may subsequently tailor the training set to incorporate characteristics that apparently were not emphasised sufficiently in initial training data.

5.1.4 Parameter Choices for a Generic System

One can think of a Genetic Programming system as a whole "world" in which the birth, life and death of thousands of individuals are simulated by a series of sub-components. As such, one of the problems of developing solutions with GP is the sheer number of parameters required by each component: there are parameters for the minimum and maximum initial depth of individuals, the kind of tree builder to use, the population size, the number of generations, choice of fitness function, and so on. Moreover, it is difficult to assess the true effect of any parameter since all these components are rather inter-related. The work published by other researchers indicates that parameter tuning and experimentation is often called for, which doesn't bode well for a supposedly generic problem solver. In order to maintain the aspiration of a generic system, it is worthwhile to consider the research and theory behind different parameter choices in order to establish a set of parameters that work well on a wide range of problems. Some of these are discussed below.

Genetic Operators A standard breeding pipeline involving crossover, mutation and reproduction (copying) operators is used in SXGP: each new individual in the new population is produced from one or more "parents" using one of the above operators. The number of individuals generated by each operator is usually determined by allocating a given percentage to each. The optimal "blend" of the learning operators crossover and mutation is something not all researchers agree on. Koza himself used no mutation, preferring a 100% crossover pipeline[9]. Other researchers [126] have shown that Genetic Programming can proceed without using crossover at all, by using 100% mutation. In fact, provided that there is a suitable means of selection, and that there is some genetic search proceeding, the GP process is remarkably robust[10].

Work by Luke and Spector [127] aimed to establish a way forward through systematic comparisons of evolution using different proportions of crossover and mutation. They found that, in general, evolution using only crossover was slightly more successful than evolution using only mutation, but that the difference was not particularly substantial. Mutation was shown to be slightly advantageous in certain situations where the population size was small. In general the authors could not find an ideal blend of crossover and mutation that would yield better results consistently. From the author's point of view this is a good thing – it isn't necessary to tune the operator rates to different problems.

Given these observations, the author used a reasonably "standard" set of values for all tasks, making use of both operators, in which crossover produces about 75% of the new population; mutation 20%; the remainder obtained simply by copying. Making use of both operators is desirable, at least from a practical point of view, since each has its own specific advantages. Crossover helps to exchange useful building blocks between good parents, while mutation can help maintain diversity by re-introducing sub-trees that might otherwise fall out of the population.

Tournament Size Genetic Programming is distinguished from random search by its bias toward more successful individuals, which is achieved through the evaluation of individuals' performance (see Section 5.2) and subsequent selection of "good" individuals to produce next generation. The most widely used selection technique, tournament selection,

[9]Koza was keen to prove that GP learning was not random search.

[10]The author's own toolkit, SXGP, for several months featured a silent bug in which crossover-generated individuals were not added to the new population. The problem was only noticed later — the bug had had little apparent effect on the results by GP, much to the embarrassment of the author.

picks t individuals at random from the population to form a “tournament”, from which the individual with the best fitness score is selected. Larger tournament sizes t will increase the selective pressure, albeit at the expense of variability in the subsequent generation. Other selection methods were discussed, starting from page 23.

During their comprehensive analysis of a series of selection methods, Blickle and Thiele [128] concluded that Tournament Selection produced the smallest loss in diversity and the highest selection variance in comparison to Fitness-Proportional and Ranking selection. For this reason, and the reasons presented above, Tournament Selection was chosen as the *de facto* selection method in the author’s Genetic Programming environment.

Motoki [129] calculated that, given a population of size N, and tournament size t, the expected loss of diversity D_T in tournament selection would be:

$$D_T = \frac{1}{N}\sum_{k=1}^{N}\left(1 - \frac{k^t - (k-1)^t}{N^t}\right)^N$$

Interestingly, this formula predicts that the loss of diversity initially grows rapidly but becomes relatively static in populations with more than 8 individuals. This means that we can select a value of t which should not require specific tuning for each individual experiment, especially if using a fixed population size. When $t = 20$, the formula predicts that around 82% of the diversity of the population is lost in any given generation, which may significantly affect the ability of the evolutionary algorithm to explore the search space. At the lowest practical value, when $t = 2$, the loss is about 43% per generation. Low selection pressures are generally suggested in GP practice and successful results have been achieved using low values [31].

But how low should the selection pressure be? Although the Genetic Algorithm community has traditionally used a tournament size $t = 2$ which preserves as much diversity as possible within the population while maintaining some degree of selective pressure, GP researchers generally use $t = 7$ as this is thought to instill higher selective pressure on the population, tending to improve the learning rate during the limited time of a GP run[11]. However, higher values of t will cause the same parents to be selected more often, which reduces the variability within the population and increases the redundancy. An experiment measuring the population redundancy[12] is shown in Figure 5.1, which shows

[11]Genetic Algorithms are potentially faster to run than GP, so can be indulged in evaluating more generations.

[12]Redundancy is measured by calculating the number of individuals that are duplicates of another member in the population.

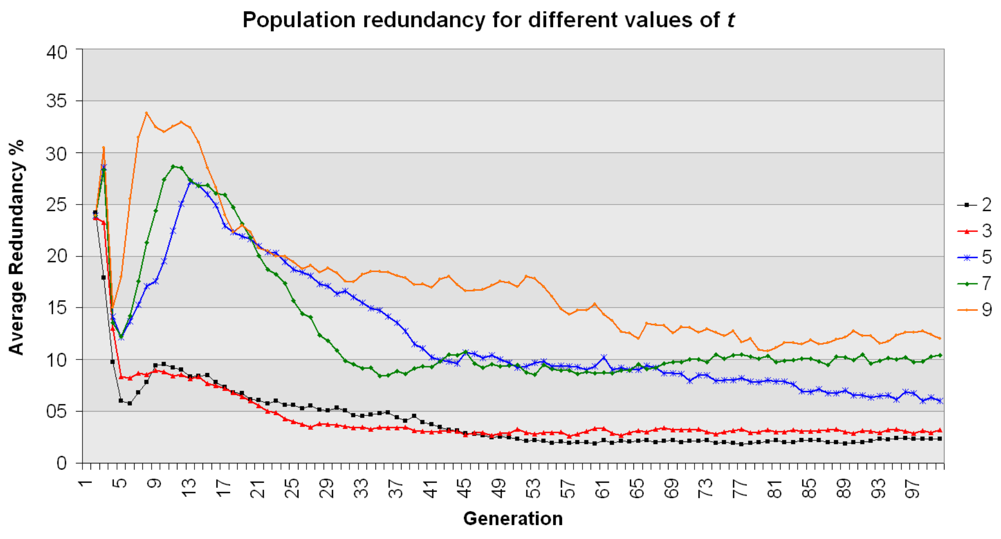


Figure 5.1: The average redundancy of a population over the course of 50 generations, for different tournament sizes.

a substantial difference in population redundancy between $t = 2$ selection and $t = 7$ selection.

The author has conducted a number of comparative experiments using the range of tournament sizes described above, the result of which is that no significant difference could be detected in training accuracy, evolution time or average population size. Neither was it observed that the performance of runs with different tournament sizes was any more variable for different values of t. As it is difficult to decide exactly which is the best parameter, one of the author's classification schemes errs on the safe side by using different sizes for different runs, and in general SXGP uses the more obvious compromise: a tournament size of $t = 5$.

Population Size The computational expense of an evolutionary run is usually proportional to the product of two parameters which specify the breadth and length of the search: namely the population size and generation count. Populations are used by evolutionary learning algorithms to store a diverse base of knowledge that can be built upon. Although the degree of selectivity employed has a significant effect on the variability within a population, such variability can only be stored within a population of solutions of sufficient size. While there is clearly a lower bound for the population size, there is no practical upper bound, since the search space is potentially enormous.

An impression of the size of the search space can be gleaned through a brief example.

If one imagines a GP system which uses n functions each of arity (number of arguments taken by the function) 2, and maximum tree depth D, whose individuals are generated by a FULL tree builder (see page 15), the search space, or number of potential programs is:

$$size = T + \sum_{i=1}^{D-1} \left(F^{2^i} - 1\right) \times T^{2^i}$$

From the exponential nature of this calculation, it can be seen that modest increases in the number of functions or the tree depth will quickly yield spectacularly large search spaces. The addition of continuous random constants can increase the size of the search space to even more gigantic proportions, which in any case is such that the GP system will never be able to explore it comprehensively. Given that the population size can only ever be a tiny fraction of the size of the search space, its choice might be considered relatively arbitrary: it doesn't appear necessary to tune for individual problems. Many GP researchers settle on a population size of 500, although higher values are used in more modern research. Nonetheless, the use of the ramped-half-and-half tree builder (see page 15) introduces a significant level of duplication into the population (for reasons explained later), which is alleviated in SXGP using fitness caching (see page 155).

Aside from problems of duplication, genetic populations have a tendency to become dominated by a particular solution, a phenomenon described as premature convergence. Various "convergence manipulation" protocols exist to avoid the homogenisation of a population, including island selection, discussed on page 27, but each introduces extra parameters to the system. Murphy and Ryan [130] suggested the use of a different tournament selection algorithm which would repel individuals which shared a similar ancestry. The algorithm starts by selecting an individual at random[13], then searching for an appropriate partner to participate in crossover in a "repulsion tournament"[14]. The partner chosen is the one with the fewest shared ancestors with the first individual. Thus the crossover operator is forced to continue exploring previously uncharted areas of the search space, while ensuring the population maintains a higher degree of diversity. Hereditary repulsion was implemented on the author's toolkit; comparative results are shown in Figure 5.2, which shows that the technique can both decrease and increase the average error.

[13]This is the protocol by Murphy and Ryan[130], in SXGP it starts by selecting an individual using tournament selection as normal, which seems to be rather more sensible.

[14]This author feels the chosen term for this technique, "hereditary repulsion", is something of a missed opportunity; "incest police" is rather more enjoyable terminology.

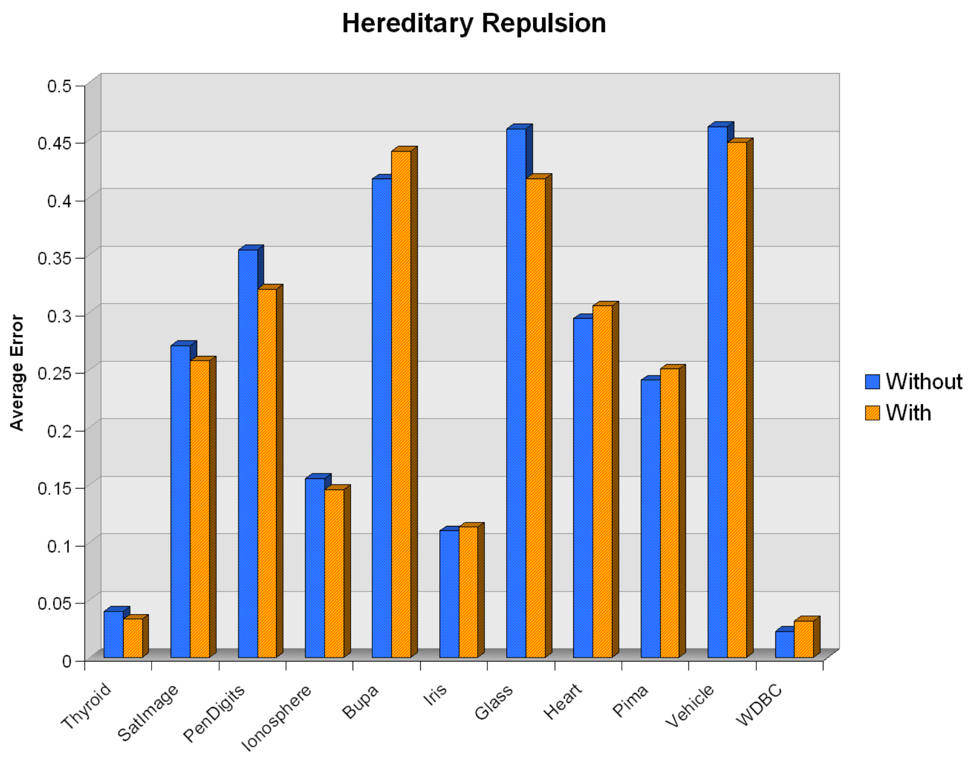


Figure 5.2: The effect of Murphy and Ryan's Hereditary Repulsion technique on average classifier accuracy.

5.1.5 Tree Builder Parameter Choices

As we saw in Chapter 2, there are two means of producing trees in Genetic Programming, the *FULL* builder, which generates "full" trees, in which every branch extends to the maximum depth D, and the *GROW* builder, which extends branches until the point where a leaf node terminates it. Because the *FULL* builder limits the number of tree structures that can be used, and the *GROW* builder will not always produce trees of the full size, most GP practitioners use the ramped-half-and-half algorithm to generate the population, which uses each technique half the time, usually for each of a range of depths $1, 2, ..., D$. The author identifies two problems with this technique, which are dealt with in this section.

The first problem relates to redundancy, or the presence of identical individuals within a population. Redundancy reduces the variability within the population, which may affect the rate of learning, and prevents the search space from being explored as thoroughly as possible.

Here is a simple calculation. Suppose the *FULL* tree builder is called upon to construct a tree of depth D, using F different function nodes and T different terminal nodes. We can calculate how many leaves (terminals) the tree will use easily (assuming that each node has arity 2):

$$terminals = 2^D$$

The total number of non-leaf (function) nodes in a binary tree will be:

$$functions = \sum_{d=0}^{D-1} 2^d$$

The number of "parent" functions at depth $D-1$, who each have two terminals as children:

$$parentfunctions = 2^{D-1}$$

The number of unique combinations for a function with two *function* nodes as children is[15]:

$$c_f = F^2 - \sum_{i=1}^{F-1} i$$

And the number of unique combinations for a function with two *terminal* nodes is:

[15]It is assumed that the arguments to each function are commutative, so certain combinations can be removed

$$c_t = T^2 - \sum_{i=1}^{T-1} i$$

Thus, a simple estimate for the number of possible trees is:

$$numtrees = F \left(\prod_{d=0}^{D-2} 2^d c_f \right) \left(2^{D-1} c_t \right)$$

Given a set of 6 features (as is the case for the BUPA dataset), and 4 functions, often used by GP programs, the number of possible trees is:

D	0	1	2	3	4	5
Trees	4	84	1,680	67,200	5,376,000	860,160,000

Of interest is the relatively small number of trees for low values of D which gives rise to a particular observation. The ramped-half-and-half technique makes an even allocation of depths, such that each depth is used $1/D$ times. To generate a population of size 500, with six depth levels, would mean that roughly 83 individuals are produced at each level. If we assume for a moment that all individuals are produced using the FULL builder, then based on the table above, it is clear that there are many more individuals allocated to depth 0 than are needed. Such a strategy would guarantee the initial population was at least 15.8% redundant. In fact, the redundancy will often climb higher than this early on in evolution. The reader may be thinking that it is rather implausible that the minimum tree depth would ever be set to 0, creating individuals composed of just a single node – but both ECJ and Lil-GP, another popular open source toolkit, do so; single terminals, in combination with DRS classifiers can be useful, not least when forming replacements for sub-trees during mutation.

The author's adaptation for the tree builder is to ensure that it does not allocate any more trees to a particular depth than are calculated by the above formula. "Spare" individuals are allocated to other depths. This helps to ensure that the population will be less redundant. The results comparing the "standard" tree builder, which allocates each depth the same number of individuals, and the author's "weighted" tree builder, which allocates depths more reasonably, are shown in Figure 5.3. Results shown are the average over 25 runs on the BUPA classification problem and show the average population redundancy at each generation. They show a marked difference in population redundancy: the author's tree builder significantly reduces it. The reader may recall that the tree builder is used to generate the initial population and then to generate sub-trees for mutation. Although mutation is only used for 20% of reproduction events, and not all mutation involves sub-tree

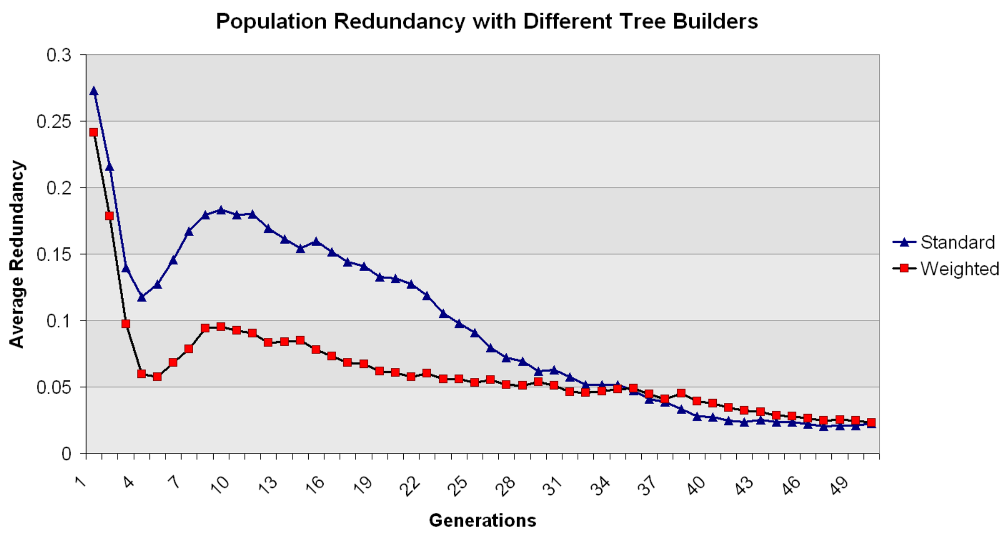


Figure 5.3: A comparison between the author's weighted tree builder, and the standard tree builder. The authors builder yields populations with substantially less redundancy over the course of 50 generations.

mutation, the choice of tree builder nonetheless has a significant effect on the redundancy of the population.

5.1.6 Performance Considerations

The nature of many evolutionary learning systems is to produce a lot of solutions then throw the majority of them away. Although this process is key to learning by simulated natural selection, it is also inherently wasteful. As a stochastic process, it is usually wise to run the genetic learning system several times in order to estimate the true performance, which introduces further computational expense. Although the programs evolved can be very concise, the computational demands of GP learning, especially on computer vision problems, is always a concern – and possibly one of the reasons members of the computer vision community do not often use GP! By contrast, techniques such as XCS [50] aim to evolve and improve a single system through less blunt reinforcement and thus may converge upon a solution more quickly. The nature of the genetic operators themselves is also of concern. The process of crossover and mutation, which drive the search through the program space, can become more destructive than constructive.

Although GP will often be more processor intensive than other techniques, there are ways in which one can preserve the learning capabilities of GP while significantly reducing the time required to evaluate a given number of individuals. In this section we shall provide a brief performance comparison between SXGP and ECJ, an established evolutionary computation system.

A common issue in GP which has significant impact on performance is the presence of useless code segments within individuals, known as "code bloat", that reduce the efficiency of the learning process. As the amount of useless code increases, so it becomes increasingly less likely that the GP operators will have a noticeable effect, since they end up swapping one unused code fragment for another. Left unchecked, the process of learning can effectively cease.

Various techniques are used to reduce bloat, including parsimony pressure and dynamic maximum tree depth [131], which penalise or remove large individuals respectively. However, for a generic system, it is difficult to quantify what is "large", so an alternative solution was employed.

Elitism, a commonly-used technique which copies the n best individuals in the population directly into the next generation, is used instead. Studies have shown that evolution with elitism enabled yields smaller average population sizes [46]. One can argue that individuals that are padded out with unused, and thus expendable, pieces of code are more likely to survive the processes of crossover and mutation intact: so there is a selective pressure which encourages the proliferation of code bloat. The direct copying into the next generation removes some of this selective pressure that encourages individuals to protect themselves with unused code.

SXGP's tournament selection mechanism will also choose smaller individuals in the case that two equally-fit programs are competing for the same tournament. This is a form of multi-objective optimisation analagous to lexicographic ordering, described in Chapter 2. The tree checking and strong-er typing functions in SXGP also prevent bloat code from becoming prevalent. Since SXGP can automatically remove many forms of inactive code, additional trees that have a negative effect on the individual's ability are quickly eliminated.

The combined effect of these additions is presented in Figure 5.4, which compares the average size of best-of-run programs for both SXGP and ECJ following evolutionary runs of 50 generations. Although the *max-tree-depth* parameter was set to the same value in both toolkits (all other parameters were similarly matched), ECJ tends to produce significantly larger individuals than does SXGP[16], whose best-of-run programs are typically a third of the size of ECJ programs. Consequentially, evaluating 500 individuals over the course of

[16]In addition, the SXGP equivalent is generally slightly fitter.

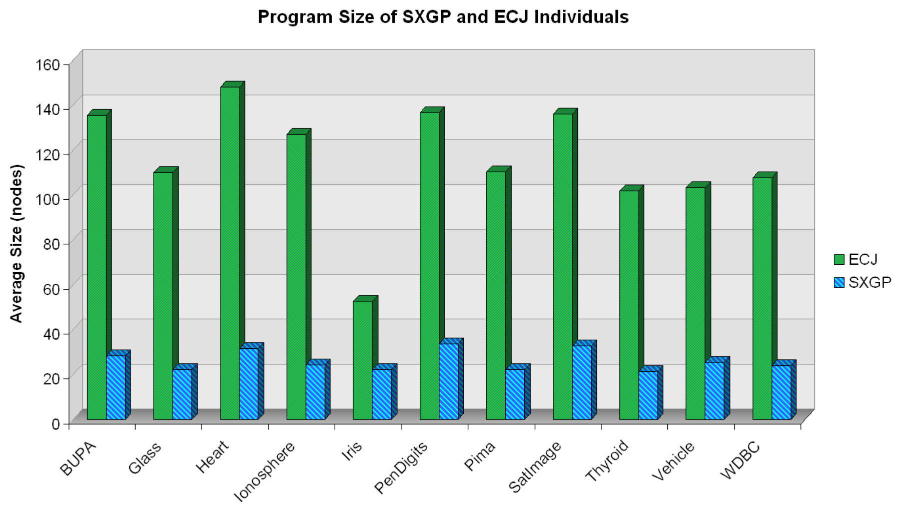


Figure 5.4: Comparison between best-of-run program sizes (in terms of number of nodes) following 50 generations of evolution for SXGP and ECJ.

50 generations takes about three times longer in ECJ.

5.1.7 Function Set

Thus far we have not discussed in much detail the *function set* to be used by defined for the GP system. The function set is a pool of nodes that the tree builder can use when generating problems. Functions, be they mathematical, logical or boolean in nature, are essential for combining terminals together in order to produce a one-dimensional output. It is typical for GP researchers to include their chosen function set in their published work, and indeed there are variations, although many restrict the function set quite tightly to purely mathematical operators. The author suggests that this limits the system to evolving linear operations. The most common non-linear function in use is the $IF()$ function (example on p 17), which in turn requires boolean operators such as $AND, OR, LESSTHAN, MORETHAN$ to form its condition.

One can embellish the function set with any number of additional functions, including trigonometric functions, statistical functions, and other non-linear operators. An interesting question is whether the use of such functions yield better individuals. An experiment was conducted to assess the average accuracy of individuals produced from two different function sets. The first “limited” set used only the four arithmetic operators $+, -, \times, \div$, and the $IF()$ and boolean operators mentioned above; the second “extended” dataset added a variety of additional operators, including trigonometric functions $\sin, \cos, \tan$, non-linear

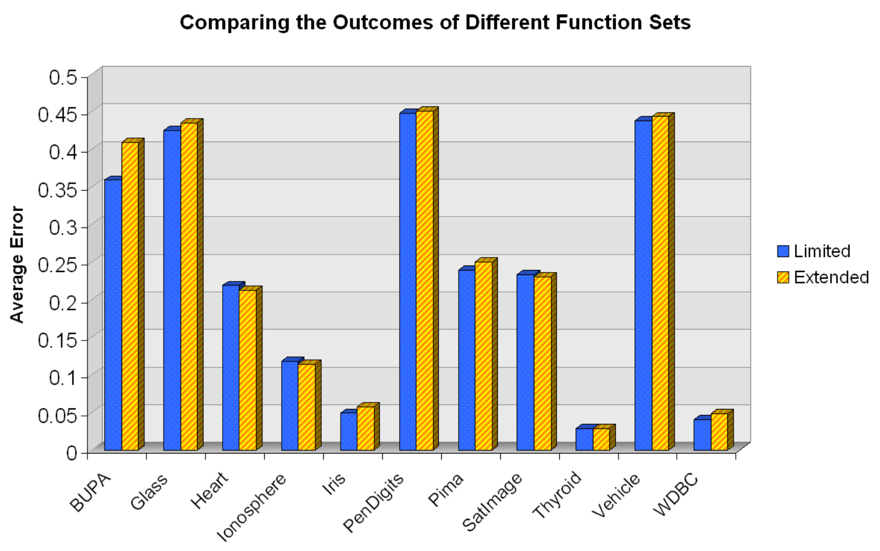


Figure 5.5: Average errors for individuals evolved by GP systems with "limited" and "extended" function sets.

functions max, min and additional mathematical functions $exp, ln, squared, cubed$. GP classifiers were evolved using each function set 50 times apiece; the results are shown in Figure 5.5.

The figure shows the average error for two different function sets. For the eleven datasets, the difference in error was insignificant on eight occasions, indicating once again that GP is able to select appropriate functions automatically in most situations. For the three datasets for which there was a significant difference (BUPA, Pima, WDBC) however, the extended dataset was found to perform worse on each occasion. In each case the fitness on training data was also affected, indicating that the additional functions do not hamper the classifier's ability to generalise, but rather the system's learning rate. On the basis of this experiment, it would appear that any advantage offered by these more complex operators is offset by an increase in the size of the search space. The author's generic system therefore makes use of the limited function set, which appears to yield better results in most situations.

5.1.8 Terminal Set

The terminal set includes all the attributes within the feature vector that describes a particular object. In the case of the public datasets used throughout this chapter, the features are predefined, ranging in number from 4 – 36. The generation of the features themselves from images is discussed in the following two chapters. In any case, feature vectors are provided to the GP system through a common dataset interface, which permits samples loaded from files or databases to be loaded, or those taken directly from images. Each feature is packaged up as a terminal node type, and becomess a member of the terminal set.

In addition, the terminal set includes a number of so-called ephemeral random constants (ERCs), constants whose values are set at the time of tree-creation and subsequently remain fixed. These terminals give GP the opportunity to perform scaling, other arithmetic operations, or comparisons. To ensure that comparisons are meaningful, the constants are matched using the author's stronger-typing technique (see page 100): features which return, say, values between 0 and 1 are only compared with constants which return values within a similar range.

5.2 The Evaluation of Fitness

We shall move our discussion of parameters onto the evaluation of fitness (discussed in Chapter 2, page 18), which merits its own section. The computation of fitness is the primary means by which the selection algorithms can differentiate between different programs, and is therefore very important.

The appropriate fitness function to use in computer vision is often task dependent. For general classification tasks, it is usually the case that each error is equally costly, so a straightforward fitness function that measures simply the number of mistakes made by a given program on a training set can yield good results. It is sometimes the case that certain classes are more prevalent in training data than others, and therefore receive more attention than others. This is not necessarily a problem provided the training data is a true representation of the world. If not, one strategy is to calculate fitness per-class, and choose the average or indeed lowest fitness. The former gives each class an equal weighting and the latter encourages "generalists".

Binary detection tasks, by contrast, may be considered as multi-objective optimisation problems: ideal solutions must both be able to detect as many of the objects of interest as possible (be sensitive), and not detect erroneously other objects (be specific). In face detection, for instance, there are many more non-face objects than there are face objects, so it is beneficial to take the relative occurrence of each class into account in the fitness function.

The two criteria can be aggregated into a single fitness function using some form of weighted sum. One example is that used by Roberts & Howard [20], who incorporated two coefficients α and β to weight the system's sensitivity to the different types of error:

$$error_2 = \frac{\alpha TP}{\alpha N_t + \beta FP}$$

Where TP is the number of correctly classified samples (true positives), FP is the number of incorrectly classified samples (false positives), and N_t is the number of "true" samples in the training set. Oddly, this formula calculates $\alpha TP/N$ as a percentage, but also adds FP to the denominator, meaning that the effect of α relative to β is dependent on the total number of "false" samples. It occurs to the author that the following calculation makes more sense:

$$fitness_3 = \alpha \frac{FN}{totalN} + \beta \frac{FP}{totalP}$$

Where $totalN$ is the total number of negative samples, and $totalP$ is the total number of positive samples. In $fitness_3$ the fitness is composed of two parts, the first relating to the total weight of false negatives and the second relating to the total weight of false positives. The relative importance of each kind of mistake is encoded using an appropriate $\alpha : \beta$ ratio. The allocation of weights, however, introduces additional parameters to the evolutionary system; often the approach is to try different combinations of each such that the classifier's performance can be plotted using an ROC curve. This is one means of finding solutions distributed along or nearby the Pareto front.

From a more practical point of view, however, we are really interested in just a single solution to detect objects in images. The approach taken by researchers is usually to develop a rough detector, which is highly sensitive but yields quite a few false positives, whose output is then passed to a finer detector, which makes the final decision. During the course of this study, the author experimented with various other fitness approaches, admittedly not always with much degree of success. Some of which are briefly mentioned below.

5.2.1 Sample Weighting

Having considered the relative importances of different kinds of mistake, and whether one may wish to weight samples collectively, it is also worth considering whether individual samples should be allocated different weights according to their *individual* difficulty, since some samples will be easier to classify than others. By weighting the samples that appear more difficult (that are misclassified most often), it may be possible to improve performance of the classifier, or at least ensure that the system does not become mired in local minima.

A useful benchmark when considering this sort of approach is AdaBoost, by Freund and Shapire [92]. Adaboost is a meta learning algorithm which develops classifier ensembles over the course of several learning sessions. The weights associated with different training data samples are adjusted to influence the learning system to solve more difficult training data. Weights are adjusted only following the complete learning of one classifier, which makes it quite slow to generate accurate classifier ensembles.

The author investigated a technique whereby the weights of different samples could be adjusted *during* a single evolutionary run. As usual, the fitness of any individual is calculated by adding the weights of all incorrectly classified samples. After each generation, the weights of each sample is adjusted, such that unsolved samples increase their weight while solved samples have their weight reduced in proportion to how well the classifier solved the sample set as a whole. By implementing a dynamic fitness function, it is hypothesised that the individual would be less susceptible to becoming stuck in local minima. However, experiments revealed that this approach did not yield an improved rate of learning; classifiers were typically less accurate than those developed using a standard fitness function.

5.2.2 Encouraging Generalisation Ability

The fitness functions mentioned thus far have concentrated on the classifier's error on a given training set. However, a common problem in non-parametric classification is over-fitting, where a classifier learns the behaviour of a specific set of data set so well that its ability to work on novel data is compromised. It is desirable, therefore, to evolve classifiers that can learn to *generalise*. As has been observed elsewhere in this thesis (see page 73), different techniques may produce a lower fitness score on training data but nonetheless outperform the fitter classifier on test data. Given a set of individuals evolved by a learning function, choosing the best of them according to training data error alone is therefore somewhat insufficient.

The usual solution is to reserve a portion of the training set for the purpose of validating the classifier; the performance on the validation set gives a better indication of the classifier's ability to generalise. Of course, once the validation set has been used for this purpose it is no longer an unbiased estimator of performance, so a completely unseen test set is also employed to make the final assessment of a classifier's capability.

Training Set Splitting Of course, the use of a validation set takes away a proportion of samples from the training set that would otherwise be used for training, which itself may have an effect on the generalisation ability of the classifier.

The author devised an alternative method, intended to gain information about the classifier's generalisation ability without sacrificing any data, and without resorting to computationally costly methods such as cross-validation or jack-knifing.

The technique involved splitting the training data into two sets, which are evaluated separately. The classifier was evaluated on each set, with the sum of errors yielding the training fitness as before. To this error, the *difference* between errors on the two sets is also added, thus imposing a fitness pressure toward *consistency*. However, it was not possible to detect a significant improvement; indeed this algorithm sometimes worsened the performance of the classifier both on training and test datasets, perhaps reflecting that the two sub-datasets could not be assumed to be directly comparable. This experiment reveals once more the perils of tinkering with fitness functions!

The Validation Dataset Discarding the author's penchant for experimentation for one moment, the conventional way to use validation sets is to leave them untouched until the evolution process has completed, then to use them as estimators of the classifier's performance on unseen data. This allows the generalisation capability of the classifier to be assessed without affecting the delicate fitness function. As GP evolution sessions typically involve producing a number of individuals during the course of many runs[17], their performance on validation data, as opposed to their training fitness, can be used to select the most promising individual. A similar process can be used when trying out classifiers produced by classifier fusion.

One significant advantage of using validation data in this way is that the GP evolution time is shortened, since the training set is reduced in size. The proportion of data to use for validation sets poses a dilemma: choosing too high a proportion may make the training set too small, while choosing too low a proportion may make the validation set unreliable. However, results from the author's experiments with validation results did not reveal any significant advantage of using validation sets.

[17]This is done to avoid the effects of initial populations. The GP environment should always converge on an equally fit solution given enough time, but undertaking multiple runs is the most reliable way to ensure that a particular result is representative.

5.3 Comparisons

Having discussed the GP toolkit in more detail, we now move onto some empirical comparisons, using a set of binary and multi-class benchmark datasets (summarised on page xi).

5.3.1 Dataset Interface

Of the selection of eleven datasets used throughout this thesis, four include their own separate test set permitting validation on unseen data, which provides an indication of a given classifier's ability to generalise. The rest are validated using 10-fold cross validation, a standard technique for estimating performance on unseen data. Cross validation is usually performed by splitting the dataset into 10 sub-sets, from which one is chosen as a test set, with the rest used for training. The learning process is invoked for every test/training combination, so the overall training error can be averaged, and the test error estimated. The author's implementation of k-fold cross validation uses a *stratified* selection policy to generate the folds, which ensures that the relative class distributions within each sub-set are kept as similar to that of the main set as possible. Some benchmarks, such as the Vehicle dataset, include pre-defined folds; this is also accommodated. The author's software interface permits each type of dataset to be presented to the classification system, whether it be one of the public training datasets, or a dataset generated from images.

5.3.2 Comparing with ECJ

Before going on to evaluate SXGP in a more general sense, it is necessary to compare it to a benchmark GP implementation. For this comparison, the popular ECJ toolkit [6] was chosen, which at time of writing was available in its 18th version. Like the author's own toolkit, ECJ is implemented in Java, so comparisons of performance are meaningful. ECJ's default implementation is, where appropriate, based on Koza's original specification, making it a suitable benchmark. Finally, and despite a rather complex architecture, ECJ is surprisingly efficient when it comes to evaluating programs – so much so that this author had to expend some considerable time on optimising his own toolkit first!

In the experiments each toolkit was run on the same datasets. In order to make the comparison as meaningful as possible, most of the novel features in SXGP were deactivated such that the two toolkits could compete on an equal footing. As far as possible, equivalent problem definitions were coded using each toolkit, including identical node

Dataset	ECJ Accuracy	SXGP Accuracy
BUPA	60.6 ± 2.4	**62.4** ± 2.6
Glass	57.1 ± 2.3	58.3 ± 3.1
Heart	75.9 ± 1.9	**77.7** ± 1.8
Ionosphere	88.0 ± 1.5	**89.9** ± 1.4
Iris	89.1 ± 3.0	**94.6** ± 1.3
PenDigits	59.2 ± 3.2	59.5 ± 2.6
Pima	74.3 ± 0.8	**74.7** ± 0.8
SatImage	75.5 ± 2.7	74.8 ± 2.3
Thyroid	97.1 ± 0.8	**97.7** ± 0.3
Vehicle	60.6 ± 1.4	60.5 ±1.7
WDBC	94.3 ± 0.8	**95.9** ± 0.5

Table 5.2: Toolkit Comparison: Average accuracy of DRS2 classifiers developed by SXGP and ECJ v18 on various datasets. Averages from 25 runs, ± value indicates the standard deviation. **Bold** text indicates a statistically significant better result.

functionality, identical parameters and an identical fitness-function. The representation in each case was a multi-class DRS classifier.

Twenty five classifiers were evolved by each toolkit over the course of 25 runs, each lasting for fifty generations. The average test fitness of these classifiers on a series of datasets is presented in Table 5.2, and graphically in Figure 5.6. The difference between classifiers, and the results of an independent samples T-Test are also shown. The results show that SXGP is able to compete effectively with ECJ: SXGP produces the better classifiers on every occasion where the difference between classifiers' average fitness is significant.

It is interesting to question why ECJ's performance is not more equivalent with SXGP, given the similarity of the problem implementations and otherwise identical parameters. One answer may be found in the average program size of individuals evolved by ECJ and SXGP respectively, already presented in Figure 5.4, which shows that ECJ produces substantially larger individuals most of the time. Since SXGP can achieve equivalent or better solutions with substantially smaller individuals, it is fair to say that the additional

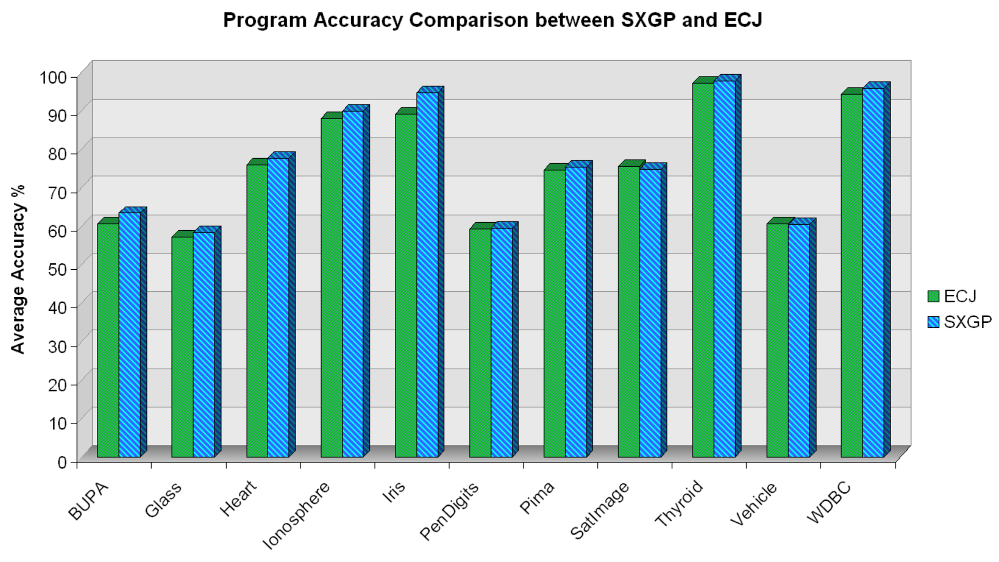


Figure 5.6: Comparison between best-of-run program test errors after 50 generations between SXGP and ECJ.

size is a result of bloat, as opposed to useful code. As well as reducing the efficiency of GP learning, bloat can prevent learning from taking place, by turning the GP system into an automatic means of swapping useless code fragments.

These results suggest that the author's toolkit is comparable to a standard and widely used Genetic Programming toolkit, which has been under continuous development for the best part of a decade[18]. This experiment justifies the author's decision to use SXGP exclusively as the GP toolkit in this thesis.

5.3.3 Comparing to Other GP Results

In the second process of validation, the author's GP Toolkit was compared to published results obtained by other GP researchers in order to establish whether the author's ICS classifier is competitive with the state of the art in Genetic Programming in a more general sense. SXGP was put to work on certain public datasets, for which figures had already been published by other GP researchers. The results of the experiments are presented in Table 5.3.

The results in Table 5.3 show that the author's Genetic Programming toolkit consistently produces classifiers that are competitive with other published results, although as

[18]This particular comparison is valid only for problems of this type, using GP: ECJ does support considerably more evolutionary learning paradigms than does SXGP.

Data Set	SXGP(ICS)	Loveard[132]	Chien [133]	Bot[113]	Muni[115]	Folino [134]
BUPA	**71.3** ± 1.6	69.2 ± 1.6			-	
Heart	**84.8** ± 1.7				-	
Glass	**67.3** ± 2.0			64.1 ± 9.1	-	
Ionosphere	**95.7** ± 0.8		92.8 ± 2.3	90.2 ± 5.5	-	
Iris	**98.7** ± 0.1		95.3 ± 1.0		**98.7** ± 0.0	
PenDigits	**92.0** ± 2.3				-	83.1
Pima	**76.5** ± 0.8	75.8 ± 0.8		71.7 ± 5.0	-	
SatImage	**87.8** ± 0.5	80.7 ± 1.3			-	81.4
Thyroid	**98.9** ± 0.3	97.6 ± 0.2			-	
Vehicle	73.0 ± 3.1	62.4 ± 2.9	**75.3** ± 2.4		61.8 ± 0.1	
WDBC	97.1 ± 0.1	96.4 ± 0.2	-		**97.2** ± 0.0	

Table 5.3: Results compared to those published by other GP researchers. Results in **bold** are the most accurate.

usual there is no single best technique for every dataset. The author's work is based most closely upon the work by Loveard and Ciesielski [132]; it can be seen that the author's toolkit delivers significantly improved results over those published by Loveard.

5.3.4 Comparing to Other Classification Techniques

In this final comparison, the author's toolkit is compared to other techniques from *outside* the Genetic Programming community in order to establish the extent to which evolved classifiers are competitive with the results of human-written learning algorithms. The comparisons are performed once again using publicly available datasets. This has already been done in the past: a previous study by Lim [135] compared 33 different classification algorithms on a series of public datasets. Equivalent results using Genetic Programming were later added [132]. With the algorithms compared in rank order, one particular GP representation (DRS, discussed in Chapter 4) was found to be reasonably competitive with the 33 other techniques at solving binary problems (ranking as high as fourth), but in general the GP programs' rankings were mediocre.

The study by Lim was dominated by a number of decision tree algorithms; the results here also include comparisons to other, more recent classifiers. The algorithms compared here include Support Vector Machines (SVM), Multi layer Perceptron Neural Networks (MLP), the C4.5 Decision Tree Algorithm, and various implementations of k-Nearest-Neighbour algorithm (k-NN). The best results found by the author for each dataset, which come from a variety of different sources, are presented in Table 5.4.

Data Set	SXGP(ICS)	SVM	MLP	C4.5	k-NN
BUPA	71.3	76.1	73.1	68.1	64.9
Heart	84.8	87.4	82.0	78.9	83.8
Glass	67.3	68.6	75.2	68.2	72.0
Ionosphere	95.7	93.2	96.0	94.9	98.7
Iris	98.7	98.0	96.0	95.3	95.7
PenDigits	92.0	97.5	93.4	96.6	-
Pima	76.5	77.2	76.4	73.0	76.7
SatImage	87.8	88.4	91.0	86.3	90.9
Thyroid	98.9	96.1	99.3	92.6	97.9
Vehicle	73.0	79.0	79.3	73.4	72.8
WDBC	97.1	97.2	96.7	94.7	97.1
Average	85.7	87.1	87.1	83.8	85.5

Table 5.4: SXGP versus Other Techniques on 11 Public Datasets.

As well as confirming the phenomenon that there is no such thing as a *best* classifier that consistently outperforms other algorithms, the results show that SXGP is competitive with a range of other techniques from outside the realm of evolutionary computation on a series of different problems. It should be noted that the same GP implementation with identical parameters was used to derive all of the SXGP results.

The Vehicle dataset, in which silhouettes of four different vehicle types must be established, is difficult to solve. This is not to say that the problem itself is intractable – our brains can differentiate all kinds of silhouettes accurately. Rather, it is the limitations in the attributes chosen to describe each silhouette that make the problem difficult since a certain amount of important information is lost. Many of the UCI datasets were constructed from a finite set of measurements taken during the course of manual experimentation. However a computer can take literally thousands of measurements from a given image with little effort, leaving one spoiled for choice. It is difficult to decide which attributes would aid accurate classification and which would be irrelevant. In fact, irrelevant attributes may actually decrease the performance of some classification algorithms, such as k-NN. Different feature sets which may facilitate classification will be investigated in Chapter 6.

5.4 A Discussion

Before moving onto the vision aspects of this thesis, the author shall complete this chapter with a short discussion of classification in general.

The many classification algorithms can ordinarily be split into two groups: parametric and non-parametric. The parametric methods generally aim to identify a set of coefficients, one for each variable, that express a combination of features whose output Y can predict a certain event or class. Techniques such as Fisher's Linear Discriminant [136] rotate the axes in order to find a linear subspace that separates the classes in a single dimension. Since Y is considered a probability, its value can be combined with prior probabilities using Bayes' theorem to obtain an answer that takes into account the relative abundances of different classes. Although the discriminant methods are intuitive, like all statistical techniques they are reliant upon on a number of rigid assumptions, such as feature independence, normal distributions or homoscedasticity, which rarely hold true for real world data.

A more pressing problem is that this kind of approach cannot accurately model non-linear relationships in data. The so-called Generalised Linear Model [137] is an extension which permits models to work with non normally-distributed data distributions, particularly those with an exponential distribution, using some form of link function. Of these, Logistic Regression, which also places fewer assumptions on the data, is one of the most extensively used techniques for parametric classification.

The advantage of the *non-parametric* methods is that they do not place arbitrary restrictions on data as they are not always based on statistical theory. They usually require more training data in order to produce accurate solutions and often take longer to train than do the simpler statistical methods.

One popular family of non-parametric techniques are the recursive partitioning algorithms, or decision-tree algorithms, of which the most well known is probably Quinlan's C4.5 [110]. The objective of the decision tree algorithm is to recursively split the dataset into two or more increasingly homogenous subgroups in order to improve the classification of a target variable. Various techniques exist to choose the point at which the data is split, but the general idea is that the root node is split by the variable that best divides the samples and so on.

Although decision tree algorithms generally exhibit good performance as classifiers, and have the advantage of easy interpretation, they also suffer from two drawbacks. The first, more theoretical, drawback is that generally the splits can only be performed orthogonally with respect to the features, so the regions in space cut out by decision tree algorithms are hyper-rectangles. The algorithm may perform badly if the decision planes are actually diagonal (or some other shape!). The second problem is that they are prone to over-fitting, which occurs naturally if the splitting process is permitted to continue unchecked, although post-processing "pruning" techniques are designed to alleviate this problem.

The most significant advantage of the non-parametric techniques is that they can develop arbitrarily complex models and thus are able to outperform parametric statistical techniques, provided they do not over-fit the data. Parametric techniques, which are usually fast to execute, may be useful when deciding which variables are important and which are irrelevant. The author shall make use of this in experiments in the following chapters.

Another non-parametric technique, the k-Nearest-Neighbour algorithm, is an example of instance-based learning. Novel feature vectors are compared to a database of the training data each according to some distance function, usually the Mahalanobis distance (which has the advantage of scale invariance). The most common class among the k nearest of its "neighbours" is chosen as the appropriate label. As such, the training procedure is trivial, although the classification stage, to which the computation is deferred, can be slow when using large datasets; memory usage can also be a problem. The choice of k is also difficult to ascertain in a principled way, although in general small values where $k > 1$ make the algorithm more robust, while the choice of $k = 1$ is generally suitable only for very small datasets (< 100 samples).

Neural networks are also often employed in classification. While Rosenblatt's [138] classic single-layer perceptron shares much in common with a basic linear regression approach, the neural networks generally used for classification are perceptrons with multiple-layers, which are capable of learning complex non-linear relationships between data, due to the usually non-linear activation functions working within each neuron.

A problem with neural networks in general is that the choice of network topology has a significant impact on the viability of the network after it has been trained. The hidden layer, if too small, may render the network incapable of suitable approximation; if too large the net also becomes prone to over-fitting. Irrelevant features can be a danger for neural networks, as they may make the back propagation learning process significantly slower.

A closely related approach to neural networks is Support Vector Machines (SVMs), another non-parametric, supervised learning technique. Like discriminant analysis, SVMs aim to locate some form of decision plane that divides clusters of data, but SVMs are non-linear and also aim to maximise the margin between the plane and the groups' outliers, so wherever possible, the plane that provides the largest gap between groups is chosen. This provably reduces the upper bound on the expected generalisation error.

A significant advantage of SVMs is that the number of features in the training data does not adversely affect the performance of the completed SVM classifier, since the number of support vectors selected by the algorithm to define the hyper plane, is usually small. On the other hand, SVMs are generally unable to classify more than two classes at once, which calls for some form of binary decomposition, and certain parameter choices can leave the SVM prone to over-fitting its model to training data.

Generally speaking, SVMs and neural networks tend to perform better when dealing with high dimensional spaces and continuous features. Categorical data is best dealt with by the decision tree algorithms. As more sophisticated algorithms, however, both require more parameters to be set than do other techniques.

Apart from the decision trees, non-parametric methods generally manifest themselves as "black boxes" since it is difficult to understand how or why they reach a particular decision. This is a particular disadvantage in areas of business, such as credit scoring, or medicine, where people are uneasy leaving important decisions to an algorithm they don't understand, but is less of a concern when producing vision systems[19].

Many of the algorithms, both parametric and non-parametric, are in some way dependent upon the initial random choices assigned to certain values, whether it be parametric coefficients or neuronal weightings. An advantage of the population approach to learning, employed by most evolutionary learning algorithms, is that the solutions cover a broader surface of the search space instead of a single point, so initial random choices are less of an issue. Of course, the population paradigm places an additional computational overhead so different runs may yet provide different performances if not run for long enough.

Although the results in this chapter show that Genetic Programming is competitive with state-of-the-art techniques on given problems, perhaps more so than have done pre-

[19]We are accustomed to not understanding how the human vision system works!

vious GP researchers, the results also correspond with a long line of comparative studies which all state that there exists no "magic bullet" solution to problems involving classification. Furthermore, although this discussion draws attention to the pros and cons of each technique in an analytical sense, these issues do not necessarily manifest themselves in real world scenarios: A study [139] found that the naive Bayes' classifier, despite its requirement for feature independence, could sometimes be superior to more sophisticated algorithms.

Nonetheless the most appropriate course of action is to select the classification technique which appears most appropriate for the task. Given the kind of vision tasks tackled within this thesis, there are certain techniques that can be ruled out. Linear discriminant analysis, for instance, generally requires that the classes have equal numbers of members, which is rarely the case in vision and certainly not in generic vision. k-NN places a similar requirement and like neural networks is regarded as sensitive to noise and irrelevant features, both of which may be present in images.

Evolutionary techniques are not often mentioned in discussions of classification. Unlike the above approaches, genetic techniques are particularly bound by any given representation: GAs can be used to determine neural network weights, or weights for discriminant functions in place of the least squares technique. GP can readily develop decision trees, and the concept of a margin can be built into a GP fitness function to offer advantages similar to SVMs. We've seen that the author's ICS classification technique makes use of different classification approaches dependent upon the task to be solved, which may be advantageous. In the following chapters we shall see how well equipped is Genetic Programming as a generic creator of vision systems.

Chapter 6

Generic Feature Detection using GP

Having established a classification strategy by Genetic Programming (GP) which is competitive with other techniques and considered how it might be converted into a "black-box" learning procedure, we can now turn our attention to using it for the automated development of vision systems. This chapter will describe some of the imaging techniques used and will discuss the construction of training datasets for image problems.

Here we begin to focus on some key problems in computer vision, and examine how GP classifiers may be employed to produce vision applications. That is, alas, too high level just now – we must first explore exactly how one might create a suitable datasets for a given vision task. Among the datasets used for validation in Chapter 5 was the Iris dataset [136], in which the task is to distinguish three species of iris given four measurements of the length and width of the flower's petal and sepals[1], and the Wisconsin Diagnostic Breast Cancer set [140], which consists of a number of measurements relating to cell nuclei which may be predictors of the presence of cancer. The nature of the Iris and WDBC measurements seem rather obvious, given their specific domains, but exactly which measurements should we take from images in order to develop a generic vision solver? This chapter starts by discussing the kind of image measurements that may be useful. Later it shall be demonstrated how they can be used by GP to approach a selection of low-level vision tasks.

Exactly what makes a *useful* image feature is a significant point of discussion in itself. Some algorithms, such as SIFT [60] and SURF [61], were developed to find generic points of interest within an image, which seems quite promising. A range of other detector families are concerned with the detection of many kinds of features, such as corners, edges, blobs, lines and geometric shapes. Although the kind of features detected by SIFT

[1]A sepal is part of a flower, usually green, lying under the more colourful petals.

are quite robust, making them suitable for image matching and 3D reconstruction work, one has to consider the extent to which different *objects* can be described using such features. It may be these points of interest are in fact *too* generic to work well in our experiments; indeed a substantial database is needed when SURF features are used for classification[2].

While the goal of this thesis is to discover a generic *methodology* for solving problems, this does not necessarily mean that it should be reliant on generic features. Instead the intention is that our GP-powered learning algorithm should find highly *specific* solutions to particular problems by tailoring or tuning features. Although such solutions may have limited scope for re-use, the automatic means by which they are developed makes this concern somewhat irrelevant. The idea of this work is to develop a system capable of producing “disposable” vision systems!

6.1 The Approach

As we’ll see in the next chapter, the author’s vision framework may develop solutions with up to four separate, evolved stages of processing. Each stage may be defined broadly as either feature detection or object classification. The first, feature detection, is the topic of this chapter. The author’s approach is introduced following a brief discussion of two different image processing techniques.

6.1.1 Segmentation

Images are usually represented as arrays of pixel data. Such arrays are typically large, yet each individual pixel bears little information about the overall content of the scene. To understand the image it is helpful to look at groups of pixels rather than in isolation. Segmentation (discussed on page 48) is a method for grouping the pixels into larger “segments”, each of which may represent information about the image content more succinctly. Although there are various means for going about segmentation, the author’s approach is to have GP develop some measure of similarity which categorises a given pixel into a particular class. A decision is made for every pixel in the image. Segments are then constructed from homogenous regions of similarly classified pixels.

The nature of the similarity is problem-specific, as is the number of different classes, but the concept of “apparent material” is a useful rule-of-thumb. Given that objects are of-

[2]According to David Lowe, the inventor of SIFT, a “typical” 500×500 image will give rise to approximately 2000 stable SIFT features.

ten composed of components made from a single material, identifying those pixels which have similar characteristics to known pixels of a given class will go some way toward identifying the position and contours of objects and sub-objects within a scene. The computer vision developer may then choose to discard, investigate or compare different regions.

Segmentation has a large range of applications. It may be used to identify bands of mineralogical or geological features based on multi-spectral image data, to identify human skin, for background subtraction, are part of optical character recognition, or in medical image processing, to name a few examples.

Although segmentation can be quite flexible, it does have certain drawbacks. For instance it works best when objects have clearly defined boundaries, so segmenting more amorphous objects with soft edges may not yield robust results. It will typically distinguish objects based on material, so each object must have reasonably consistent appearance. Finally, it isn't always invariant to changes in scale.

6.1.2 Window Detection

One common method which addresses some of these issues is window detection, which is usually employed to detect particular patterns or shapes within a given rectangular area or window, which "slides" over the image. At each position, a number of features are calculated for different areas and shapes within the window, and a decision is made whether an object exists at the location or not.

Rather than basing all features on or around a central pixel, the window detector may take its decision based on relationships between different groups of pixels within the window. If the size and location of features within the window are defined by percentage measurements of the window's own dimensions, then the window can be increased or decreased in size in order to achieve a degree of scale invariance.

Window detection has two particular issues. The first is that for an $x \times x$ window, there will be an $x/2$ thickness border around the image that cannot be identified. Second is that the appropriate window features for a given task often have to be discovered themselves.

6.1.3 Feature Detection Approach

The author's approach is to combine the two techniques into a single generic feature detection system. Since the segmentation process outlined above (by which each pixel is classified in turn) is essentially an image filter, it is broadly similar to the window detection method. If all the features are made to exist within and relative to a variable scale window, then the approach should be able to detect objects in a scene of variable size.

Although the detector is capable of higher-level detection when demanded, the technique will typically be used to label the whole image, for the purpose of region extraction, yielding an output similar to that of a segmenter.

The author's approach thus works as a general feature detector, intended to identify certain regions by surrounding context, or texture, or colour (or a combination of each). To render it capable of doing so, it is necessary to choose image functions that can robustly quantify these characteristics. In the next section we will discuss the kind of features that may be useful, before going on to describe how they can be used to generate a suitable dataset for training the evolved detector.

6.2 Evaluating Image Features

Having discussed some of the ways in which Genetic Programming may be used to classify an object, given an n dimensional feature vector, we now turn to the construction of the vector itself, specifically: what image metrics might be useful, and how can we ensure they are as robust as possible?

Since there is a potentially endless supply of metrics that can be harvested from an image, it is worthwhile considering criteria by which each may be judged. The author has decided upon four different considerations that should be taken into account.

1. Robustness Robustness may be defined as a feature's invariance to image transformations and other external factors prone to change, such as illumination or noise. The general idea is that the feature should return a consistent value if referring to the same object. Common geometric transformations include rotations and scaling. Certain problems, such as the recognition of vehicles from satellite images, will require rotation invariance around one axis, and many problems will encounter objects at different scales (face detection, for instance). Processes such as noise reduction (covered on page 38) may help make the output of certain features more consistent.

2. Localisation A second important consideration is the ability of the feature to localise particular objects accurately. It is no use being able to determine the presence of an object in a scene without being able to accurately pinpoint its location! In general single-pixel features will be able to do this, although they consider less of the overall context. Features that make use of several pixels, such as convolution masks or other summed areas, may be more useful than individual pixel features, but may localise less well, unless they are highly specific. It is important to have a balance of features using various different area sizes.

3. Efficiency As the feature detector may be run on every pixel of the image, and by an interpreted GP system at that, it is essential that all features be as computationally efficient as possible. There are various means of increasing the efficiency of features. Convolution masks can be optimised or approximated, and in recent years the usage of an *integral image*, or summed-lookup table [141] was popularised by the work of Viola and Jones [3]. An integral image, as its name suggests, is a summed table produced from an original image in which a point at (x, y) consists of the sum of pixels' intensities in the rectangular region from the origin $(0, 0)$ to that point. Using such a table, it is possible to calculate sums or arbitrarily-sized image regions using just four array accesses, two addition and two subtraction operations. Certain statistics, such as standard deviation, may also be calculated more efficiently[3].

4. Usefulness In a general sense, each feature should aid the decision making process by providing "useful" information. To contrive an example to illustrate the point, the number of credit cards that a person has may help predict how much debt they are in, but the last three digits on the back of each card is unlikely to be a successful predictor: both features come from a similar source, but one is significantly more helpful than the other. In imaging the same may also be the case.

Naturally enough, these requirements tend to conflict with each other: the more localised a feature is, the more sensitive it is to small fluctuations, which may impact its reliability. The need for efficiency limits the complexity of features that can be used. Needless to say, it is difficult to find a feature that can meet all the requirements, but the idea is to provide a library of features that is sufficiently diverse as to allow each requirement to be met in different ways. The task of the GP system is to choose the appropriate features required to develop a suitable feature detector.

[3]The efficient calculation of population variance is an interesting topic in its own right. Since the standard deviation is calculated as the sum of differences for a series of values $x_1, ..., x_n$ from their mean μ, it can be calculated in two passes, one to calculate μ, and one to calculate the standard deviation. However, this requires storing all values of x which is inefficient. A naive one-pass solution calculates the population variances as $\sigma = (sumofthesquares)/n - (squareofthesums)/n^2$, but this is numerically unstable in the cases where n is large and the difference between the numerator values is small. The author uses a one-pass algorithm by Knuth [142], which iteratively calculates the mean and is numerically stable.

The final requirement, concerning the assessment of usefulness, will be discussed later in this chapter. However, before we can discuss the *selection* of features, we must first give some consideration the features themselves.

6.3 Image Features

In order to render the system capable of solving a variety of different problem types, image features from three different families were explored:

- Colour Information, about the intensity and colour of a particular pixel.
- Texture Information, regarding the pattern surrounding the pixel in question.
- Spatial Information, regarding the location of the pixel within the image.

These are discussed below.

6.3.1 Colour Features

In this section, the author shall discuss the kind of *colour* features that can be made available to the feature detector. However, since "colour" is more of a perception than a physical property, it is worthwhile to consider the pre-processing operations that may make colour information more robust in general.

Robustness to Illumination

While light sensors (including our eyes) only perceive a beam of light in terms of its wavelength – a scalar property – the light is actually the product of at least two sources: the irradiance of the light(s), and the reflectance of the surface being viewed. The human vision system seems able to recover the surface's reflectance, what we describe as "colour", with relative accuracy, regardless of the light source, a phenomenon known as colour constancy. While it is sometimes difficult simply to calibrate camera white-balance in order to reproduce colours faithfully, the development of software that can *perceive* colours accurately is even more tricky.

Developing colour constancy algorithms is just one part of the huge task of developing a sensor that can compete with the masterpiece of engineering buried within the human vision system. The discipline of colour science is rather beyond the scope of this thesis, but a couple of ideas will be discussed briefly here. We shall make use of one of them later in this chapter.

One algorithm, white-patch retinex [143], assumes that the true colour of the light can be identified by looking at white patches in the scene (which are assumed not to affect the irradiance). White patches are relatively easily identified, as they are the usually the brightest. When the light's colour is known, then the colour of the rest of the image can be divided by that value to recover the "true" colour of the scene's materials. The white-patch algorithm is used by many digital cameras for calibration, although it is dependent the presence of something white, and that the illumination is reasonably consistent throughout the scene.

Another approach, grey-world assumption [144], is based on the expectation that the *average* colour of images tends to be grey. Therefore if one finds the actual average colour of an image, each pixel can be adjusted such that the average colour becomes mid grey. Although slightly more computationally expensive than the white patch algorithm (the image needs to be read twice, once to calculate the average colour, second to adjust the values) the technique can produce reasonably good results.

Figure 6.1 shows the author's implementation of a basic grey-world retinex algorithm on two images that have been "lit" with different lights. While it is difficult to know what the "true" appearance of the image, minus the lights is, the algorithm appears to work reasonably well, here at least producing the kind of consistent perception necessary to make decisions more robust, although the saturation and contrast in the lower images is reduced because of division operation. While the author does not claim to have implemented an ideal solution to the problem, its inclusion here is recognition of a useful processing step for a system designed for creating generic vision systems which may have to work in non illumination-controlled environments.

Exposure Compensation Machine vision systems may also produce inconsistent results in environments where the range of intensities is too large for the camera's exposure system to accommodate. Although humans are able to perceive both very bright and relatively dark objects within the same scene with relative ease, cameras tend to correctly expose one area while underexposing or overexposing others, manifested in some images by either completely white-out skies ("blown out highlights"), or dark silhouettes in the foreground ("blocked up shadows"). The human retina is composed of both extremely sensitive intensity sensors (rods), and much less sensitive colour sensors (cones). This allows our vision system to support a much higher dynamic range than current camera

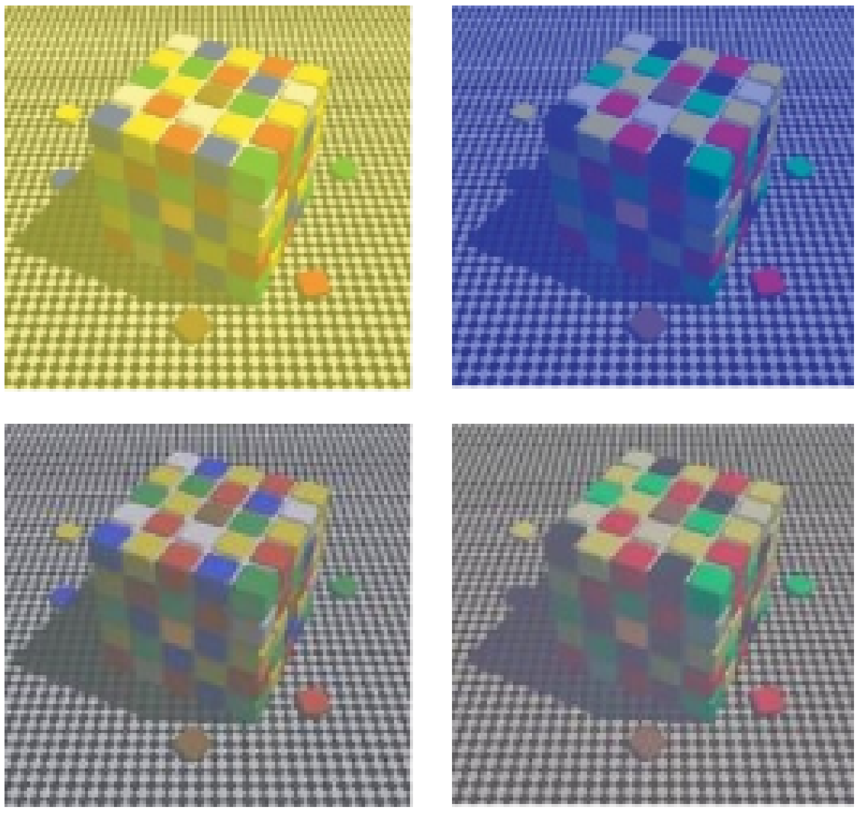


Figure 6.1: The two images at the top have different tints, to indicate different lighting conditions. The images on the bottom show each images following adjustment by the author's grey-world-assumption implementation. The lower pair of images are more comparable than the top two.

sensors, which typically only apply one level of sensitivity to the whole CCD at once[4]; exposure in cameras is typically governed by a single light meter. Furthermore, camera light meters assess exposure by subject brightness rather than incident illumination.

Until camera sensor technology improves in this regard, computer vision software in environments with uncontrolled lighting conditions will remain a limitation affecting the robustness of machine vision applications. But again, there are algorithms which can improve the camera's exposure in software, one of which is described here. As it is generally the case that illumination affects large areas while a surface's texture is finer, an estimate of the average exposure of a pixel can be gleaned from looking at a weighted average of the pixels around it. If the weighted average is treated as the level of illumination, then the pixel's intensity can be updated accordingly such that all pixels in the scene have more equal illumination. This forms part the ideas proposed by Land in his Retinex Theory [145][5]. The author has implemented a simple example of this algorithm, the results of which are shown in Figure 6.2. While the image cannot be perfectly restored, the algorithm is capable of reducing some of the blocked up shadows; the contrast boost may be enough to permit image processing algorithms such as edge detectors to find features where they would otherwise fail.

This completes this author's brief foray into the world of colour constancy and exposure control. Although there are many other more complex approaches, these techniques can, at least, aid the perception of an image by making some parts visible, and providing a degree of the consistency which is vital for the robustness of learned vision approaches. Later in this chapter we shall consider some vision tasks which may benefit from colour constancy pre-processing, and investigate whether these algorithms produce more robust solutions.

Appropriate Colour Spaces

Having discussed how images might be rendered more consistent before processing, we turn to the numeric description of colour itself. The traditional way to express a colour in computing environments is dictated by the construction of display hardware which

[4]The Sigma SD14 Digital Camera is intended to be better in this regard since it uses an unconventional sensor using a separate element for each of the three colour components instead of a single one; however, camera manufacturers generally do not publish the dynamic range of their sensors.

[5]Edwin Land, founder of Polaroid and prolific researcher, developed his retinex theory after conducting experiments which showed people can perceive colours in an image even when that colour isn't used at all by the illuminant. He stated the perceived colour is not absolute, but depends on the surroundings.

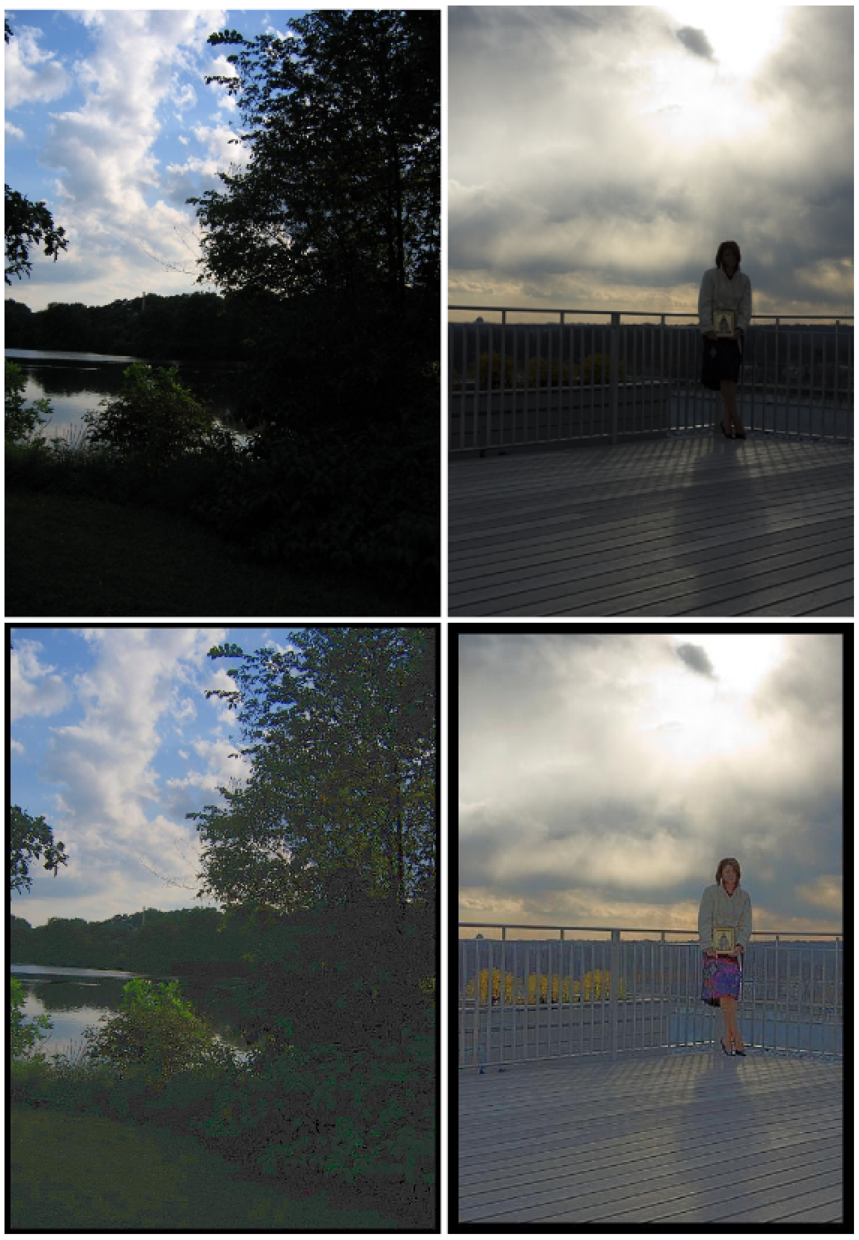


Figure 6.2: The two images at the top are both poorly exposed. However, it is possible to recover the image to an exposure more akin to the perception of a human, using a software exposure compensation algorithm (see text). Some colour information is lost, however, due to the 8-bit nature of this image.

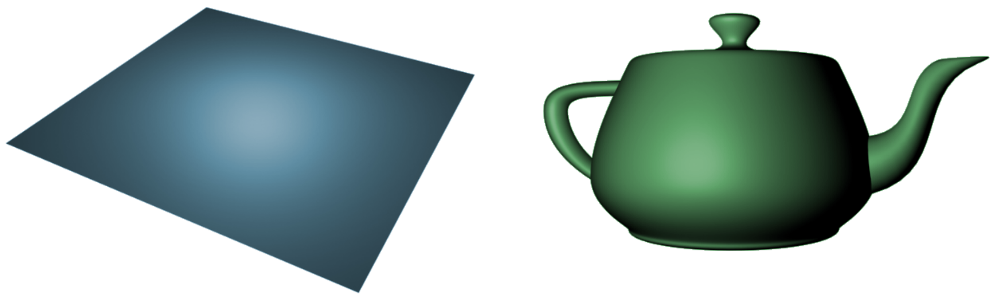


Figure 6.3: Planar surfaces will generally receive variable illumination from point lighting, the geometry of non planar objects will cause different parts of the object to be illuminated differently.

present a colour in terms of its red, green, and blue components. The intensity, or brightness, of a given colour can be calculated by performing a weighted sum of each component. The weights are chosen that match most closely our own perception[6], accepted values are generally in the region of:

$$I = 0.3R + 0.59G + 0.11B$$

The benefit of using RGB values is that intensity can be computed rapidly using the above formula; however, the additive nature of RGB space means that none of the three components is independent of the illuminant's intensity, which leads to issues of robustness with regard to illumination and object geometry. Illumination originating from a point light source will shine with different intensities on most 2D or 3D surfaces since all points are not the same distance from the light source (see Figure 6.3). Directed lighting in general will illuminate a three dimensional shape in different ways according to its geometry. Thus while an object may be composed entirely from the same material, the RGB components are neither illumination or geometry invariant; algorithms using them may be prone to over-segmentation. Consequently, raw RGB components are not used by the author's system.

However, RGB may be used as the basis for producing different colour spaces. Many

[6]The cones (colour sensors) in the human retina have different sensitivities to different wavelengths. The most sensitive cones are those that respond to green light, while the least sensitive are those that respond to blue light. This ratio is taken into account in the formula above. Since yellow is regarded as the combination of red and green, the two most sensitive wavelengths, this makes it the most suitable colour for high-visibility clothing.

computer vision researchers choose to use colour spaces which separate the illumination component of a colour wholly or partly into its own independent channel. Hue/Saturation/Intensity (HSI) is one space which partly fulfils this criterion, defining a colour according to its shade, its saturation ("strength of colour") and overall illumination intensity. HSI colour components make potentially more succinct information about the pixel's colour available to the learning system making each feature inherently more useful. Although the conversion from RGB to HSI is reasonably straightforward there are other RGB transformations that also benefit from more illumination invariance.

Since Hue and Saturation are less affected by changes in illumination intensity or shadow, the features are generally considered more robust, although the Hue value is unstable along the entire achromatic axis, and unsurprisingly is best avoided in grey-scale images. The author's system can automatically detect the use of grey-scale images; in these cases colour-based features such as Hue are disabled.

Although RGB components in their raw forms pose disadvantages, they can be readily used to generate different colour features. For example, they are often normalised as follows:

$$R = \frac{R}{R+G+B}$$

$$G = \frac{G}{R+G+B}$$

$$B = \frac{B}{R+G+B}$$

Since the RGB space is additive, the total intensity can be estimated by summing each component. Dividing each component by the intensity estimate gives a value that should be less affected by the intensity of the illumination. Although the colour of the light itself will still have an effect, this may be mitigated by colour constancy pre-processing.

Other easily computable models include the $c1c2c3$ model by Gevers and Smeulders [146]:

$$c1(R,G,B) = tan^{-1}\left(\frac{R}{max(G,B)}\right)$$

$$c2(R,G,B) = tan^{-1}\left(\frac{G}{max(R,B)}\right)$$

$$c3(R,G,B) = tan^{-1}\left(\frac{B}{max(R,G)}\right)$$

This model is sometimes used for shadow identification owing to its illumination invariance due to the geometry of the object, although this only applies to illumination by white light. The $l1l2l3$ model [146] is also used, as it is invariant to specular highlights:

$$M = (R-G)^2 + (R-B)^2 + (G-B)^2$$

$$l1 = \frac{(R-G)^2}{M}$$

$$l2 = \frac{(R-B)^2}{M}$$

$$l3 = \frac{(G-B)^2}{M}$$

In common with any feature derived using a division, the normalised RGB operators are unstable near the black vertex of the RGB cube. The best that can be done is to ensure that no divide-by-zero errors occur. HSI values, normalised RGB values, $c1c2c3$, and $l1l2l3$ are made available to the GP segmenter as *channels*, using which any image statistic may be computed. Of course, statistics typically require the input from a number of different points; features which combine data from many pixels are discussed in the following section.

6.3.2 Texture Features

So far we have discussed single-pixel features, which can only be meaningfully described by their intensity or colour, and are limited to a few reasonably straightforward design choices. This section will deal with features that combine the inputs from an *area* of pixels near to a central point of interest.

A benefit of multiple-pixel features is their potential to describe the spatial context around a point of interest, otherwise known as pattern or texture. Although colour or intensity information is often useful in segmentation, in low contrast or grey-scale images the texture of different objects may be the only suitable discriminant. One example is the distinction between human skin and wood, which are quite often to be found in close proximity and yet may appear very similar in terms of colour alone – especially through low quality cameras.

One approach is to provide the feature detector with the intensity and other colour information regarding every pixel in a region surrounding the central point of interest, and leave the GP system to invent texture discriminants of its own. Given arithmetic, statistical and logical operators, it is perfectly possible that the GP system could evolve a form of convolution mask or indeed a more complex descriptor. Since no data are lost, it may be possible to develop a measure that accurately represents the texture. We saw in Chapter 2 that Song and Ciesielski [86] compared this approach to evolution using Haralick statistics (covered shortly) as image features. The problem is that a relatively large number of pixels is required in order to produce a discernible pattern, and typically each pixel is represented by one terminal in the GP tree, so the evolved programs would have to be relatively large in turn. Given sufficient time, a learning system may discover useful ways of combining pixel data to develop texture features, but it may longer than the genetic programmer is prepared to wait! This kind of approach, attractive in some senses, is thus unlikely to be a practical solution. Indeed, Song and Ciesielski found that the individuals using Haralick features yielded more accurate results.

Therefore, the requirement of texture features is that they should reduce the dimensionality of the data by processing a number of pixels together in order to arrive at a single statistic that concisely represents a notion of texture. Since several features may be used for each pixel, their processing should be as fast as possible to ensure that the learning process can proceed and that the eventual program can operate in practical circumstances.

One family of texture features are those which produce averages of colours by using a $n \times n$ mask to convolve pixels in some way. Common examples include Gaussian, Laplacian and Sobel Masks. Gaussian masks may yield an intensity estimate that is more robust to noise, while Laplacian and Sobel masks are useful in detecting edges and their direction. Different masks with different weight matrices can easily be applied, permitting the segmenter to discover edges, or identify certain basic patterns. Although convolution masks are typically used to emphasise certain parts of an image, they don't give much indication of texture, as their primary effect is to blur or discard spatial data, or match specific patterns.

Other descriptors of texture may be imagined quite readily. Indeed we saw in Chapter 2 that GP researchers have made use of a number of *ad hoc* features. Possible descriptors include the difference between neighbours' intensity values, which may give an indication of the contrast of the texture, the arrangement of neighbours may that reveal the directionality of patterns within the texture. The rate of intensity change across a series of points could reveal the roughness or smoothness of the texture. Given a set of pixels,

Contrast	Contrast is measured by the sum of GLCM probabilities, weighted according to their distance from the diagonal point of the GLCM (the further the distance from the diagonal, the higher the contrast).
Angular Second Moment	Calculated from the sum of squared probabilities, high values provides a measure of the "orderliness" of the texture since high probabilities (definite patterns) will be emphasised, and low probabilities will be emphasised less.
Entropy	Gives an indication of the "disorderliness" of the image by summing the natural log of each probability, which gives highest weight to low probabilities.
GLCM Mean	Measures the average probability of whatever relationship the GLCM is measuring. Horizontal and vertical GLCMs may produce different means.
GLCM Variance	Measures the Standard Deviation of GLCM probabilities and is similar to the entropy measure. A completely uniform image will have a GLCM variance of zero.

Table 6.1: A Description of Several Haralick Statistics

one obvious approach is to calculate the variance in value of a particular channel, which gives an indication of the smoothness. Other straightforward statistics such as the range may be used to get an impression of the contrast in that area of the image. The number of edge pixels in a particular area may provide some indication of the "busy-ness" of the image, and the directions of each may provide useful, relatively scale invariant information about the nature of the texture itself. Although simple, these calculations may be computed more easily than other techniques and may still provide useful discriminants.

One of the most well-known texture analysis methods computes texture descriptors from Grey Level Co-Occurence Matrices (GLCM). A GLCM records probabilities between different intensity levels given a particular spatial relationship between two pixels, for instance the relationship between one pixel and the pixel directly below or above it. Each GLCM can be used to calculate a variety of different texture metrics, known as Haralick Statistics, some of which are summarised in Table 6.1.

While the GLCM is relatively easy to understand and calculate, its usefulness is somewhat dependent upon the quantisation level of the image. For 8 bit images, producing 256 separate grey levels, the matrix will require a 256×256 array, which is resource hungry, especially if the process must be repeated for every point in the image. With an array of this size the matrix will inevitably be extremely sparse, unless the area surrounding the pixel is expanded significantly, although this in turn will produce a "border" effect around the edges of the image and reduce the feature's ability to localise. The practical reaction is to quantise the image intensity values into larger ranges, although this has the drawback of de-sensitising the matrix to more subtle changes.

Haar-like Features

Haar-like features are used for object detection/recognition rather than analysis of texture *per se*. Although some objects are deformable and may be better identified by texture, shape is often a useful descriptor. A differential operator, Haar-like features *compare* two or more adjacent rectangular areas within a portion of an image. For each area the sum of pixel intensities is calculated; the feature returns the difference between different areas. The feature can be in any location or scale relative to a central decision point. Haar-like features were popularised by the work of Viola and Jones [3], who used them for face detection. Viola and Jones invented additional features, composed of three and four sub-rectangles (see Figure 6.4 for some examples). The principal advantage of Haar-like features is that they can be computed readily using an integral image (see page 131). The integral image can be manipulated to return sums for rectangles tilted at 45°, which may be useful in some applications.

The disadvantage of Haar-like features is that they cannot be used "out-of-the-box", as it were. The parameters that identify useful Haar features in terms of their position and scale need to be determined first, and this process may also be processor intensive. Typically a parameter optimiser such as a Genetic Algorithm is used to identify the parameters that define useful Haar features for a given task.

In order to render Haar features more consistent, the author's implementation divides the differential result by the area of the feature, such that the result of any feature will occupy the same numeric range (0-255).

Dissociated Dipoles

"Dissociated dipoles" are similar in nature to Haar-like features, except where the comparison made between adjacent rectangles is inherently local in nature, dissociated dipoles

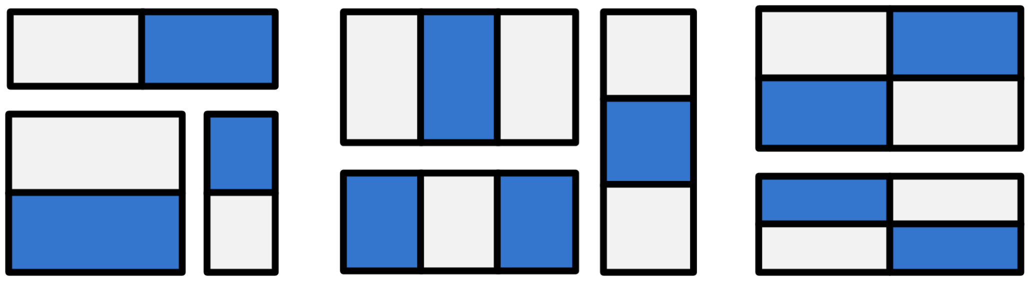


Figure 6.4: Some examples of 2/3/4 rectangle Haar-like features. The darker rectangles correspond to those regions that are subtracted.

make comparisons between different areas of the image, which may be far apart. In some ways this is a more flexible structure than Haar-like features. The original dissociated dipole was an operator that measured the differential between two gaussian convolution masks of variable scales and locations.

Again, dissociated dipoles require their parameters to be chosen according to the vision task, in a process of feature extraction. The author spent some considerable time looking into this, including using Genetic Algorithms and other search procedures. The solution eventually devised by the author was to develop a new type of GP terminal node which includes its own parameters. The node can initially choose random parameters for itself (within sensible limits), and subsequently generates different features using a similar procedure to produce a population of features. Given a score, the node can go about a hill-climbing technique to improve its discriminative abilities. This technique is integrated into the author's feature selection and extraction interface which is covered in the following chapter.

6.3.3 Location-Based Features

A more straightforward family of features gives an indication of the pixel's location within the image. These features include the distance of the pixel from the centre of the image, and the angle thereof. Certain problems in vision may pay more attention to those objects closer to the center of the image than on the outer edges. Many problems, however, will not require such features; the feature selection technique employed by the author's toolkit (see Section 6.4) will remove these features if not needed.

6.3.4 Other Features

By now we have covered a variety of different image operators, each of which are made available to the author's GP system for the purposes of evolving a feature detector. Other features include the author's *DistanceFeature* which computes the Euclidean distance between the entire current input feature vector and a known feature vector from the training set. This allows GP to create solutions slightly similar to k-nearest-neighbour, which is often surprisingly effective at finding solutions. Unlike k-NN, however, the distance of only one vector may be checked at a time, so the feature instead provides another means of non-linear transformation.

6.3.5 Invariance

Integral images make it possible to compute Haar-like features efficiently. Indeed it is this kind of feature that makes it possible for modern digital cameras to recognise faces in real time, despite their limited processing capabilities. Although they can be used to approximate circular regions, rectangles are generally not rotation invariant, which limits their applicability for certain problems.

If we confine ourselves to the recognition of 2D objects from a top-down perspective, then rotation invariance can be achieved by using circular features. The disadvantage is that they cannot be computed so quickly, although they can be approximated. One may use either completely circular features, rings, or circular perimeters. The latter were used by Roberts & Howard [20] for recognising vehicles and ships from satellite images, with some success. The author prefers perimeter based-features, because they involve fewer pixels and are therefore more efficient to compute. Various calculations can be made from each perimeter, including the mean and standard deviation of pixel intensity. Perimeter features can also calculate the number of edges they encounter: an "edge" is detected if the difference between two adjacent pixels' intensity is larger than half the image's standard deviation.

For some problems it is not necessary for the feature detector to be rotation invariant. In the following chapter we shall see examples of features which lend themselves to rotation invariance more elegantly.

The scale invariance of certain features may be measured by scaling them (and the image) up and down to see whether they return a consistent result. An experiment was run for this purpose, using a number of image points and a selection of 6 straightforward features, which calculate the means and standard deviations from horizontal and vertical

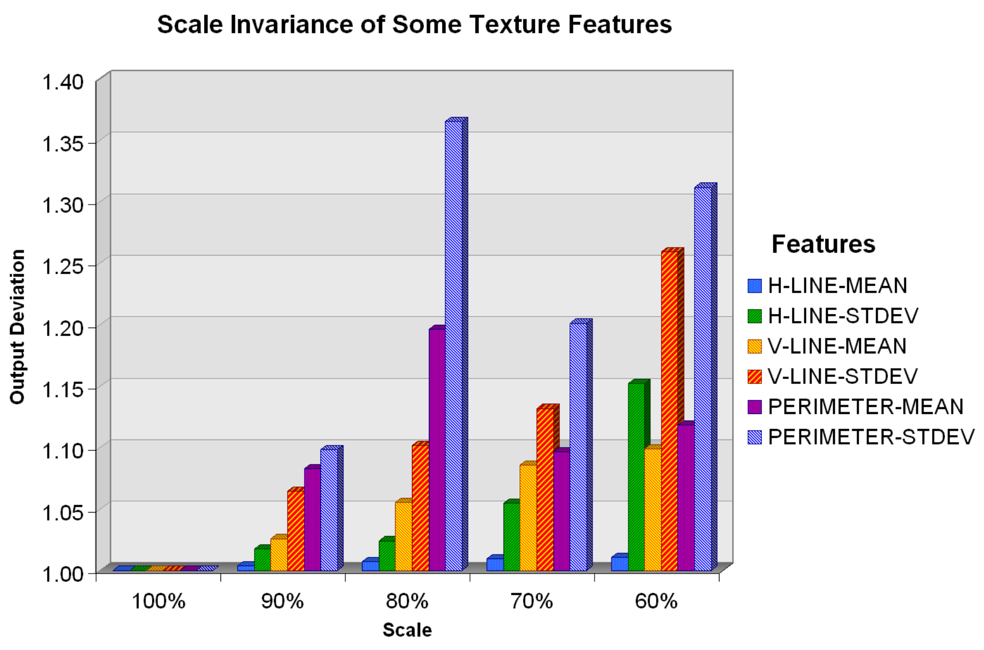


Figure 6.5: Measuring the invariance of certain features to changes in the image scale. As the image approaches half its original size, some features' output deviates by over 30%. The perimeter, which increases in size while maintaining a single pixel thickness is naturally most prone to deviations of scale. Other features such as the mean, unsuprisingly, are very robust to changes in scale.

lines, and from circular perimeters. The average deviation of each feature from its value at 100% image size is plotted in Figure 6.5. As one can see, many of the *ad hoc* features do not produce invariant results at different scales. As one might expect, the mean is largely unaffected by changes in scale, so mean-derived descriptors, such as the Haar-like features should produce similar robustness to changes in scale.

A common approach to scale invariance is through the use of a framework referred to as "scale space", which considers an image at various different scales, following convolution by a gaussian mask of different sizes. This method can reveal those points which are apparent at all scales in the image, so it is commonly used in feature point detection techniques, such as SIFT [60]. Once these features have been detected, they can be used for a variety of tasks, including object detection.

However, it is less straightforward to adapt existing features into scale invariant ones, since most are inherently local in nature. One reaction to this is to define the scale of features themselves according to some window. The window can then be applied to images at different scales in order to find objects or features of different sizes. The author's segmentation features may be defined in terms of position and location within an arbitrary window, expressed in percentages.

To summarise, the author's feature detector has access to approximately 60 different image features, including image intensity, normalised colour operators, Haralick statistics, Haar-like features, dissociated dipoles, rectangular, convolution operators and linear and circular area statistics.

6.4 Feature Selection

Given the large number of potential features with which one could describe an image, a few of which have been described above, it is worthwhile to identify those features that may be redundant or irrelevant. Motivations for eliminating "useless" features include avoiding the so-called "curse of dimensionality", the effect in which the search space grows exponentially as extra features are included. Learning only using relevant features may also improve the learned system's ability to generalise, as its outcome is dependent only on true discriminants. Finally, in the spirit of *Ockham's Razor*, feature selection can help make the classifier more efficient, both in terms of final execution speed (fewer features need to be calculated for the classifier to make its decision), and the rate at which the classifier can be learned in the first place.

Two schools of thought dominate feature selection philosophy filtering and subset selection, discussed in the following paragraphs.

Filtering The first approach is concerned with evaluating features individually. Features are usually scored against certain criteria, with the top n being selected for inclusion within the feature vector. The criterion may involve measuring the variable's linear correlation with the ground truth, or indeed their individual predictive power when used by a classifier such as Linear Discriminant Analysis.

Other metrics aim to assess "usefulness" in more concrete terms, for instance by calculating the *information gain* of a feature, which is defined as the difference between the entropy of the training data $H(Y)$, and the average conditional entropy $H(Y|X)$ given a particular feature X . The larger the information gain, the more useful is the feature X.

A criticism is that these techniques do not generally consider the correlation *between* different variables, which may lead to a feature set characterised by a high level of redundancy. Some techniques, such as [147], aim to maximise relevance while minimising redundancy by developing a more complex scoring system that takes both into account.

Subset Selection A distinct approach is to consider the set of features as a whole, without trying to assess the usefulness of each variable individually. The rationale is that simply because a feature doesn't perform well against some metric individually doesn't mean that it doesn't have any merit.

A straightforward technique in this regard was reviewed in a popular paper by Kohavi and John [148], concerning the "wrapping" of learning machines as black boxes. The black boxes were used as evaluators for different variable subsets. Since our perception of what should make a useful variable will not necessarily translate into a positive outcome for the solution, this method dispenses with any arbitrary scoring and focuses instead on a more direct predictor of success.

This kind of approach attracts its own criticisms for being rather brute force in nature. Putting to one side the idea that an exhaustive search could lead to over-fitting on the training data, for all but the simplest problems the computational expenditure required makes exhaustive search impractical.

It appears that the rough search strategies implemented by greedy hill-climbing algorithms seem to do well instead, both in terms of alleviating over-fitting and preventing the computation from becoming intractable. *Forward selection*, by which features are added to the dataset one at a time, or *backward elimination*, by which features are eliminated one at a time, appear to be effective greedy algorithms. Nonetheless, Genetic Programming is already slow enough, so the author chose not to use wrapper methods for feature selection, favouring the individual filtering algorithms which can be executed in a matter of seconds.

In any case, one of the advantages of Genetic Programming is that it permits a reasonably large library of features to be used; feature selection is a natrual consequence of the selective process. The nature of genetic learning systems is to breed the useful solutions more often than the use*less* useful ones, so a viable form of feature selection is already included. In any case, genetic programs are under no obligation to use all the features available, and the usual fitness criterion favouring smaller individuals may help to ensure solutions are as general as possible[7].

[7]In fact, without proper checks, GP programs will sometimes use none at all, and simply return the most popular class.

6.4.1 Experiments

Nonetheless, with an expanding set of image features, it is worthwhile investigating which are most useful for a particular task. Conversely, one can consider feature selection as a means for removing irrelevent features that may affect the ability of a classifier to generalise or operate efficiently. At this point it is necessary to move away from the public datasets and their fixed feature sets, and start using images directly to generate feature vectors. Over the course of his research, the author has developed a number of image datasets for different purposes; four of which are described below.

Flag Colours Dataset. Colour. 8 Classes The first dataset is concerned with the recognition of different colours, taken from images of 16 different European flags. Images of each flag were printed and then cut out to produce different cards. Images of each card were then captured through a webcam[8]. The goal is to reliably identify the colours apparent in each flag design.

Given the relatively poor reputation of webcams in terms of photographic excellence this set provides a useful experiment to identify the robustness of vision algorithms created by the author's system, using everyday equipment in habitual environments. The samples in the dataset are categorised into 7 classes, corresponding to 7 colours[9] and a background class.

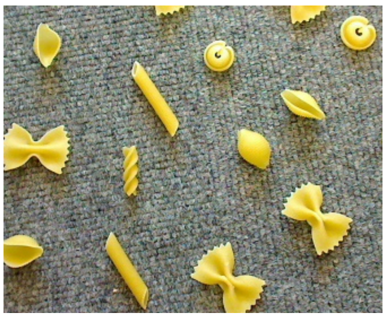
Pipeline Dataset. Texture. 4 Classes The author's pipeline dataset is derived from top-down images collected from an unmanned aerial vehicle (UAV) at a height of several hundred feet. The task is to identify automatically different regions within each image: ground, vegetation, water and buildings/roads/pipeline.

The pipeline dataset is a more useful assessor of segmentation by texture, as each class cannot readily be distinguished by colour (the saturation of the images is quite low).

[8]A Logitech Quickcam Pro 5000, a reasonably good quality camera, costing about £50 in 2007.
[9]'Red', 'Yellow', 'Green', 'Light' 'Blue', 'Dark Blue', 'White' and 'Black'.

Leaves Dataset. Background Subtraction. 2 Classes The third dataset is concerned with locating different types of leaf, photographed against a variety of cluttered backgrounds. Images come from the Caltech leaves dataset [149]. This is a binary dataset with two classes: leaf and not-leaf.

Pasta Dataset. Background Subtraction. 2 Classes This dataset is representative of some kind of industrial inspection task. Like the leaves dataset, the primary task here is background subtraction, permitting the localisation of objects.



Three different feature selection techniques were applied to select features for each dataset. Each was an attribute filtering technique, as opposed to a subset-selection method. These were:

Linear Discriminant Analysis The feature's capability as an independent discriminant was measured using Linear Discriminant Analysis (see page 123). The score was measured according to the number of samples the LDA classifier could correctly classify using the feature, using the fitness function $error_1$ (see page 19).

Program Classification Map (PCM) The author's implementation of a DRS2 Program Classification Map (see page 69) was used to convert the feature into a classifier. The score was measured according to the number of samples the PCM classifier could correctly classify using the feature

Information Gain (IG) The score of each attribute was measured by its Information Gain (see page 146) with respect to the known classes, which measures the difference in entropy between the training set and the output of the classifier on the training set.

Each feature selection technique was used to reduce the size of the feature set from 60 down to 20 features. Each attribute was individually scored, with the top 20 chosen to form a new feature set, from which a dataset was produced. Both the reduced and original datasets (using all the features) was then used as training data for the author's ICS classifier, which developed solutions in the usual manner for each. Each experiment was repeated 50 times; the results are shown in Table 6.2. In the Flags dataset there were no statistically significant differences in performance on test data. In the Pipelines task, the reduced datasets did produce, on average, individuals that were significantly better, although the absolute difference was not huge.

The numerical basis for an interesting observation is shown in Table 6.3 which shows the percentage difference between the training error and test error for each dataset. For both tasks, the difference is significantly higher when using the entire set than when using filtered subsets. This indicates that classifiers developed using the reduced feature sets are able to generalise better, perhaps because those irrelevant features that may affect generalisation have been removed successfully.

In conclusion, the results on these tasks appear to show that the feature sets for feature detection problems can be quite sharply reduced without causing any significant decline in performance; indeed the performance of pipeline classifiers improved for all filtering algorithms. With the desire to add more and more image operators to render a system capable of solving a wide variety of problems comes the need to ensure that the feature sets are maintained at a manageable size, and it would appear that this kind of feature selection technique is a suitable means for doing so. The observant reader will note that subset selection techniques are not covered here: the author's preference is to use filtering techniques that can assess the relevent features in seconds. A lengthy process of genetic learning, preceded by an even lengthier feature selection operation would be likely to test the patience of the end user!

Having covered three different feature filters, a reasonable question is which is the most useful? Based on the limited results provided here, a definite choice would be premature. Nonetheless, the author has a preference for the *PCM* technique on the basis that it is fast, and naturally shares much in common with the GP classifier itself, so would appear to be a good model. As we'll see later in this chapter, there are further uses for this type of feature selection.

Task	Entire Set	Filtered (LDA)	Filtered (PCM)	Filtered (IG)
Flags	93.5 ± 3.5	93.9 ± 2.6	92.7 ± 2.6	93.9 ± 2.9
Pipelines	95.4 ± 0.5	**95.9** ± 0.5	**95.6** ± 0.5	**96.3** ± 0.3
Leaves	97.6 ± 1.0	**98.0** ± 0.6	**97.7** ± 0.9	97.7 ± 0.8

Table 6.2: Comparing the average performance of classifiers, evolved using different feature sets, on test data. Significant differences in **bold**, ± denotes standard deviation.

Task	Entire Set	Filtered (LDA)	Filtered (PCM)	Filtered (IG)
Flags	7.39	2.17	2.32	3.16
Pipelines	14.16	0.61	0.37	5.34
Leaves	17.4	10.26	1.59	7.05

Table 6.3: Comparing the average percentage difference between classifiers' training and testing performance (an indication of their ability to generalise), evolved using different feature sets

6.5 Performance Considerations

Like GP learning, machine vision is often computationally expensive. If the former is used to develop the latter, then it is doubly important to ensure that evolution can proceed as efficiently as possible! We have already touched on various means by which the GP process can be made more efficient, by decreasing population redundancy or employing integral images and faster means of calculating statistics. Here we shall look at some further means by which the performance of GP vision learning can be improved.

6.5.1 Image-Level Caching

Efficiency can be achieved both by ensuring the algorithms themselves are not computationally wasteful, and by minimising the number of re-calculations made, typically by caching results.

Certain sub-trees become common in genetic populations because of the selective pressure imposed on good individuals, leading the crossover operator to repeatedly copy certain sub-trees of popular parents. An approach taken by Roberts [150] to improve the speed of GP on image processing problems was to cache the results of common sub-trees on the training data, saving the need to re-evaluate them continuously. While this technique was shown to be quite successful, it was memory hungry: one value for every pixel on every image for every sub-tree is stored, probably at least one byte in size. Accordingly, a cost-based decision was made to determine which sub-trees should be included in the cache by estimating the time taken to retrieve the cache from disk versus the time taken to make the calculation in the first instance. Roberts reported decreases in evolutionary time were measured between 1.5% and 52.2%, which is quite promising. Experiments were conducted on a variety of different population sizes; the largest improvement was attained when using a population size of 2000, rather larger than is generally used, and thus more prone to benefit from any kind of caching.

The author's own caching approach takes place at the image level, rather than the sub-tree level. Since image processing is the most significant bottleneck in learning vision systems, it makes sense to cache the imaging functions themselves, rather than the sub-trees, which often amount to little more than performing arithmetic operations upon underlying image calculations. SXGP makes use of the author's own imaging library which caches the results of imaging operations so that subsequent processes can proceed more quickly. Basic pixel metrics, such as the mean grey-value of an image may be called several times *per pixel*, for instance by convolution masks. These are cached. The caches are implemented using one-dimensional arrays which can be accessed efficiently. The results of other, more complex, operations may also be cached.

Figure 6.6 shows the advantage and disadvantage of this type of caching, here implemented in two ways: "half" caching, in which only the most essential operations are cached[10], and "full" caching, where a variety of other results from more computationally expensive operations are cached. The figure shows that the time taken to process n images is significantly lower for the cached methods, with "full" caching requiring the least

[10]Grey-scale, Red, Green, Blue, Hue, Saturation, Lightness.

amount of time overall. Caching permits the operations can be run on the images nearly three times faster than otherwise.

While caching permits evolved programs to work efficiently during deployment, it is dependent on large amounts of memory. In Java, the language in which SXGP is coded, every integer or float requires 4 bytes of memory, so storing the result of any image operation on a 640×480 image will require approximately a third of a megabyte, assuming Java is memory efficient[11]. While this is not too much of a concern for vision applications which typically look at one image at a time, it may be overwhelming for the Genetic Programming training process, in which all the images (and associated caches) are typically held in memory at the same time.

Accordingly, the lower graph in Figure 6.6 shows the memory requirements of each caching technique – the "full" caching technique requires up to 9Mb per image, which could charitably be described as a limitation on the number of images that could be used in training[12].

The problem is exacerbated by the bluntness of the caching mechanism, which is intended for operations on the whole image. The training data, by contrast, may only comprise a small number of chosen pixels from each image. Nonetheless, the cache is calculated for the whole image, which is wasteful.

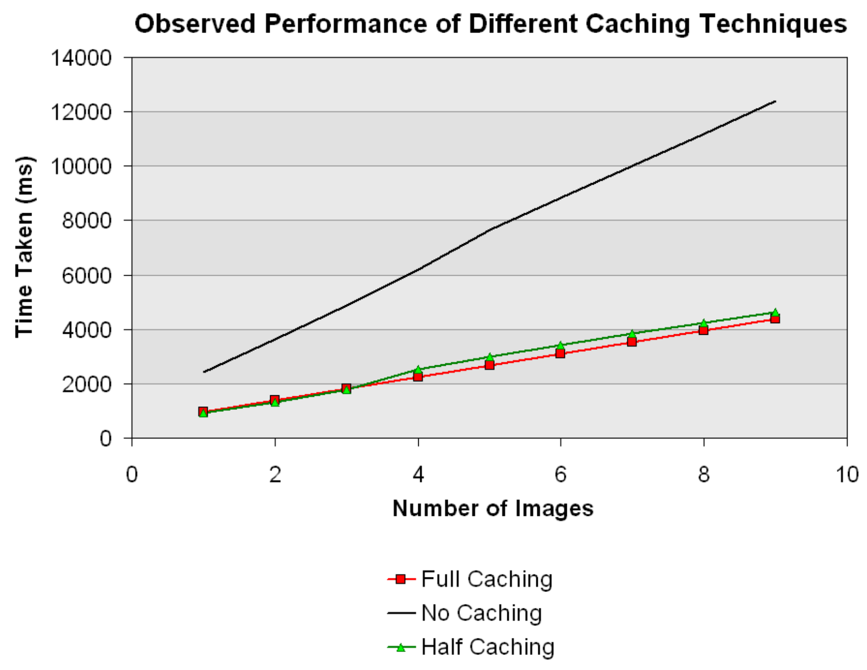
The author's solution is: another level of caching! The idea is to pre-compute the outputs of the feature set for every pixel in the segmentation training set, and store them in a separate dataset structure, the same used to represent the public datasets used throughout Chapter 5. Not only does this permit solutions to both public datasets and imaging problems to be evolved by exactly the same system, the dataset structure requires significantly less memory per image, as only the pixels part of the training set are included.

To implement this approach the terminal nodes in SXGP trees are able to execute in either of two modes: the first using the pre-computed values from the dataset, the second calculating the feature directly from the image as before. This permits the GP system to produce "live previews" of the segmenter on test images, while training on a precomputed dataset that doesn't use images at all.

It must be acknowledged, however, that this strategy is not limitlessly scalable. In the author's current implementation, the dataset resides entirely within memory; some huge datasets would still be too large to accommodate. Experiments involving the selective

[11]It isn't.

[12]The images in this case were 320×211 in size. Uncharitable descriptions of this level of memory usage are left to the reader's imagination.



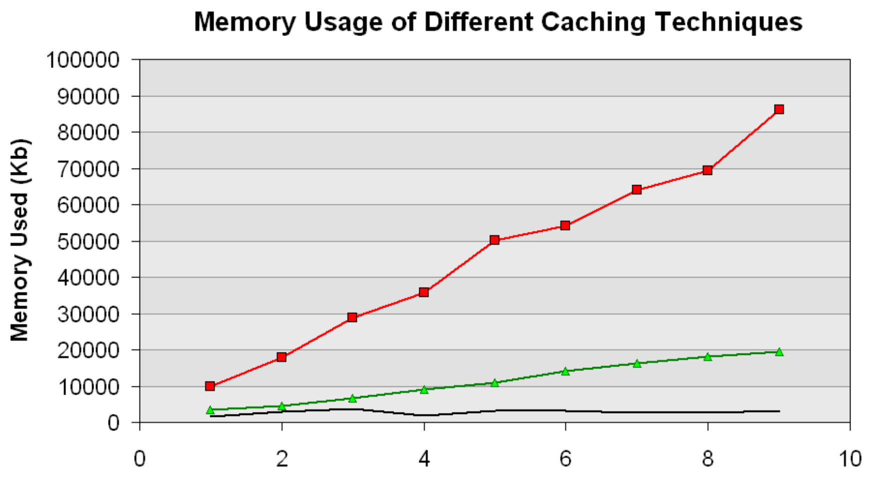


Figure 6.6: Comparison between different levels of image caching in terms of performance and memory usage.

culling of training data, to ensure speedy and feasible evolution, await the reader in the following chapter.

6.5.2 Fitness Caching

The author, whose enthusiasm for caching continued unabated, went on to invent a different form of caching, similar in spirit to that of Roberts [150] but with a lower memory overhead. The approach, termed "fitness caching" ensures that some needless evaluations do not take place.

As was discussed in Chapter 5, a feature of the ramped-half-and-half tree builder (see page 15) is that it generates a number of duplicates. This is because the builder generates a fixed number of trees at every tree depth; since there are fewer small trees than large trees, a number of them will not be unique. A plot of the proportion of duplicates in a randomly generated population of different sizes is shown in Figure 6.7. We have also seen that any selection technique will bring about a reduction in variability. Despite the vastness of the search space, the paradox is that there is always some degree level of duplication within a GP population.

The author's image caching mechanism ensures individuals can be processed quickly, but there is a further saving to be made. Given that it isn't computationally practical for the tree builder to ensure that every tree is unique, it makes sense to attempt caching on a per-individual level, rather than a per-sub-tree level. This requires much less memory, and means the duplicated individuals do not have to be evaluated at all. Instead of caching the output of the tree on each data point, which is memory intensive and still requires the entire training to be iterated through, the author's technique simply caches the final *fitness value* of the individual, on the basis that two identical individuals must be allocated identical fitness.

Each individual's structure is given a unique code, by taking the hash code of its LISP expression[13]. When an individual is evaluated, a record of the fitness for its code is recorded. Subsequent evaluations start by checking the record first; a "full" evaluation only proceeds if fitness hasn't already been calculated for that individual.

The results of an experiment comparing performances with and without fitness caching are summarised in Table 6.4. The results show that fitness caching is able to achieve up to

[13]The bracketed syntax of LISP is well-suited for expressing tree structures. It is acknowledged that hashing functions are not guaranteed to produce unique codes. The author conducted an experiment in which 10,000,000 individuals were hashed. Individuals with identical hash codes were checked to see if their trees were the same. No instances were found in which the hash code was not unique.

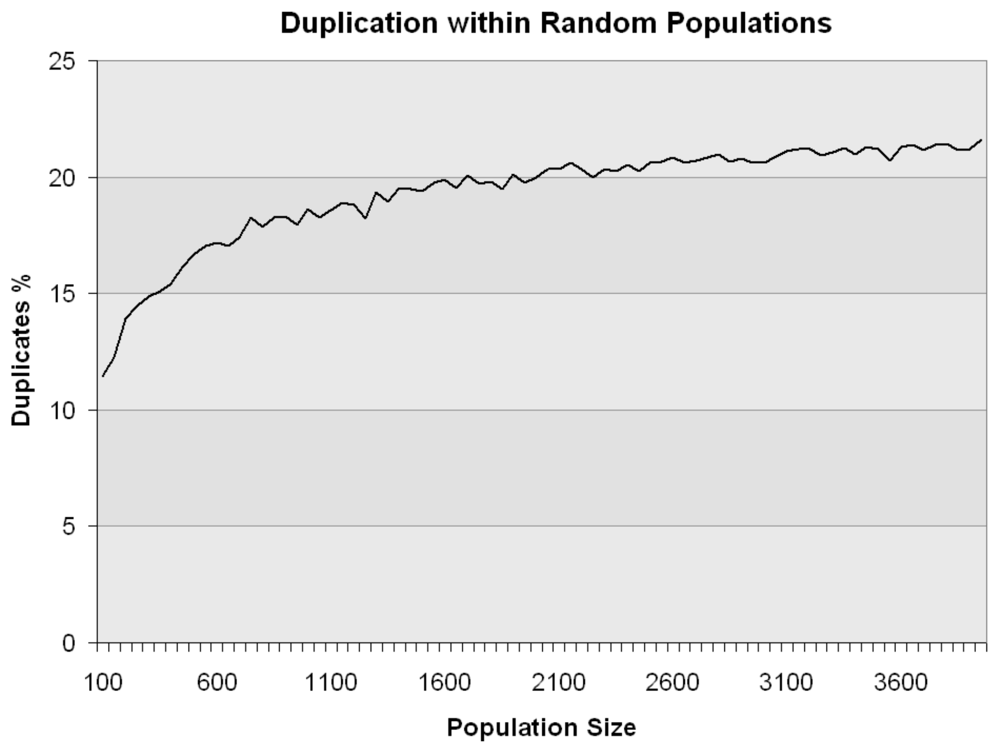


Figure 6.7: The level of duplication in random populations approaches over 20% when generated by the standard ramped half-and-half population builder in the author's GP system.

Data Set	Normal	Fitness Caching	Improvement
	seconds	*seconds*	*%*
BUPA	35.35	32.52	8.7
Heart	28.63	26.66	7.4
Glass	62.46	61.85	1.0
Iris	22.74	21.83	4.2
Pima	51.82	49.31	5.1
Thyroid	102.47	94.27	8.7
WDBC	25.07	21.92	14.4

Table 6.4: Comparison of average runtimes of a 250 member popupation over 25 generations with and without fitness caching. Fitness caching can produce a performance increase of up to 14% on classification datasets.

an additional 14.4% reduction in processing time (6.2% on average). Although this improvement is modest, it requires significantly less memory than other caching techniques and can therefore be implemented *in addition* to any other caching mechanisms.

6.5.3 Deployment Procedure

A further performance enhancement is initiated following the evolutionary process. In common with most Genetic Programming systems, the author's toolkit represents programs as trees. Since Java is a compiled programming language, the Java process cannot execute programs written during runtime, so the tree must be executed by an interpreter which traverses the tree according to the logic of the nodes. Once the Genetic Programming process is completed and a suitable program discovered, it is desirable to dispense with the interpreter, both to remove its computational overhead and to disentangle the program completely from dependence on any Genetic Programming libraries.

Following any tree optimisation operations enacted upon the tree, the author's toolkit begins the process of deployment by converting the optimised tree to Java source code. Each node implements a *toJava()* function, which translates its functionality into Java source code, and in general each node will produce a single programming statement. A source interpreter runs on the tree and combines the statements together, before adding the headers and footers to make a complete Java program. Any functions that are used more than once within the function are cached inline. The Java compiler is then instantiated to compile the source into Java byte code, which may then be executed by the Java runtime. Finally the compiled class file is loaded back into the Java virtual machine as an instantiated class that can be run on any vision problem.

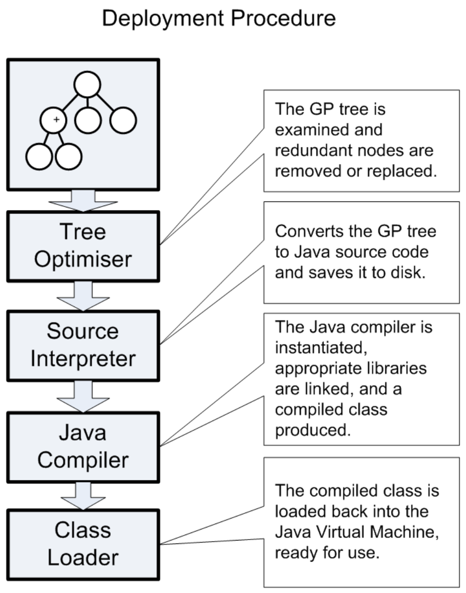

Deployment Procedure
Tree Optimiser
Source Interpreter
Java Compiler
Class Loader
The GP tree is examined and redundant nodes are removed or replaced.
Converts the GP tree to Java source code and saves it to disk.
The Java compiler is instantiated, appropriate libraries are linked, and a compiled class produced.
The compiled class is loaded back into the Java Virtual Machine, ready for use.

Figure 6.8: The deployment procedure for converting a GP tree into a compiled Java class, which typically runs about 20% faster.

Although this process is too involved to be run during the GP run, it yields “pure” Java programs which run faster than interpreted ones and are reliant only upon the author’s imaging libraries. The Java process doesn’t need to be restarted, so the program can be used immediately. Experiments (summarised in Table 6.5) show that, on average, the compiled programs execute between 20–30% faster than the equivalent interpreted versions.

6.6 Experimenting with Preprocessing

Earlier in this chapter several straightforward algorithms intended to achieve consistent colour constancy and better exposure control were described. There were a couple of subjective demonstrations of these algorithms in Figures 6.1 and 6.2, but it seems worthwhile to devise a general, objective means of assessing the usefulness of preprocessing algorithms. In this section, we shall present the results from a limited set of experiments that demonstrate some of the effects of the colour constancy algorithms.

Data Set	GP Tree	Byte Code	Improvement
	ms	*ms*	%
Flags	273	218	20.0
Pipelines	175	127	27.2
Leaves	320	218	31.8

Table 6.5: Comparison between the average performance of interpreted GP trees and compiled Java programs, when running on vision datasets. The compiled version yields a performance boost of between 20–31.8%

We have already seen several of the author's own image datasets, including the straightforward Pasta dataset and the Flags dataset (see page 148). The Pasta dataset equates to an industrial inspection task, and has relatively consistent illumination. The Flags dataset, by contrast, is primarily a colour recognition task, so each image was captured at different exposures and white balances to ensure robustness.

In this section, we shall investigate whether the colour constancy algorithms can affect the quality of classifiers evolved by Genetic Programming. Two datasets were generated for each task, each using the same set of training data points. In the first experiment the images in their original form were used; in the second experiment the images were preprocessed first using the grey-world colour constancy algorithm[14]. All other parameters were identical. GP runs were then performed on each dataset. The results are shown in Table 6.6 and graphically in Figure 6.9, which shows typical GP runs by generation.

Data Set	Unprocessed	Preprocessed
Pasta	0.001	0.001
Flags	0.173	0.015
Pipelines	0.094	0.145
Leaves	0.030	0.060

Table 6.6: Measuring the effect of the author's colour constancy implementations on four segmentation datasets. Shown here is the average error for each dataset after 25 runs apiece.

[14]The white patch colour constancy algorithm was not used – requiring the continual presence of white objects is something of a restriction for a generic vision system builder!

Perhaps unsurprisingly, the colour constancy technique has little effect on the Pasta dataset, which features very little variance in illumination, and in any case already achieves a respectably low average error. On the Flags dataset, however, the difference is marked: the error is reduced from a rather high level down to a much lower level (the accuracy was 98.5%), which is a substantial improvement. Inspection of the images before and after preprocessing reveals why – much of the inconsistency in the training data has been removed by the grey-world algorithm. Although the pre-processed images do not really represent our own perception of the images, they may render a series of differently exposed images to a much more consistent intermediate.

However, for the Leaves and Pipelines dataset, the preprocessing operation makes the classifier significantly worse. This is perhaps because the types of images in these datasets do not fit the assumptions underlying the grey-world assumption.

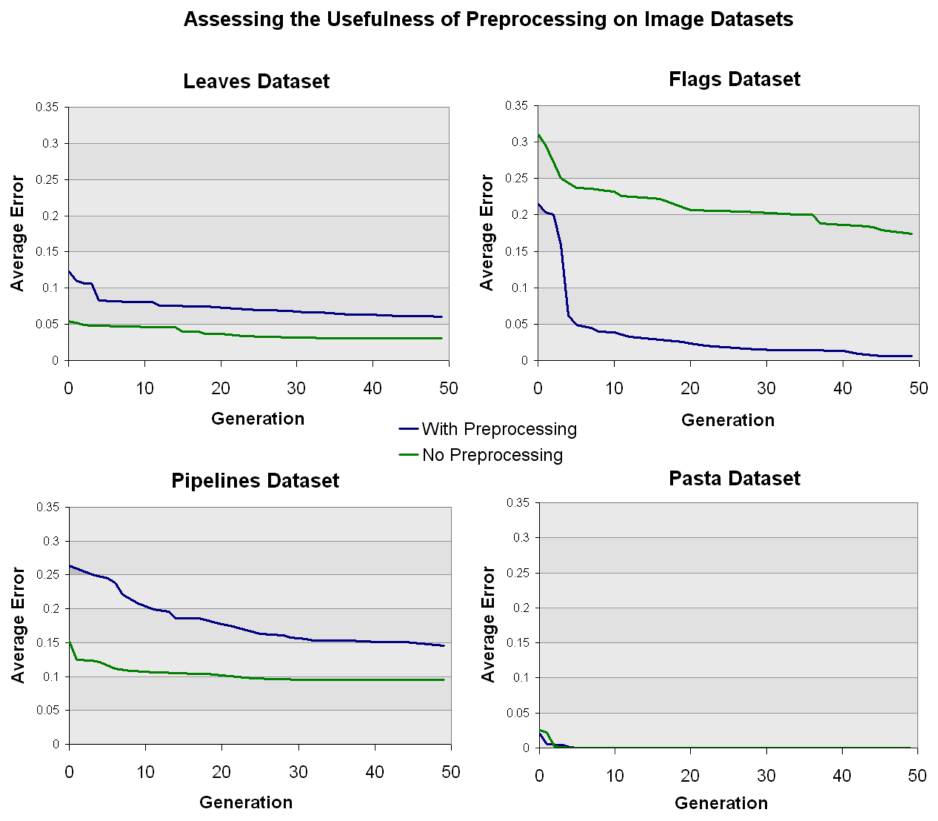


Figure 6.9: The outcomes of the evolutionary process, with and without grey-world colour constancy pre-processing. For the Flag dataset the reduction in error is significant. In other datasets the pre-processing either has a neutral or damaging effect.

We have seen that colour constancy algorithms can make a substantial difference to some datasets, sometimes beneficial and sometimes detrimental. Given the significant improvement on the Flags dataset, it is preferable to involve the colour constancy algorithm where suitable, but a decision must be taken whether to run it or not. Running a colour constancy algorithm that doesn't yield any improvement only adds to the computational burden of the vision system, and may even decrease the accuracy. Since this is a generic system, it would be nice to decide whether certain preprocessing operations are useful *prior* to running the GP process and preferably in an automatic manner. Fortunately, it appears the colour constancy algorithms also have an effect on the author's feature selection techniques – the average feature "score" was improved by between 8–20% for the preprocessed Flags dataset, but it remained unchanged for the Pasta dataset. Similarly it was worsened on each occasion for the Leaf and Pipeline datasets. There would, therefore, appear to be a mechanism for quickly identifying the usefulness of certain preprocessing techniques automatically, using any of the feature selection techniques discussed. All feature filters gave similar results, but Information Gain provided the clearest distinctions in value, shown in Table 6.7.

Data Set	Unprocessed	Preprocessed
Pasta	0.471	0.472
Flags	0.249	1.525
Pipelines	0.865	0.721
Leaves	0.374	0.340

Table 6.7: The average Information Gain by the top 20 features for a given task, with and without colour constancy preprocessing. The difference in error correlates with the observed results, so Information Gain can be used to assess whether preprocessing should be used or not.

6.7 Applications of Segmentation

In this chapter we have covered the design of a generic feature detection system, the means by which image datasets may be produced from suitable image features and techniques for ensuring the evolution and execution of the detector can proceed rapidly.

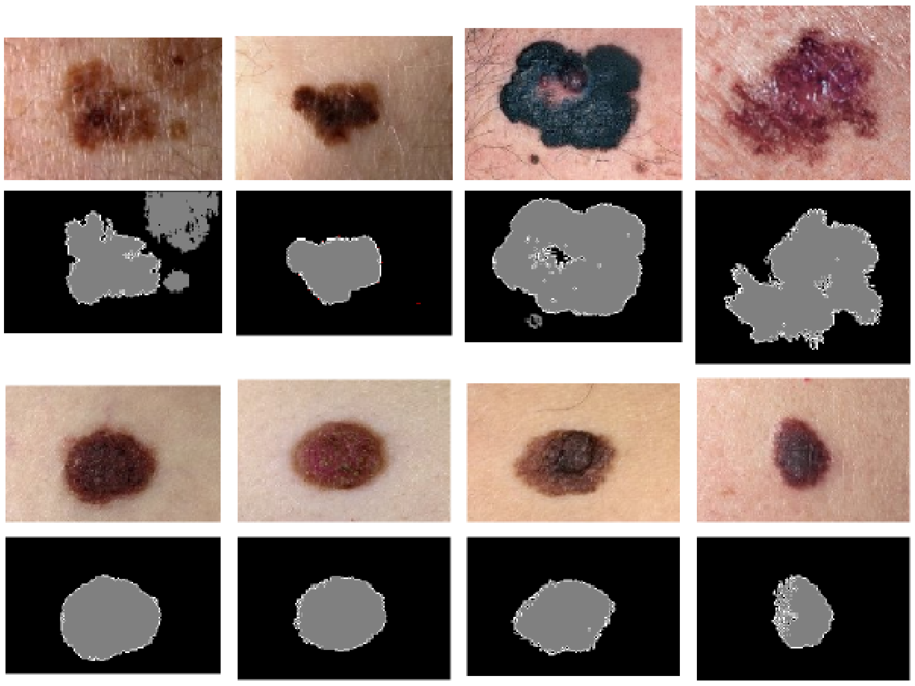
This chapter is concluded with some images showing examples of the applicability of the feature detector. Although numerical results are provided, it is difficult to make empirical comparisons to other work since it is not often possible to access the same datasets. Nonetheless, the idea here is to present how the author's software may be applied to those problems for which a result of publishable quality has already been presented by GP researchers in the past. The objective is to explore the *applicability* of the feature detector to a range of domains; it is left to the reader to assess the quality of the results on a subjective basis. All images below are unseen by the feature detector.

6.7.1 Skin Segmentation

Computer vision has a large number of applications within the domain of medical imaging. The overwhelming majority of these images must be examined by an expert radiologist, oncologist or other specialist. The consensus is that it would be of great use if computers could be used to automate the process in some way. Accordingly this field has received a lot of research, including by researchers using GP. Roberts and Claridge [84] used GP to develop a segmenter for the purposes of skin segmentation, specifically to segment those areas of skin described by medics as lesions.

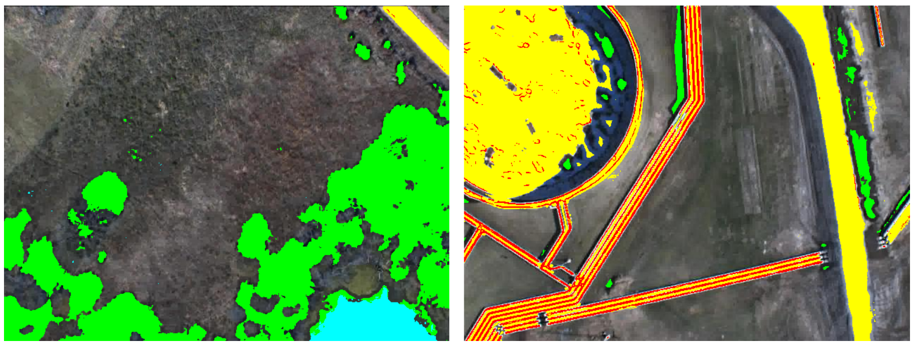
Given a set of 100 images carefully hand-segmented by an expert as training data, they evolved a program that could perform similar segmentations using just eight images as training data. The detector achieved a sensitivity of 88% and a specificity of 96%, and was able to run on images approximately 600×400 in size within 4 seconds. Although the author was unable to acquire the same dataset, it was possible to obtain a similar set of images from a series of online dermatology databases, from which 8 were chosen as representative training samples. The author's generic system evolved a segmenter within 10 minutes that achieved an accuracy of 94.3% and was able to run in under a second on each image.

The results of the feature detector are shown below, in each case identifying the boundary of the lesion. The bottom set of images are benign in nature, while the top images are cancerous legions referred to as malignant melanoma.

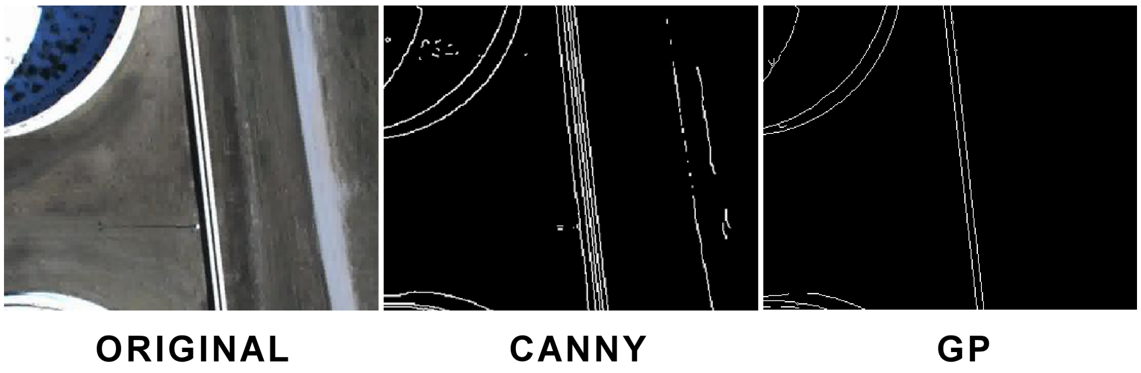


6.7.2 Terrain Segmentation

The author's pipeline dataset has already been described in this chapter. It is composed of several hundred time-lapse frames captured from a camera in an unmanned aerial vehicle. The purpose is to identify gas pipelines in the image which require human inspection on a fortnightly basis in order to meet US safety and environmental standards. The author's feature detector was trained to identify several other features in the scene, including vegetation, roads and water. This information may be helpful in identifying the nature of potential problems with pipelines. The author's system evolved a segmenter over the course of 100 generations, which took 18 minutes per run, on average, achieving an average accuracy of 89.9% on labelled areas of unseen test images. The results on two frames are presented below:



Without a means of assessing this result in a comparative manner, we shall instead consider its *utility* for further stages of processing. The detection of pipelines themselves can be performed by taking the output of the segmenter on one class channel. Following skeletonisation, the output is shown below. Since the detection of pipelines in these images is very much related to edge detection, it is worthwhile comparing the effort of a generic benchmark edge detector – such as that of Canny [64]. A comparison between the two is shown below.

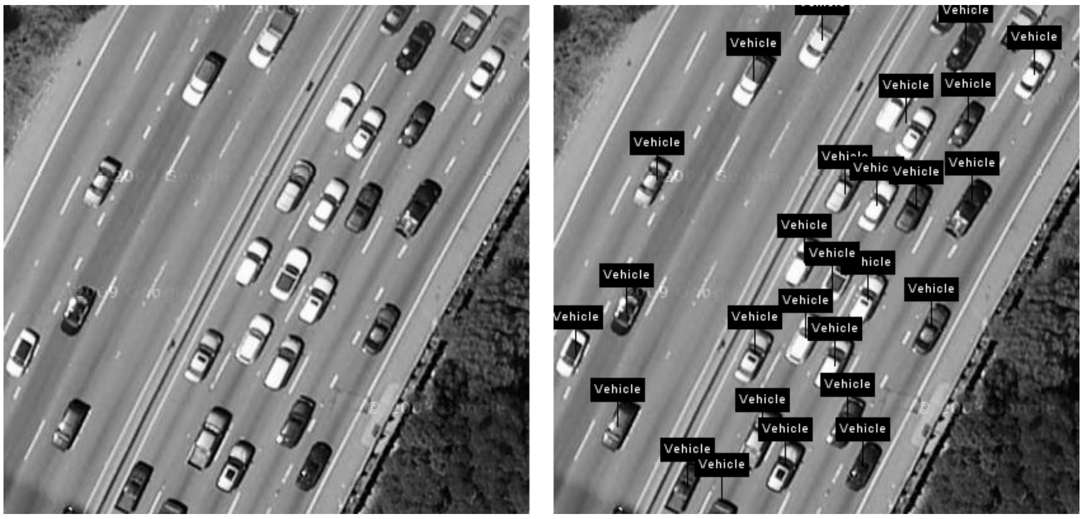


The first observation is that the evolved detector is more specific with regard to the pipelines than is Canny's detector. Furthermore, the detector can be trained to detect the edge along the *centre* of each pipeline, rather than at its edge, which yields only one response per pipeline, which is more helpful. A Hough transform may be used to identify the straight pipelines.

Of course, Canny's detector is generic and the author's feature detector is evolved with specific training data in mind, so the comparison is mean spirited in some ways. Nonetheless, the point is that domain-specific approaches can be more valuable for certain applications than the equivalent generic algorithm – provided that domain-specific approaches are straightforward to develop. The detector in this instance was developed in approximately 5 minutes by GP, following approximately one hour producing training data.

6.7.3 Object Detection

Roberts and Howard [95, 20] developed a rotation invariant object detector using GP for the purpose of recognising various vehicles from infra-red satellite images. So far we've seen examples of segmentation, so it is worth investigating whether the author's approach is applicable to more specific objects which themselves may consist of hundreds of pixels. In a brief experiment, a number of images of San Francisco were downloaded, in which over 600 cars were identified. Each car was labelled using the author's software, and a segmenter evolved. Half the images were reserved for testing. Although the detector yielded multiple responses per vehicle, producing a set of blobs, the author's shape processing software (covered in the following chapter) was used to identify the centre of each, the results on a test image are shown below; the sensitivity was 97.6% and the specificity was 91.8%.

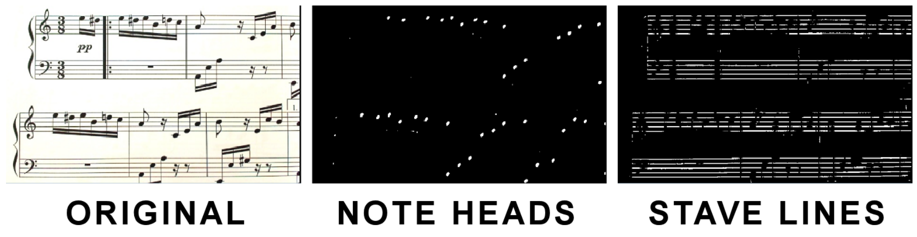


6.7.4 OCR Segmentation

One area in which computer vision has already excelled is Optical Character Recognition (OCR), in which letters and words in images are identified automatically to be converted into ASCII text. The segmentation of black text against white paper is reasonably straightforward – techniques such as adaptive thresholding generally are often sufficiently accurate. In Chapter 4 we also saw two other common approaches to automatic thresholding used in mainstream computer vision. Although the author's software is able to segment letters, it is an instance of wheel-reinvention, so is not covered here. We will, however, see an example of handwritten character recognition on the MNIST dataset in the following chapter.

Other OCR tasks, however, are more challenging from a feature detection perspective. Quintana *et al.* [88] tackled a more interesting problem of extracting musical notation from an image, a problem made more difficult by the overlapping nature of musical notes, staff lines and other notations. Quintana used GP to evolve mathematical morphology operations to extract each. The authors were able to develop a detector for the purpose of classifying note heads to an excellent degree of accuracy, but detecting beams and stave lines was more difficult. It was particularly difficult to evolve a stave line detector with the required level of specificity.

This author has also tried out this experiment on a series of images. To make the problem more challenging, the images were captured from photographs of printed sheet music using a webcam instead of purely electronic versions. The author's detector was applied to the classification of the same three features, and was able to discover solutions to each all using a single multi-class detector, shown below.



6.7.5 Conclusion

In the last chapter we saw how the author's feature-detection approach can be employed for the extraction of different features from images. Some are based on colour, others on texture, and others by their context. The author's generic learning system has been applied to develop specific solutions for several tasks where previously GP researchers had developed custom features specific to each problem. Although the image datasets are not available for direct comparison, the author claims that the solutions developed using the generic system are equivalent.

In this thesis, the detection of features is the beginning rather than the end – in the next chapter we shall look at what to do with the output of the segmenter, and how to construct multi-stage vision *systems*, rather than just vision *components*.

It has already been stated that numeric results on a particular dataset are not always enough to give a good impression of a segmenter's performance. The author would state that a better measure of the performance is its *utility* – are the features that it detects sufficiently reliable as to permit further stages of processing? In attempting to use one set of evolved solutions as inputs to another, one soon appreciates whether the former is helping or hindering the overall process! In the next chapter we shall see just what kind of tasks can be attempted by multiple stage evolved vision architectures on a series of more complex datasets.

Chapter 7

A Framework for Evolving Solutions to Vision Problems

Following a reasonably lengthy description of the author's Genetic Programming toolkit, and the means by which GP may be put to work in evolving classifiers, the preceding chapter introduced a generic approach to feature detection in images which is able to produce similar results to those published by other GP vision researchers; indeed in some cases the author's generic system was able to evolve solutions to problems that other researchers' task-specific approaches had difficulty with. In this chapter the approach is extended to render it more functional and capable of higher-level understanding. The concept is to integrate one or more stages of feature detection into a larger vision system, providing a basis for extracting more complex content from the image by secondary detectors and classifiers which have an altogether different nature. This secondary stage of processing will be described shortly. The key point is that all domain-specific components of the vision system are evolved by GP: the development of vision systems of this type appears to be novel.

The manner in which the different evolved components interact is determined by a straight-forward machine vision architecture, of a type commonly used in conventional computer vision. In this chapter we shall describe the author's approach [7], and demonstrate how it can be used to tackle vision tasks. The framework itself is then extended to render it more applicable to different problems. Toward the end of this chapter, some vision systems generated using this framework shall be presented and assessed.

The final key component of the author's software is a graphical interface that automates and simplifies the process of evolving vision systems. It will be shown how training data may be created to supervise the learning of the vision system components, and how a non-expert user can generate working vision systems in a straightforward manner. In this chapter, the idea is to bring together all the concepts discussed throughout the course of this thesis and show how they can be integrated into one software system, capable of the automatic construction of vision systems.

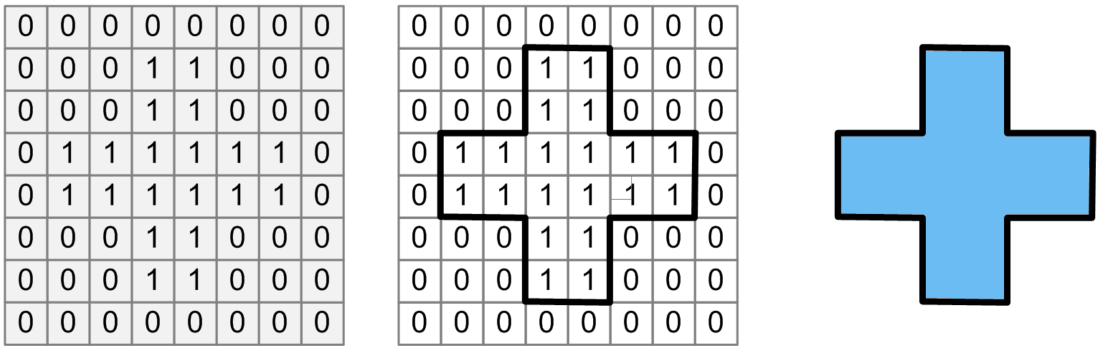
7.1 More Complex Vision

In the last chapter, the evolution of a generic feature detector was described and evaluated. The results showed that the evolved programs were capable of finding a variety of different features in images, based on colour, texture, and surrounding context. The examples presented give an indication of the kind of domains in which such a detector can be put to work. Feature detection alone, however, is not always enough to yield the kind of high-level information that an end user would appreciate – in many cases it merely highlights those items within the image that may warrant further inspection. In order for the vision system to be useful, it should make some attempt to identify those detected features, at which point further automated processing by machine could be initiated. Here we shall move on from the detection of features in general to the recognition and classification of particular *objects*.

7.1.1 Creating Objects

As we have seen, the feature detector makes an individual decision about every pixel in the image, classifying it into two or more user-defined classes. When plotted into an image this gives the *appearance* of segments, but first the pixels must be joined together into sets for the computer to process them further. Homogenous areas of similarly classified pixels are grouped together into rudimentary "objects". Since the feature detection proceeds in a supervised manner, each object can base its initial identifier on the feature classes invented by the user. As well as providing a unique identifier for each, the user can also set whether a class is useful or not. Those objects assigned to a "background" class are immediately discarded, leaving only the objects of interest in the scene; see Figure 7.1.

This set of objects is the primary output of the feature detector. It identifies all objects, permitting them to be located and counted. The output can also be used in subsequent processing for two purposes. The first is to analyse each object more carefully to see whether the segmenter was correct in classifying it as such, the second is to identify the object as a member of a more specific sub-class. An example presented throughout this



0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

Figure 7.1: Object Grouping – Taking the output from the segmenter and turning it into groups, which in turn can be classified.

chapter is pasta recognition – the feature detector cuts out the pasta shapes from the background; the secondary processor identifies the type of each piece.

7.2 Classifying Objects

The secondary processor, which processes objects rather than pixels, is, therefore, another classifier. Given a vector of features that describe a given object robustly, the same classification system can be used on each occasion. Once again we first must turn our attention to the means by which a particular object may be described.

7.2.1 Shape Descriptors

The first family of features comprises a series of descriptors that may be used to describe each segment according to its shape or outline. Although an inherently 2D processing operation, the “silhouettes” of segmented shapes in an image can yield useful information. Although the author developed these descriptors independently[1], they are similar to shape descriptors described in the computer vision literature. They include the following:

Roundness The roundness is calculated by inspecting the edge points of the shape and seeing how closely it approximates a circle, whose edges will all be the same distance from the centre. The function calculates the standard deviation of edge points’ distance from the center of the shape.

[1]One of the perils of an incomplete literature review!

Rectangularity A measure of how rectangular the object is, which is measured by comparing its area to the total area of its bounding box. As a percentage measure, this function is scale invariant. Since the bounding box is perpendicular to the axes, it is not rotation invariant.

Symmetry The symmetry may be calculated by looking at how well the shape is mirrored along a particular axis, in this case both the horizontal and vertical axes.

Roughness The roughness is calculated by looking at the *variance* in radius between different edges along the perimeter of the shape relative to the centre of gravity. The function works by comparing opposite pixels with respect to a given line of symmetry and awarding points if they have the same value. The points are divided into the size of the shape to acquire a scale invariant measure.

Density The outer and inner perimeters of the shape are identified, and the volume of any "holes" in the shape are computed. If there are H holes, each with a size of h_x, in a shape whose mass is s, then the total hole size hT is $\sum_{i=1}^{H} h_i$. The density is $hT/(s + hT)$. The density measure returns a percentage which is scale invariant.

Number of Corners Although it is rather difficult to measure the number of corners in an image, corners are defined as those points where the differential of the tangental angle of an edge is above a certain threshold. This is similar in concept to the Wang and Brady corner detector.

Number of Edges Following a process of skeletonisation on the image, the number of skeleton edges can be counted. This is a useful descriptor of the structure of the shape and is scale and rotation invariant.

Number of Joints Following the same process, the number of joints, or points at which edges converge can also be counted. This is also scale and rotation invariant.

Maximum and Average Depth During the process of skeletonisation, the depth of each pixel from the nearest edge is measured, permitting additional statistics to be calculated regarding the maximum and average depth of pixels in the shape. This can give a rotation invariant indication of the overall thin-ness of the shape.

Balance The "balance" of the shape is measured as where its centre of gravity falls, compared to its centre point. This can be used as another measure of the shape's symmetry.

Size The size descriptor, or number of pixels in the shape, is the most straightforward measure. It is generally useful in discarding smaller segments arising from an inaccurate segmenter or a noisy image.

The observant reader will note that some of these features are invariant to certain geometric transformations[2]. Density, for instance, is both scale and rotation invariant. Others, such as *rectangularity* which requires a threshold, is scale invariant but not rotation invariant.

Some problems do not necessarily require full invariance to all transformations – OCR for instance is generally expected to work with the letters facing upside-up, as it were[3]. Other tasks, such as pasta detection, are very much dependent on rotation invariant features since each shape may be observed at different angles. The author's feature selection techniques, described in Chapter 6, can be used in a similar manner to discard those features that are not sufficiently robust for a particular problem. Since all data is placed into a common dataset interface, exactly the same procedure can take place, even though the task is quite different. Of course, a secondary process of feature selection will occur during evolution, as those features that do not yield sufficiently fit individuals will become less prevalent within the population.

7.2.2 Material Features

While shape statistics permit classifiers to distinguish basic objects based on their outline shape, they cannot be used in all situations, and indeed discard a substantial amount of information regarding the appearance of the shape itself. Although the feature detector will have selected the shapes on the basis of their sharing a similar material, that material may be as simple as "not the background". Therefore there may still be considerable variation in the appearance of segments. It is useful, therefore, to provide features relating to the colour and texture of the segments.

These features are quite similar to those used by the segmenter (covered in Chapter 6), but are calculated as averages for the whole object. The features include:

- Average Colour/Intensity Values

[2]It is assumed that illumination invariance will have been accounted for by the feature detector.

[3]Humans too are trained this way – our own recognition of upside down text usually leaves something to be desired! Of course, you can train yourself to read upside down.

- Colour/Intensity Values Standard Deviation
- "Within Shape" Haar-like features
- Intensity Averages for the border of the shape
- Intensity Averages for the centre of the shape

The material and shape descriptor features may be used in conjunction – again the feature selector and the GP system will decide their applicability to particular problems. Following the process of feature selection, each shape is represented by a tuple of shape or material descriptors, and is sent to the classifier.

7.3 A Framework for Evolving Vision Systems

It should be reasonably clear to the reader how these components may integrate readily into a two stage architecture, generating vision systems that can cut out arbitrary parts from the image, and then make higher-level decisions about them. The first phase is required for the detection of objects in the scene, and the second aims to classify each object into a particular class. This system is illustrated in Figure 7.2 with reference to the pasta recognition example. Such an architecture is sufficiently abstract as to cover a variety of different problem domains, and is commonly used in conventional computer vision.

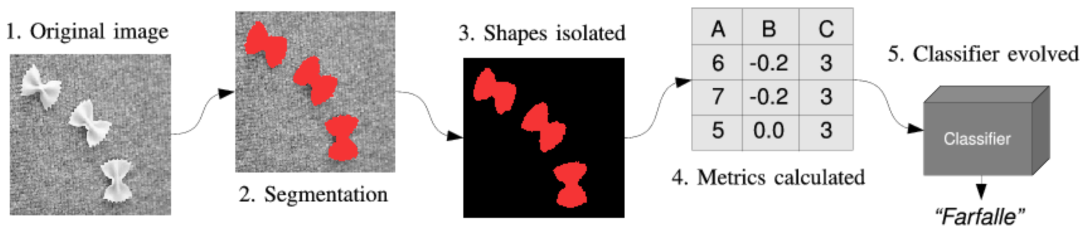


Figure 7.2: The First Vision System Used for Experimental Work

7.3.1 An Extended Architecture

Towards the end of this chapter we shall see some examples of vision systems created using this framework, including a gesture recognition system and pasta recognition system, which are certainly quite different domains. Both of which were created using the framework described above [7]. However, the initial framework implies certain restraints that limit its applicability. Chief among these is the requirement that the object be cut out in its entirety by the segmenter. In most situations this demands that the shape must be relatively homogenous. This is sufficient for objects like pasta, which (should!) be composed of a single material. However, some objects do not suit this approach – such as the flags in the author's Flag data set (introduced on page 148). The segmenter can either cut out flags from the background in their entirety, or segment individual colour regions of the flags. In the first case, the object classifier would make the startling discovery that all flags are rectangles; even with average colour information the overall shape descriptor features may not be concise or robust enough to make accurate distinctions between them. In the latter situation it would be be unable to understand how the different components related to each other.

The author's solution is to divide the process of segmentation/feature detection into two stages: a background subtraction phase, followed by a process of finer segmentation. In the case of the flags dataset, the first detector would cut out the flags in their entirety from the background to identify the objects. A second phase of feature detection could then proceed *within* each flag, this time segmenting based on colour, attaching a number of "sub-objects" to each object. Each object is thus described as a basic hierarchy – another technique employed to make sense of scenes in conventional computer vision.

The sub-objects are processed into shapes in the same manner as the main object. Descriptors relating to the sub-objects can then be used to help with the classification of the main object. Examples may include the number of "red shapes" within a given flag. Thus, as well as being able to represent content as a basic hierarchy of objects and sub-objects, additional information becomes available to help with the process of classifying the object. Furthermore, by dividing the feature detection stage into two parts – background subtraction and segmentation – the task is made more straightforward; the segmentation stage in particular can be evolved more easily, because it need only operate within the confines of specific objects defined by the background segmenter. This is a process in some ways analogous to binary decomposition, which was shown in Chapter 4 to learn solutions more rapidly than multi-class classification performed during a single run.

The approach can actually be extended further, such that the sub-objects can themselves be classified. In the case of the Flags dataset, the sub-objects can be classified as "stripes", "crosses" etc. The types of sub-objects can thus be of assistance when classifying the main object – it can make use of features such as the number of "white crosses" within the flag. Later in this chapter, we shall also see an example where sub-object classification can yield useful information directly. A diagram of this second vision framework, which comprises up to four different evolved vision components, is shown in Figure 7.3.

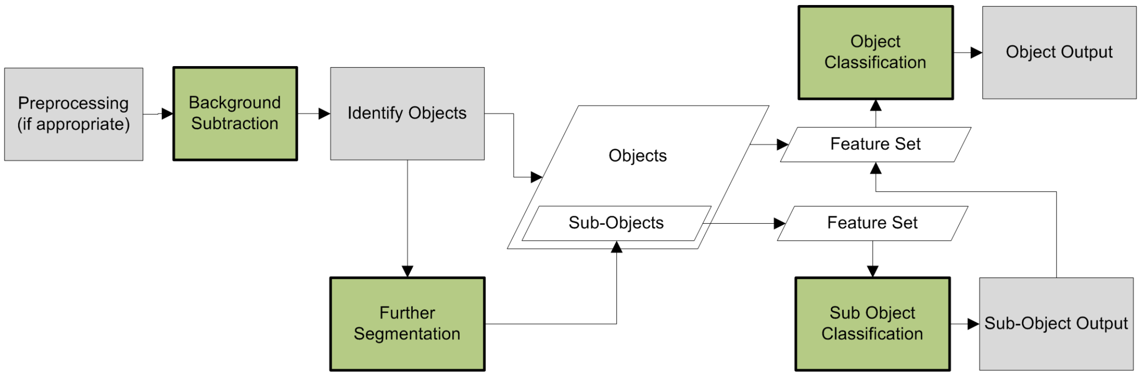


Figure 7.3: The author's updated vision system framework. It is composed of up to four evolved components (shown here in green). Following a process of background subtraction, which locates the objects in the scene, a secondary process of segmentation may be initiated to detect sub-objects within each object, distinguished by different material. The sub-objects may themselves be classified, and that information can either be outputted directly or added to the object classification feature set. The objects themselves may also be classified to produce useful high-level output.

7.3.2 Discussion

One could argue that any framework for designing systems will usually introduce limitations in flexibility, which could be at odds with any claims of genericity. Nonetheless, there are two good reasons to make use of such a architecture. Genetic Programming is a powerful tool for automated feature selection and learning. . . provided the task itself is not unduly ambitious. While it is tempting to provide the GP toolkit with a generous set of components, sufficient for Turing-complete programming, and to leave it to evolve a complete vision system in a single run, the reality is that the learning process would probably not yield anything particularly useful, at least in the limited time available for artificial evolution. As we saw in Chapter 5, the search space can grow to enormous proportions quite easily. Since time and processing are always limited, a practical reaction is to divide the task into separate components. The author believes that this partitioning into a pipeline of separately-evolved stages is critical as it allows a range of domain-specific operators to be used in each stage without causing an explosion in the dimensionality of the search space.

The second advantage of using a framework is that users do not need to start from scratch each time they attack a new problem. Given a fixed set of objectives or processes, it becomes easier to automate the production of systems in general, and to ensure the useful ideas and concepts discussed earlier in this thesis are made use of. Indeed, the author's software encapsulates the vision system architecture into a single user interface, permitting the advanced user to generate vision systems within a couple of hours.

7.4 Jasmine

Having developed a reasonably generic vision system architecture, which is dependent upon one or more evolved vision components, a significant challenge is the creation of a unified interface by which the user can provide training data to the system and thus produce vision systems with minimal effort. The author was inspired in this regard by work by Brumby *et al.* [81] on a project called GENIE ("Genetic Imagery Exploration"), which included a basic user interface named *Aladdin* for marking up images for binary segmentation. The author's own interface, named Jasmine[4], which was initially similar in functionality to Aladdin, later extended into a reasonably substantial platform for producing evolved vision systems. Besides its support for object recognition as well as segmentation, Jasmine is distinguished from Brumby's interface by its support for arbitrary numbers of feature classes; users may create and use as many classes as they wish. It has already been mentioned in passing that classes can be embellished with meta-data, for instance stating that it is a "background" class: that information is used later by the vision system.

The software is project-based, meaning that users can import a series of images, create training data and evolve solutions, with all the information being saved together in one file. This permits a problem to be shared easily, and for a user to resume their work quickly. Training images may be added to the project either from pictures on disk or directly from a compatible video camera.

[4]Jasmine was a character featured in the Disney adaptation of Aladdin, and has the advantage of starting with 'J', a crucial requirement for all applications written in Java. In line with the best traditions of computer science, the name is a recursive acronym which the author fancies as standing for "Jasmine: A Segmented IMage Notation Environment", or (a suggestion by Riccardo Poli) "Just Another Simple iMage Interpretation Environment".

In this section, the author will describe how a user may go about using Jasmine to produce a vision application concerning the recognition of different shapes of pasta. The process starts by choosing appropriate classes, in this case "pasta" and "background" and labelling the training images appropriately.

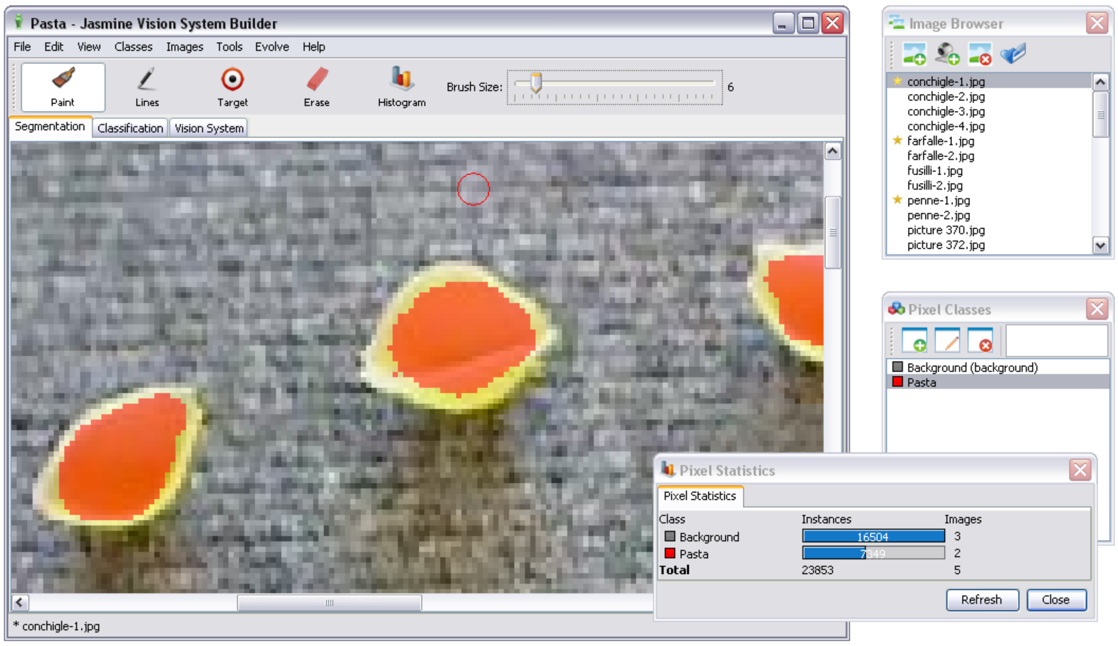
7.4.1 Training Procedure

By this point the reader should no doubt be familiar with the concept underlying the author's feature detection procedure, which works by making a decision about each individual pixel. The detector is evolved using a supervised learning paradigm in which programs are selected according to their performance against a training dataset: a series of examples for which the "correct" class has been established. Since most images are composed of tens if not hundreds of thousands of pixels, generating meaningful training sets can be time intensive. One way to create the training set is to create a binary mask to fit over a given image, which defines which pixels in the image belong to a given class and which do not. This can be performed using off-the-shelf graphics software such as Adobe Photoshop®. However, in the author's experience, marking up *every* pixel in an image is not always a good idea, as some parts of the image cannot be assigned to one class with absolute certainty. If the user does fully mark an entire image, the training data may contain contradictions that are broadly invisible to the human editor (seeing only the bigger picture) but nonetheless confuse the learning system.

The author started his experiments using this approach and can attest that such masks are tiresome to create! Furthermore, they do not readily permit the marking of more than two classes. For this reason the author commenced development of Jasmine, which permits the user to mark up parts of a scene by roughly "painting" overlays, one per class, onto one or more training images (see Figure 7.4). The whole image need not be marked up, the user can select those pixels most representative of a particular material – the Genetic Programming process will discover the boundaries itself. In Chapter 6 it was shown how samples from multiple images could be taken without using excessive amounts of memory. All the feature detectors demonstrated at the end of Chapter 6 were evolved using datasets produced in Jasmine.

7.4.2 Training Data Selection

Painting classes directly onto images is a reasonably painless means by which to produce training data, but each stroke of the mouse can add hundreds of new pixels to the training



Pixel Statistics

Class	Instances	Images
Background	16504	3
Pasta	7349	2
Total	23853	5

Figure 7.4: Creating training data via a graphical user interface in which the user simply "paints" the different classes onto a training image. We shall use the detection of pasta as an example throughout this chapter.

set. It does not take long before the training set is filled with hundreds of thousands, it not millions, of pixels. As was mentioned in Chapter 6, each pixel individually does not convey a huge amount of information; indeed two adjacent pixels will often exhibit similar characteristics. Therefore, there is good cause to reduce the training set size such that evolution can proceed more briskly. Preliminary experiments by the author reveal that segmentation training data can typically be pruned by about 80% with no significant loss in accuracy on test data.

Having decided to prune the training data in some way, we now turn to reasonable methods for doing so. The author's approach is to discard a certain proportion of training data while maintaining the approximate class distributions, similar to the technique employed in stratified k-fold cross validation. This first method shall hereafter be referred to as "by Class" selection. Another sensible suggestion in this regard was made by Roberts and Claridge [151], who proposed using k-means clustering to group the training data into ten classes, after which an equal number of points are taken from each class. This, they said, would ensure that each type of pixel would receive an equal representation in the training data, and would permit the training data to be smaller. Their approach is referred to hereafter as “by Cluster” selection. However, no results were published to prove their claim one way or the other. Table 7.1 shows the results of an experiment conducted by this author in order to determine the difference between the two techniques on imaging datasets. The training dataset was culled to approximately 20% of its former size on three different vision datasets, using both techniques. The results, averaged over 50 runs, are the error for the classifiers on test data, which was left full-size. The result of an independent samples T-Test is also shown.

Data Set	By Class	By Cluster	Result
Flags	0.048	0.057	By Class is better, p = 0.0369
Pipelines	0.039	0.042	By Class is better, p = 0.0008
Leaves	0.158	0.191	By Class is better, p = 0.0121

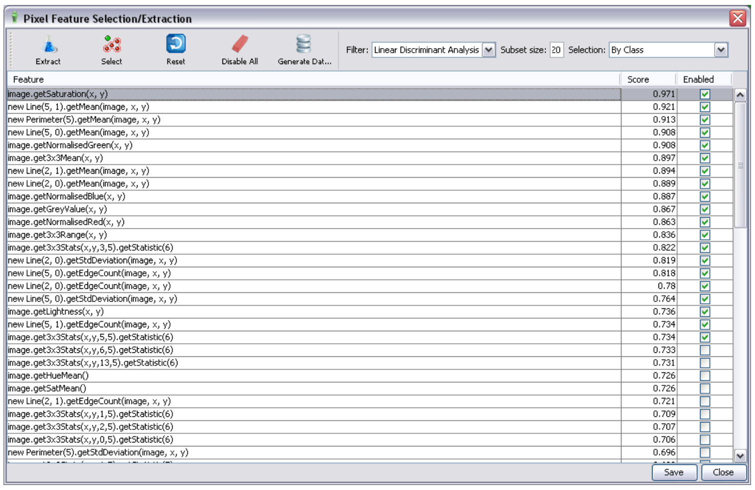
Table 7.1: Comparison between different training data selection techniques

The results show that the “by Class” method produced better results on each of the datasets, which in some ways is not surprising because the test set is more likely to be similarly stratified to the culled training set, so the experiment is somewhat biased. However, if we are interested purely in the results, it would appear that the clustering method does not, in fact, offer any advantage.

7.4.3 Feature Selection

Once the user is satisfied with his/her artistic endeavours, he/she may employ either of the above techniques in order to select a fixed number of samples; the relevant pixels are extracted and formed into a set of training data for the feature detector’s evolution. However, before the process of evolution is started, it is worthwhile selecting the most useful features. The feature selection techniques described in Chapter 6 are implemented within Jasmine, and can be invoked to automatically cut the feature set down to a suitable size, shown in Figure 7.5.

Returning to our pasta example, the figure shows that the most useful feature was identified as the saturation level of a particular pixel, which appears to be a wise choice – the pasta shapes are a reasonably saturated shade of yellow, but the background in this simple example is textured but mainly grey. The second highest feature is a texture feature. The user may still choose to disable the colour features – forcing the software to develop a system that works on grey-scale images, or disable features that are computa-



Filter: Linear Discriminant Analysis Subset size: 20 Selection: By Class

Feature	Score	Enabled
image.getSaturation(x, y)	0.971	✓
new Line(5, 1).getMean(image, x, y)	0.921	✓
new Perimeter(5).getMean(image, x, y)	0.913	✓
new Line(5, 0).getMean(image, x, y)	0.908	✓
image.getNormalisedGreen(x, y)	0.908	✓
image.get3x3Mean(x, y)	0.897	✓
new Line(2, 1).getMean(image, x, y)	0.894	✓
new Line(2, 0).getMean(image, x, y)	0.889	✓
image.getNormalisedBlue(x, y)	0.887	✓
image.getGreyValue(x, y)	0.867	✓
image.getNormalisedRed(x, y)	0.863	✓
image.get3x3Range(x, y)	0.836	✓
image.get3x3Stats(x,y,3,5).getStatistic(6)	0.822	✓
new Line(2, 0).getStdDeviation(image, x, y)	0.819	✓
new Line(5, 0).getEdgeCount(image, x, y)	0.818	✓
new Line(2, 0).getEdgeCount(image, x, y)	0.78	✓
new Line(5, 0).getStdDeviation(image, x, y)	0.764	✓
image.getLightness(x, y)	0.736	✓
new Line(5, 1).getEdgeCount(image, x, y)	0.734	✓
image.get3x3Stats(x,y,5,5).getStatistic(6)	0.734	✓
image.get3x3Stats(x,y,6,5).getStatistic(6)	0.733	☐
image.get3x3Stats(x,y,13,5).getStatistic(6)	0.731	☐
image.getHueMean()	0.726	☐
image.getSatMean()	0.726	☐
new Line(2, 1).getEdgeCount(image, x, y)	0.721	☐
image.get3x3Stats(x,y,1,5).getStatistic(6)	0.709	☐
image.get3x3Stats(x,y,2,5).getStatistic(6)	0.707	☐
image.get3x3Stats(x,y,0,5).getStatistic(6)	0.706	☐
new Perimeter(5).getStdDeviation(image, x, y)	0.696	☐

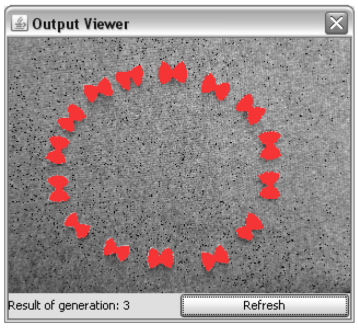
Figure 7.5: Feature selection is performed directly within Jasmine.

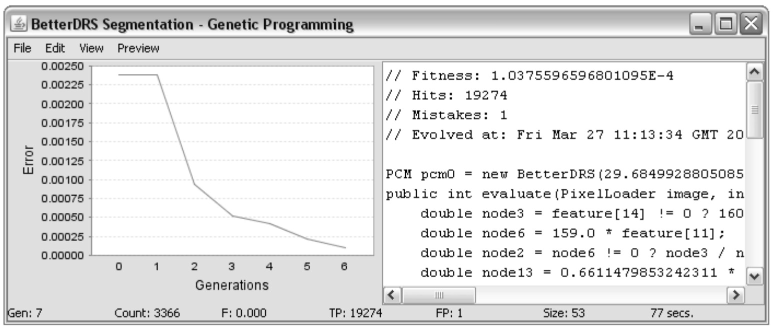
tionally expensive in order to evolve a more efficient detector.

7.4.4 Feature Detector Evolution

Following the process of feature selection, a final dataset is produced and fed to the author's GP classification system (see Chapter 4). Jasmine can automatically choose the appropriate classifier representation by running short mini-evolution runs; it may choose either DRS2 or ICS. As we saw, ICS tries many of the tricks a human would in finding a suitable classifier, including performing binary decomposition, experimenting with different parameters and trying out classifier ensemble techniques. In Chapter 4 a number of classification representations were described; it was shown that classifier ensemble techniques can improve the accuracy of the classifier, but they have the disadvantage of slowing down the process of classification (several decisions must be made per pixel instead of one). In the case of the feature detector, which must make decisions for tens of thousands of pixels, this is not ideal but may depend on circumstances – the user is given the option whether to use classifier ensembles or not, although by default the option is switched off for feature detection.

While the feature detector is being evolved, a live preview of its performance on the training images is shown (see Figure 7.6). This can give a good indication of how the detector performs on entire images, especially when the training data is taken from just a few representative samples on each. Sometimes it becomes clear that certain characteristics in the image are under-represented. Based on visual observation, the user may choose to update the training overlays; often discrepancies become clear quite soon into





```
// Fitness: 1.0375596596801095E-4
// Hits: 19274
// Mistakes: 1
// Evolved at: Fri Mar 27 11:13:34 GMT 20

PCM pcm0 = new BetterDRS(29.6849928805085
public int evaluate(PixelLoader image, in
    double node3 = feature[14] != 0 ? 160
    double node6 = 159.0 * feature[11];
    double node2 = node6 != 0 ? node3 / n
    double node13 = 0.661147985324231l *
```

```
Gen: 7    Count: 3366    F: 0.000    TP: 19274    FP: 1    Size: 53    77 secs.
```

Figure 7.6: As the GP evolves a solution, the best of run individual can be run immediately on training data to visualise how well the program is performing

the evolutionary process. Thus the user can iteratively improve and fine-tune the training data to produce the desired result.

Once the classifier has gobbled up the data and completed its business, the evolved feature detector is saved and automatically compiled to Java byte-code. This means the code can be executed without the need for interpretation of the GP tree, and runs at the same speed as hand-written code.

We have seen so far the process necessary for feature detection (in this case background-subtraction) on the simple pasta dataset. The process of segmentation (which yields the sub-objects) can proceed in a similar manner. However, the pasta experiment does not demand the extra step, so we can skip forward to the object classification stage.

7.4.5 Object Classification in Jasmine

Once the feature detector has been evolved satisfactorily, the user may deploy it on the training images such that individual pieces of pasta are cut out as objects. The user may then define another set of classes that define the higher-level characteristics of the objects. In this case this may be the *type* of pasta[5]. As will be shown shortly, the number of classes required in some circumstances can be quite large – over 30 in some cases. The author does not recall having seen classification by GP beyond 16 classes, which itself is unusual: usually the number is around 4–5. In Jasmine each object can simply be clicked on to assign it a particular class.

When the user's mouse finger is tired, the shapes may be used to assemble another set of training data, this time using the features described in Section 7.2.1. These features are

[5]'penne', 'conchigle', 'fusilli', 'farfalle', etc

passed through a second process of feature selection, this time with respect to the object classes, and the classification system is invoked once more. Due to the relatively small number of shapes within the image relative to the number of pixels, the system can be indulged in using classifier ensembles, which generally improve accuracy (in the author's experience).

Again, if pasta were more complex, we may have run a classifier for sub-objects first. In the case of the vision system, the sub-object classifier must be run before the object classifier, such as to provide the object dataset with more information. Such practicalities are taken care of by the author's software – the end user need only concentrate on evolving the components, which can then be added to a visual representation of the vision system in Jasmine.

After the evolution of each stage, the evolved component can be evaluated against a series of test set images, not used in training, to assess how well the evolved solution copes with unseen data. The software can calculate accuracy metrics for each component in the vision system.

7.4.6 Vision System Object

A separate vision-system program stores all the evolved components and performs the necessary in-between processing. Following the evolution of all necessary components, this vision system program may be exported into outside applications of the user's choice. The vision system raises events at appropriate times during processing, permitting other software to interact with it. In the next section we shall see how the vision system performs.

7.5 Results

This final results section presents examples of the vision systems that can be generated using the author's framework. We shall start with those systems using the author's initial two-stage architecture, then move onto problems that demand the extra steps of the author's second, more complex architecture. Numeric results will be shown wherever possible; the quality and applicability of the solution will be assessed in each case.

7.5.1 Pasta Shape Recognition

Approaching the end of this thesis, the reader may be rather tired of pasta, or indeed vaguely hungry. Nonetheless, having discussed it during this chapter, it seems fair to present the results of the vision system. The feature detector, which cuts out pasta shapes, was evolved using the author's DRS2 technique in approximately 90 seconds on a modest 3Ghz single core machine. It achieved a high accuracy of 99.9% on training data. It was then put to work on a series of previously unseen images for the purposes of detecting objects, in this case individual pieces of pasta, of five different types. The detector yielded a sensitivity of 100% and a specificity of 97.4% with respect to detecting the pasta shapes on test images. These are high levels of accuracy; we shall shortly see whether they yield in accurate classifiers – a better test of their utility.

Each object was labelled into one of 5 different classes corresponding to different pasta types; producing an object training set of approximately 500 samples and a test set of 1250 samples[6]. This is a straightforward process of clicking on each shape in each image to assign it an appropriate class. A shape classifier was then evolved using ICS in approximately 2 minutes and was able to achieve 96.8% accuracy on the object dataset. Thus the total accuracy of the system on test data is 96.8%, which corresponds to the likelihood that any piece of pasta in the images can be detected, located and correctly identified. An example of the system working this is shown in Figure 7.7 below. The system was developed in a morning, which included generating the training data.

7.5.2 Hand Gesture Recognition

The second experiment concerns the recognition of hand gestures. Although this kind of problem has been tackled effectively by researchers using other learning techniques, notably by Starner [152], it has not previously been investigated using Genetic Programming[7]. The author devised a set of ten different hand gestures, some of which are shown in Figure 7.8.

A training set of images was developed using images captured using a webcam, which can be loaded directly into Jasmine. Two classes were created: "skin" and "non-skin", and a series of training samples were marked up in the usual fashion.

[6]The author's mouse finger was quite tired at the end; fortunately each training image only contains instances of one type of pasta – Jasmine includes an option to classify every object in an image into a particular class which reduces the number of mouse miles to be travelled!

[7]Although it has been used for hand detection in shape silhouettes by [103], see page 58.

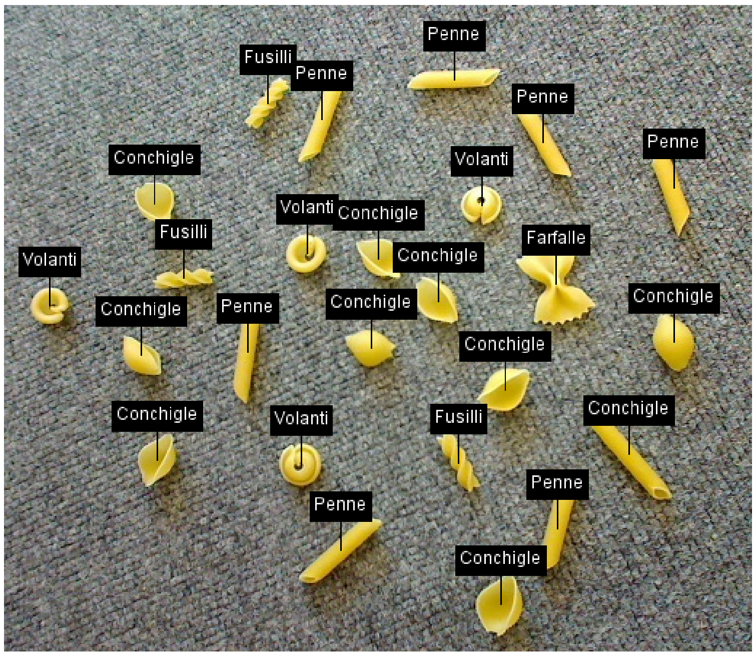


Figure 7.7: The Pasta Recognition Vision System, working on an unseen image.

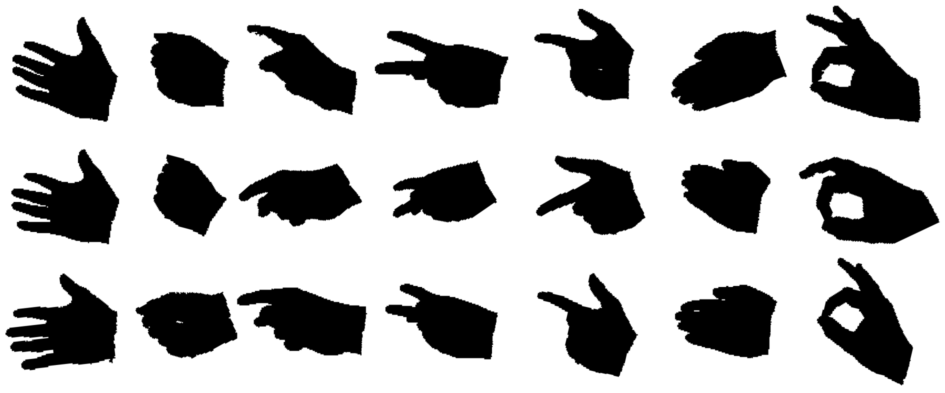


Figure 7.8: Examples of some of the hand gesture types.

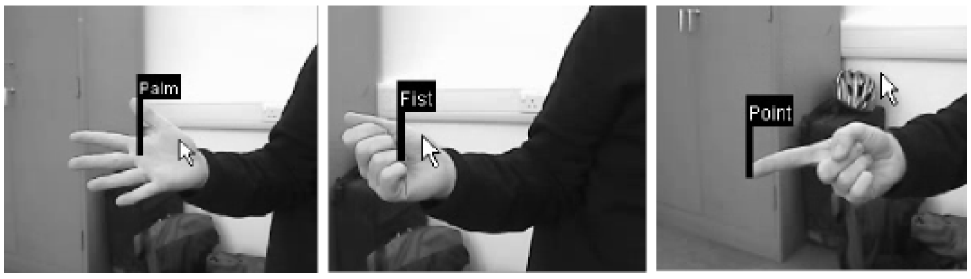


Figure 7.9: The posture vision system working on a webcam in real time.

A background subtracter was evolved to distinguish the two, and achieved an accuracy of 96.4% on a reduced training set comprising approximately 15,000 pixels, and accordingly could be evolved rapidly. The feature selection process was run as normal, but some of the computationally expensive features were disabled manually to ensure the detector could run quickly. The detector was able to cut out hands from test images with a sensitivity of 99.6% and specificity of 94.3%.

The ten different classes were then added to the project, and each hand "object" was then labelled with the appropriate class. An object dataset was produced, consisting of some 600 hand images, with 200 used for training. Unsurprisingly, the feature selector revealed that shape descriptors were the best available means of classifying these objects.

ICS was put to work learn to classify the gestures and achieved an accuracy of 96.2%, yielding a final accuracy per gesture of 95.8%. The vision system was developed in approximately 2 hours. Since the primary application of gesture recognition is human-computer interaction, it is essential that the vision system work in real-time. The system was able to recognise hand gestures at a rate of 6 frames per second, shown in Figure 7.9.

7.5.3 Lesion Classification

In the last chapter, the author's generic feature detector was applied to the task of skin lesion segmentation (see page 162). Without a ground truth from an expert, it is difficult to assess how accurate the author's segmenter was, although from a layman's point-of-view, the system seemed to identify the lesion boundaries accurately, was not affected by hairs or other skin features, and could work on a variety of different skin colours. It has already been stated several times that the performance of a vision technique can also be assessed by its *utility* – is its output meaningful enough for higher-level decisions to be taken? Fortunately it is somewhat easier to find images from medical databases which state the type of lesion – either a benign nevus (mole), or malignant melanoma (a particularly unpleasant form of skin cancer). Although it is sometimes difficult to tell the two apart (malignant melanomas will often grow out from moles), medics have devised an "A-B-C-D" test to distinguish malignant melanomas, which are typically A-symmetric, have irregular B-orders, uneven C-olor, and have a D-iameter over 6mm. Although the last criterion is difficult to measure from images when there is no idea of scale, the author's vision framework should be quite well placed to assess the first three and classify the regions previously extracted using the lesion feature detector.

A set of 68 images were acquired by the author, which had been identified by dermatologists[8]. The images were loaded into Jasmine and processed in the same manner as was previously described. A segmenter was evolved to cut out the lesions, with an object classifier later employed in order to classify them as being malignant or not. Since it was difficult to acquire a large number of appropriate images, 10-fold cross validation was used to assess the result of the classifier, which achieved an average accuracy of 85.8% on test data following 50 runs.

This result shows that the author's system can extend the functionality of vision components evolved by other GP researchers, and also goes some way to demonstrating that the output of the segmenter is meaningful.

Despite attempts to acquire the original datasets cited by other researchers in their results, it was unfortunately not possible to acquire the same sets. In order to develop some means of comparison, or an assessment of the difficulty of the images themselves, the author developed a web-based training system by which humans could learn to discriminate between different lesions by following examples after an introduction to the "A-B-C-D" rule. They then had their new ability tested. The system used the same procedure as the GP system, with 90% of the images used for training and 10% reserved for testing. The results from 50 respondents were interesting: the average error on test images was 81.3%, indicating that the author's GP system is competitive with amateur humans.

7.5.4 The MNIST Dataset

Optical Character Recognition (OCR), has been mentioned at several points during this thesis, as it is a good example for a variety of different processes, including thresholding, template matching or shape analysis. Indeed, the author's software has already been put to work on the detection process for musical notes. The cogitant reader have considered whether the author's system is suitable for other character recognition tasks. A back-

[8]The images came from the Dermnet Skin Disease Image Atlas, www.dermnet.com.

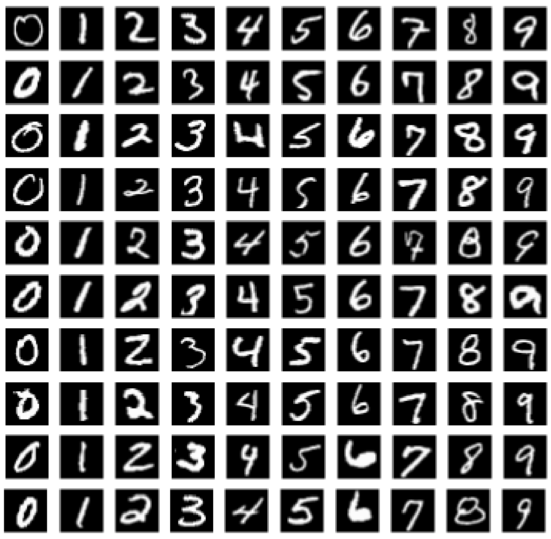


Figure 7.10: Random examples selected from the MNIST database.

ground subtractor can be evolved to cut out shapes, and some of the shape descriptors may do a good job of characterising letters.

To compare its performance relative to other techniques for this task, the author's system was applied to the MNIST dataset [153], a large set of hand-written digits (0–9), collected from hundreds of writers. The dataset consists of 60,000 training samples and 10,000 test samples (see Figure 7.10 for a few examples). Although the recognition of printed Latin characters is regarded by some as a solved problem, the recognition of *hand-written* characters remains a more difficult task, due to the high variation in written forms.

Unfortunately Jasmine could not be used for this purpose, because the author did not feel up to hand-labelling all the digits manually! The author instead wrote a programmatic interface to produce the training set. A background detector was evolved by GP, which was trivial in nature. A process of feature selection took place in order to select the most useful features. The system was then left to its own devices to develop classifiers.

The system took, on average, 50 minutes to produce a multi-class classifier using ICS. The average accuracy following 10 runs was 92.7% on training data, and 90.6% on test data. The results, compared to other approaches, are shown in Table 7.2. Although some techniques in this case, notably the Support Vector Machine, achieved significantly better results than the author's system, the result by ICS is nonetheless reasonably competitive with other approaches, and quite reasonable for a generic system! The addition of further features may help improve the result further.

Dataset	Result
ICS	90.6 ± 1.3
Neural Network	88.0
K-NN	95.0
SVM	98.4

Table 7.2: Results on the MNIST database test data.

We saw in Chapter 3 Teredesai used Genetic Programming to recognise characters on the NIST dataset (of which MNIST is a subset). Teredesai used a set of image features specifically designed for the purposes of OCR. Teredesai reported results of binary classifiers *per class* of between 95.9% and 97.6% each. Assuming the best case, and equal class distributions, this yields an overall accuracy of 0.976^9 or 80.3%. This is the closest comparison available from other Genetic Programmers; again it indicates that the author's generic approach is competitive with the more specific attempts by other GP researchers.

7.5.5 Flag Recognition

Moving onto the author's more complex vision system architecture, we encounter once more the author's Flag dataset, which comprises a number of flag images captured using a webcam. The dataset consists of images of flags from 16 EU member states. Since flags are primarily defined by their colour, the dataset is a test of the vision system's ability to recognise colours accurately. The first stage is to detect the flag images from the background (the author's desk), which is relatively straightforward given its reasonably plain nature. Accordingly the flags could be extracted on all occasions from the test set following the evolution of an appropriate background detector by GP. The second stage is then to identify the colour of objects within each flag – 7 classes were identified: Red, Yellow, Green, Dark Blue, Light Blue, White, Black; these were inputted into Jasmine and a further set of training data pixels was produced. Following the evolution of the colour segmenter, on the test set the colour segmenter achieved an accuracy of 91.5%. This accuracy was boosted to 97.1% by the author's classifier ensemble technique.

The segmenter permitted the system to identify the sub-objects within each flags; the individual coloured shapes that make up the flags' designs. Four further sub-object classes ('long stripe', 'short stripe', 'cross' and 'other') were added to the project, and instances

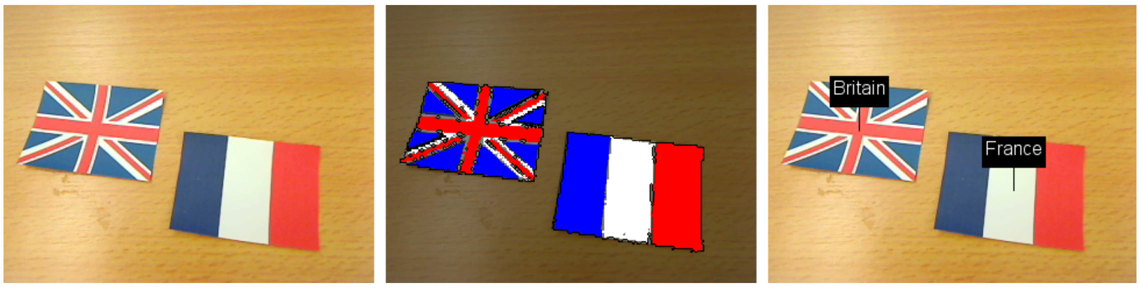


Figure 7.11: The Flag vision system in action. The Flags are cut out from the background and then segmented in order to yield sub-objects within each. This information is then used to decide the class of each flag, displayed on the right side.

of each were labelled to generate a sub-object dataset. A sub-object classifier was then learned by ICS, which achieved an accuracy of 98.2%.

Given this information, including the other colour and shape information, the flags themselves were finally classified, again using ICS, which achieved an accuracy of 95.9%. The accuracy per flag therefore was 95.9%. The vision system was developed in approximately 2 hours.

7.5.6 Automatic Number Plate Recognition (ANPR)

We shall complete the results in this section with a task relating to images in a much less constrained environment, which again makes use of the author's more complex architecture. The problem concerns the detection, location and reading of car number plates. Over the last decade or so, ANPR systems have become quite ubiquitous, and are used extensively by the police, in traffic cameras, and in petrol station forecourts. Although many ANPR machine vision systems use infra-red filters to identify number-plates optically, the author developed a similar system using pure vision software. The system was trained on a series of images taken from cars in a local car park, and tested on images of cars in motion, travelling past the author at 50–60mph.

The first stage is to detect the number plates using a background subtractor. As before, examples of numberplates and non-numberplates were marked up on Jasmine, with a suitable discriminator learned by GP. The evolved solution achieved an accuracy of 99.0%, following just 40 seconds of evolution. Since UK rear number-plates are a relatively specific shade of yellow, the background-subtractor was reasonably straightforward to construct. However, there are often other similarly coloured objects within a scene. Two object classes were created to distinguish true numberplates from other yellow regions, and another object dataset was created. A classifier was developed in order to identify the true number plates, this time based on their shape; it was essentially a rectangle-finder. Although the author's system seeks to classify sub-objects before objects (to aid the object recognition process), if the object is found not to be rectangular, then any identified digits are ignored.

The third stage of evolution was to develop a segmenter that can accurately distinguish characters on the number plate as sub-objects. A separate set of training data for this purpose was painted in Jasmine and another segmenter evolved. The segmenter was then deployed into the ANPR vision system. The sensitivity of the system, or the probability of any letter being correctly detected, was 99.4%. Fortunately the detector doesn't have to detect *every* pixel in the letter, but sufficient ones to identify its shape.

Each character was then labelled appropriately, yielding a dataset whose members represented 34 different alphanumeric classes (all numbers and all letters apart from Q and I). This is a significantly larger number of classes than has previously been tackled by GP vision researchers, or indeed in GP OCR tasks. The final process, of sub-object classification, was then initiated. Each letter in each training numberplate was labelled with the appropriate sub-object class, and a classifier was evolved using ICS. Each letter was classified with an accuracy of 98.4%. This means that the probability of correctly segmenting *and* classifying a given letter by the system is 97.8%. If a number plate is generally composed of 7 letters, then the probability of getting the whole plate right is 85.6%. The vision system was constructed in the course of a day, which included taking pictures of the vehicles. Figure 7.12 shows the process in action.

7.6 Applicability

In this chapter the author has presented a framework based on learning by GP for producing vision systems. Throughout, the features and learning systems have been assessed according to their genericity, or applicability to a wide range of problems. Indeed the examples just presented are representative of a reasonably diverse set of tasks – more diverse, at least, than has been reported by GP researchers to date. Still, there are a number of limitations to the applicability in this approach, which merit a brief discussion of their own.

One limitation is manifested in results on COIL-100 dataset [154]. The dataset consists of images of 100 different objects. Each object is rotated 360° about an axis roughly

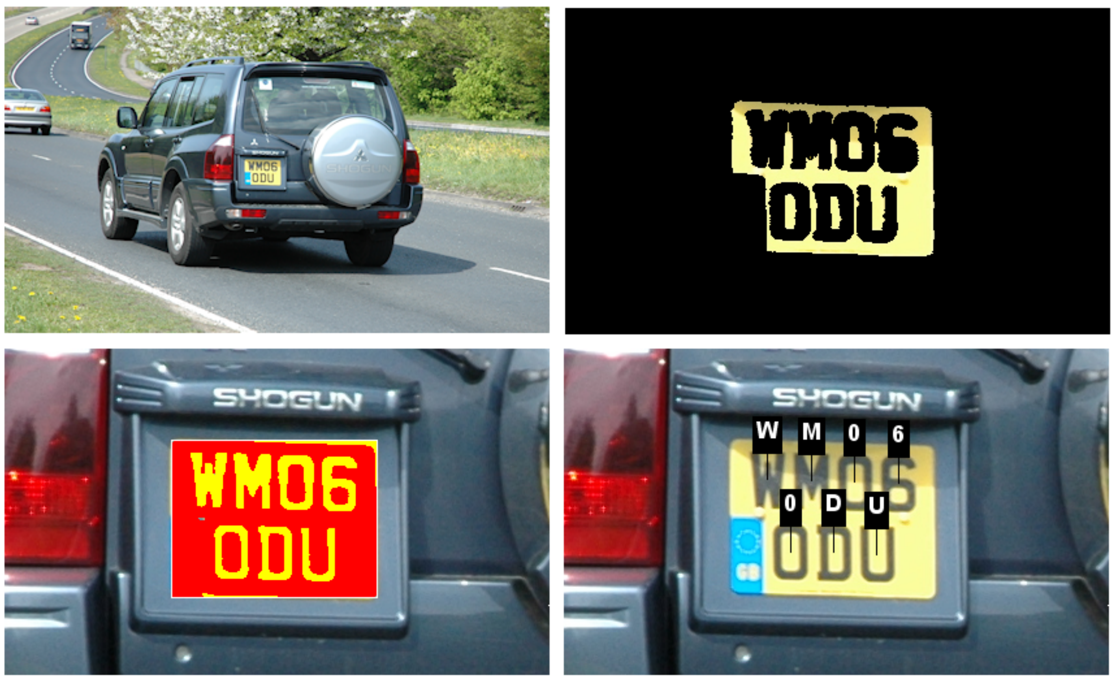


Figure 7.12: An Evolved Vision System Recognising and Reading Car Licence Plates. The background segmenter (top right) identifies the areas of number plate. The segmenter (bottom left) then cuts out the letters, which are individually classified and labelled in order to read the plate (bottom right).

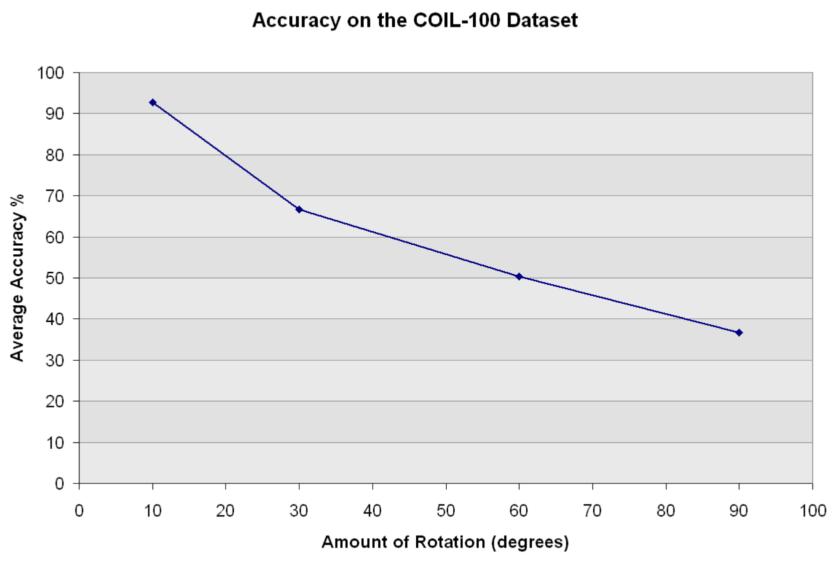


Figure 7.13: Results on the COIL-100 Dataset, (attained using cross validation). Results are shown for a number of angle ranges. As the range of angles increases, so the performance of the classifier decreases.

perpendicular to the camera view in 5° increments. The author's system, however, uses features that are not invariant to this type of rotation. An experiment was run to assess whether the author's software could develop a vision system could distinguish among all the objects, and to analyse the extent to which the software is invariant to this kind of rotation. The author's system was put to work on the dataset in much the same way as before, first segmenting the objects from the background, then performing a finer process of colour segmentation, then evolving an object classifier.

The results of the evolved solution for different amounts of rotation are shown in Figure 7.13. For small changes in angle, the author's system is reasonably accurate (achieving an accuracy of 92.7% for rotations within $\pm$ 10°), certainly given the number of classes, which is far beyond the number usually tackled by GP researchers investigating multi-class classification. Nonetheless, it is clear that the author's vision systems are inherently limited to two dimensional problems – the figure shows that the performance of the system degrades markedly as the extent of rotation is increased. A new set of features is necessary to render the author's software more capable of a wider range of tasks in three dimensions. Of course any function that returns 3D invariant information could be plugged into the author's framework.

In the author's opinion, the other key limitation of the existing framework is its inability to cope with occluded or overlapping objects, which would otherwise be treated as a single large object that would probably be mis-classified. Although the feature detector can be trained to divide objects using a number of metrics, including edges, it is difficult to *insist* that it does so. One solution is to implement an idea (proposed by Campbell [80], see page 49) in which the image is first pre-segmented using a watershed or other unsupervised learning technique. Given suitable parameters, such approaches tend to over-segment but this process also tends to preserve edges. Segments could then be combined together using Genetic Programming to assess their similarity, taking account of edges between. A further discussion of applicability and extensions to this work awaits the reader in Chapter 8.

The addition of extra features may also help improve the accuracy of the vision systems developed by the author's framework. Although it was shown that a given letter in the ANPR task could be classified at a rate of 97.8%, this would lead to a probability of recognising only 85.6% of entire plates entirely correctly, which police or security organisations might prefer to be a little higher before placing orders for the first GP-evolved ANPR system! The author's system works on images in quite a "raw" way – additional preprocessing operations such as normalisation or shape de-skewing may yield more robust results that bring the author's software into the range of genuine commercial applicability.

7.7 Conclusion

This brings us to the end of the experimental work in this thesis. In this chapter we have covered extensions to the author's feature detection software which permits more complete vision systems to be constructed using GP. Some are able to run in real time. In this thesis the author has explored the means by which a reasonably large range of problems can be tackled by a generic GP approach. In the next, final chapter, the applicability and limitations of the author's approach will be discussed in more detail.

Chapter 8

Conclusions

Perhaps one of the reasons we enjoy so-called “optical illusions” is not because they shed some light on what might be going on inside our brains but because they demonstrate that our vision system can, just occasionally, fall short. Our vision system is usually so relentlessly flawless that its failure is something of a novelty! In emulating vision, by contrast, we encounter a seemingly endless number of difficulties, limitations and hidden complexities. In this thesis, the author has tackled a few of the classic problems in machine vision, and attempted to develop practical solutions in an automated fashion. In this final chapter, the author shall make a critical assessment of the applicability and value of the author’s work. We shall start by revisiting some of the concepts that motivated the research in this thesis.

The Automated Production of Vision Systems So exceptional are our eyes and brains that certain tasks in vision feel so trivial as to be tedious. Accordingly, there are a great many tasks that we would like machines to attend to on our behalf. Indeed, if one were to sit down and enumerate all the things that we can do with our eyes, but which we’d prefer a computer to do for us, the list would rapidly become gargantuan. It appears that machine vision is one of those topics for which the number of expedient solutions is outnumbered by the volume of desirable applications.

To develop a “complete” vision system on a par with our own is currently not achievable, so we must restrict our ambitions to solving one specific problem at a time. Indeed, although some commercial vision systems claim to be applicable for “all situations”, they are in fact usually limited to a subset of tasks, usually involving industrial inspection [155, 156]. Faced with the huge amount of human expertise required to create any non-trivial vision system, comes a desire to automate production in some way, which was the primary motivation underlying the work in this thesis.

Extending GP Vision Research The author's work uses Genetic Programming (GP) as its principal learning component. Although Genetic Programming has been applied to vision problems throughout the history of its own evolution, a gap in GP machine vision research provided a second motivation for the author's study: work is usually concentrated on a single component of an overall vision system, for instance pre-processing operations, low-level components or object recognition. As such, those evolved components do not represent working vision systems; and until now, it has not been shown how a more complete, working vision system may be constructed using GP-evolved components.

The Spirit of Automatic Programming A third aim was motivated by another feature of some GP research, which is generally to devise a highly task-specific set of features and parameters, and then show how they may be put together using Genetic Programming [27, 157, 100, 158, 159]. In the author's opinion, this is somewhat contrary to the intention of machine learning, which is to have computers develop software themselves. If the learning system requires customisation for each different problem then it isn't learning enough!

8.1 Assessment

Over the last three years, the author has developed a software framework in which the machine itself is almost entirely responsible for the technical aspects arising from the production of various vision systems. The human user may define the problem with the tools provided, but expertise in machine vision itself is generally not required. In an ideal world it would be left for a domain expert, such as a radiologist, geologist or zoologist, to define the "truth" in each case, leaving the task of identifying patterns to the software. In this section, the author shall critically assess his system in terms of its applicability, quality and competitiveness with other techniques, and determine how well the system meets the original motivations for this work.

In Chapter 2, we saw how many components and parameters may make up a Genetic Programming system, and in Chapter 5 the author suggested how to avoid some of them! The author has tried to show how Genetic Programming can be rendered reasonably parameter-free for a variety of problems, and in certain respects this has been successful: all the results in Chapters 6 and 7, for what they are worth, were obtained from a system that did not require any tuning. In the author's system, feature selection and extraction, training data creation, segmentation and shape classification are all performed automatically.

To what extent is the author's toolkit parameter-free? Is the quality of solutions compromised when the system cannot be tuned to each problem? Indeed, in many cases, such as those discussed in Chapter 5 there isn't a straightforward "best" parameter for a particular solution. For this reason the author developed a classification system which attempts to make intelligent decisions and try out ideas, instead of delegating those decisions to the human operator. Although the binary decomposition process within the system can make this more accurate, trying our different ideas increases the learning time, although this too can be offset by using multiple CPUs. The author fancies that this is similar to the process that a human researcher might undertake while developing a system. It may well be possible to discover parameters that suit a particular problem better, but the time taken in doing so may outweigh the advantages!

8.1.1 Multi-Stage Vision Systems

The author claims to have gone further than other GP researchers by developing vision systems that comprise two or more separate stages of processing, each using evolved components. Multi-stage evolution is not a new topic in GP, but is almost exclusively confined to refinements of the *same task* [20, 151, 160]. In this thesis, the author has shown that Genetic Programming can evolve solutions to a number of image processing operations, and that each stage is sufficiently accurate as to provide meaningful data to the next stage. This would appear to accommodate the requirement of extending GP research, but the question is whether the author's systems are competitive.

8.1.2 Competitiveness

It may be that the machines have more to offer than cheap labour. Given sufficiently well-defined criteria, computers are able to try out possibilities more quickly than can a human, so systems whose development requires some degree of trial-and-error may benefit from more comprehensive exploration if studied by computer. In general, machines are not biased towards a particular approach, so it could be submitted that their exploration of the search space is more fair. Of course, it could also be argued that they also lack the intuition necessary to concentrate their search in promising areas! Still, it may be the case, one day, that the state-of-the-art requires more work than can be accomplished by humans alone.

But to what extent does the author's work represent the state-of-the-art in computer vision today? The author would readily acknowledge that the vision systems produced using his system are not as advanced as some of the approaches taken by vision researchers in recent years. It is the nature of generic systems to follow behind the leading edge to some degree. Still, the author would submit that his system has value: it can be used to generate working vision systems in a fraction of the time that would be taken if being developed manually.

Other considerations, so elegantly managed by our own eyes and brain, such as geometric or illumination invariance, provide substantial stumbling blocks for vision systems and demand significant research in their own right. In some senses, therefore, the author's remit in this work is too broad, especially since it covers both a substantial swathe of computer vision and evolutionary computation; the reader will have noticed a relatively equal split between the two during the course of this thesis. Although issues such as colour constancy, noise reduction and others have been touched upon, the author does not claim to be an expert in these fields. Algorithms have been introduced and integrated into to the author's system, but these may be regarded as "placeholders" rather than advanced systems with which the author would claim satisfaction. Accordingly, the author's framework is designed such that other vision algorithms and learning techniques could be plugged in at a later date to improve its robustness. Still, one of the major benefits of using Genetic Programming is that the imperfections of other algorithms are inherently accounted for!

In some ways, to concentrate on vision *systems* in general rather than vision *components*, then to divide one's loyalties between machine vision and the field of evolutionary computation makes it difficult to examine everything to the detail desired. Perhaps it is better to concentrate more intently on a smaller area, although the challenge of taking on more is sometimes difficult to resist. The author would have preferred to spend more time investigating individual components in machine vision and evolutionary computation, but there is simply too much to cover during the course of three short years!

In Chapter 5 we saw that the author's system was broadly competitive with other machine learning techniques on classification tasks, more so against classifiers published by GP researchers. In vision too, the author's system goes beyond what other GP researchers have published. Indeed, the author's generic system has been used to replicate and extend work recently published by other GP researchers. Still, it is acknowledged that Genetic Programming does not represent the current mainstream approach towards machine vision; indeed, some of the problems attacked by genetic programmers are simplistic. The "edge" offered by Genetic Programming is its ability to solve problems in an automated way; as already discussed, it appears the author's system does offer such an advantage in terms of the speed and ease with which vision systems can be produced. This benefit is most meaningful if the system can solve a host of *different* tasks.

8.1.3 Applicability

As we saw in Chapter 6 and 7, the author's system has been applied to problems encompassing a wide variety of domains; the system was not developed with any particular application in mind. But to what extent can the author claim that his system is widely applicable? Despite claims toward genericity, and a system that doesn't require explicit tuning to particular problems, the reality is that the author's software can solve but a small fraction of the gamut of machine vision systems that exist. The field of computer vision is such that it is not easy (or perhaps possible) to find a set of metrics that can adequately describe all the possible information we might like to draw from an image. Nor is it straightforward to bolt them all together into a single architecture or interface. The further that one proceeds with computer vision, the more different applications come to mind; some can be accommodated, many cannot. There are a number of areas where the author, given time, would choose to expand his vision system with a view to increasing its applicability. Some of these are described later in Section 8.2.

8.1.4 Genetic Programming

Throughout the thesis, the reader may have questioned whether Genetic Programming is the most appropriate tool for developing *all* domain-specific components of the system. As was discussed toward the end of Chapter 5, machine learning paradigms each exhibit their own strengths and weaknesses. Perhaps it is rather inflexible to use Genetic Programming exclusively, when other techniques may be more appropriate in certain situations or for certain tasks in general? Neural networks are widely used in machine vision for various tasks, and Genetic Algorithms are also used for feature selection and other optimisations. The author acknowledges this is a reasonable suggestion, which may benefit future versions of his software. Like much doctoral research, this work has concentrated on the development and use of a specific system, with the inevitable consequence that some more practical considerations are compromised.

That said, Genetic Programming remains a flexible and promising machine learning technique. There are few learning algorithms that are sufficiently generic to allow such a wide range of learning applications. As we saw in Chapter 5, the search space for all but the simplest tasks can grow to astronomical proportions and yet the evolutionary process can pick out competitive solutions within a matter of minutes or hours, which is quite remarkable. The fortunate thing is that while the search space grows, so does the solution space.

8.2 Future Directions

Over the past three-and-a-half years, the author has somehow managed to write slightly over 100,000 lines of code. Still, even while putting the finishing touches to this final chapter, there are still many new possibilities and areas of endeavour that come to mind! In addition to those already hinted at so far in this chapter and previous ones, there are a number of directions for extending this work that can be recommended.

The most straightforward extension is to introduce additional imaging features and shape descriptors in order to enhance the system's ability to generalise. With a feature selection system and the inherent ability of GP to select useful terminals, additional descriptors could be added quite liberally. Such is the breadth of ideas conceived by computer vision researchers, that one could no doubt find many new ideas to take advantage of!

As we saw in Chapter 6, there are certain features which work “out of the box”, as it were; but other feature families, such as Haar-like features, need to have parameters chosen for them in order to be useful. Indeed, certain features may be combined together to produce more useful ones, a common application of principal components analysis. Genetic Programming has also been used for this purpose [161]. Feature extraction, by which parameters for Haar-like features and others may be discovered, was touched upon in this thesis, and indeed appears to work satisfactorily, but like many other things, the author's system may benefit from a more comprehensive study.

The author's vision framework is designed to recognise two-dimensional objects, or at least objects which are captured from a consistent camera direction, although they can be rotated around an axis parallel to the camera's point-of-view. For many applications, this is sufficient, but as vision systems grow more complex, it becomes more practical to define an object by points and edges in 3D space. An intermediate layer, which takes the shapes identified by the feature detector and identifies corners and edges in 3D (perhaps using stereo image pairs), for use by the shape classifier, may help create an object recogniser that is more robust and more applicable.

As has already been suggested, the author's system may be rendered more robust by more advanced colour constancy and exposure manipulation algorithms. The author's software would serve as a useful test bed for evaluating the reliability of such algorithms on a variety of different imaging problems. Although the author's system uses Genetic Programming exclusively, the nature of the system's dataset/classifier interface makes it straightforward to introduce different learning techniques to the fold.

Aside from purely vision related matters, there are a number of additions that could be made to the evolutionary system itself. The author has always considered *co-evolution* to be a fine idea – where features can be evolved alongside the algorithm itself. Binary decomposition, implemented by the author's classifier, helps guarantee that sub-solutions, once learned, can be protected, but this is dependent upon the user defining multiple classes, and does not assist in binary situations. Although the author has implemented a co-evolution system in SXGP there hasn't been enough time, alas, to investigate it thoroughly.

8.2.1 Resources

For readers interested in implementing some of the ideas, the entirety of author's software is available online. You may either run the software directly from the web and generate vision systems of your own (a tutorial is provided) or download the source code and start on Version 2. The URI is:

http://vase.essex.ac.uk/software/jasmine

8.3 Closing Remarks

It is clear that there are many different avenues that interested parties could explore. There is plenty more work to do on the author's system before the word "towards" on the title of this thesis could be safely removed. Therefore, the most suitable conclusion to this thesis is, perhaps, that there is no satisfactory conclusion so far. The field of computer vision remains open and, as yet, there is no truly generic means of approaching vision in general, short of building a complete emulation of the human brain and visual system. The author suspects this will be the ultimate realisation of machine vision. In the meantime, there is a need for more straightforward vision systems, perhaps of the type that author's software is able to create, until such a time as a colossus can be constructed – which may take some time yet. Don't despair though reader, in case you're recalling that your own vision system's evolution commenced some 540 million years before you started reading this thesis – the rate of human science and discovery is such that the colossus may be here sooner than any of us can imagine.

– THE END –

Bibliography

[1] N. Eldredge and S. J. Gould. *Models in Paleobiology: Punctuated Equilibria: An Alternative to Phyletic Gradualism*, chapter 5, pages 82–115. Freeman, Cooper and Co, 1972.

[2] A. Parker. *In the Blink of an Eye*. Perseus Publishing, 2003.

[3] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[4] Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In John J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 July 1985.

[5] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[6] Sean Luke. ECJ: A Java-based evolutionary computation research system v14. `http://cs.gmu.edu/~eclab/projects/ecj/`.

[7] Olly Oechsle and Adrian F. Clark. Feature extraction and classification by genetic programming. In *International Conference on Vision Systems (ICVS)*, 2008.

[8] Huey et al. Rapid evolution of a geographic cline in size in an introduced fly. *Science*, 287(5451):308–309, 2000.

[9] Ingo Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Technical University of Berlin, 1971.

[10] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[11] John R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. MIT Press, 1992.

[12] R. Poli, W. B. Langdon, and N. McPhee. *A Field Guide to Genetic Programing*. http://www.gp-field-guide.org.uk/, 2008.

[13] J. P. Nordin. *A Compiling Genetic Programming System that Directly Manipulates the Machine Code*, pages 311–331. MIT Press, Cambridge, MA, USA, 1994.

[14] Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming. In Thomas Back, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 346–353, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann.

[15] Astro Teller and Manuela Veloso. PADO: A new learning architecture for object recognition. In *Symbolic Visual Learning*, pages 81–116. Oxford University Press, 1996.

[16] Ogino Shirakawa and Nagao. Graph structured program evolution. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1686–1693, 2007.

[17] David J. Montana. *Strongly Typed Genetic Programming*, volume 3, pages 199–230. MIT Press, 1995.

[18] Hitoshi Iba. Random tree generation for genetic programming. In *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation, LNCS*, volume 1141, pages 144–153. Springer Verlag, 1996.

[19] Sean Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, 2000.

[20] Simon C. Roberts and Daniel Howard. Evolution of vehicle detectors for infrared linescan imagery. In *Evolutionary Image Analysis, Signal Processing and Telecommunications*, volume 1596 of *LNCS*, pages 110–125. Springer-Verlag, 1999.

[21] Ankur Teredesai and Venu Govindaraju. Issues in evolving GP based classifiers for a pattern recognition task. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 509–515. IEEE Press, 2004.

[22] Riccardo Poli and Nicholas McPhee. Parsimony pressure made easy. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1267–1274, 2008.

[23] Jeff Palmucci Gilbert Syswerda. The application of genetic algorithms to resource scheduling. In *International Conference on Genetic Algorithms*, pages 502–508, 1991.

[24] M. P Fourman. Compaction of symbolic layout using genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 141–153, 1985.

[25] Fonesca C. M. and Fleming P. J. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, 1993.

[26] J. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *In Proc. Intenational Conference on Computer Vision and Pattern Recognition*, pages 1000–1006, 1997.

[27] Marc Ebner. Evolving color constancy for an artificial retina. In *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 11–22. Springer-Verlag, 2001.

[28] R.A Fisher. *The Genetical Theory of Natural Selection*. Dover, 1958.

[29] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 14–21, 1987.

[30] Darrell Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufman, 1989.

[31] Riccardo Poli. Tournament selection, iterated coupon-collection problem, and backward-chaining evolutionary algorithms. In *Foundations of Genetic Algorithms 8*, volume 3469 of *Lecture Notes in Computer Science*, pages 132–155. Springer-Verlag, 2005.

[32] Huayang Xie, Mengjie Zhang, and Peter Andreae. Another investigation on tournament selection: modelling and visualisation. In *GECCO '07: Proceedings of the*

9th annual conference on Genetic and evolutionary computation, pages 1468–1475. ACM, 2007.

[33] Artem Sokolov and Darrell Whitley. Unbiased tournament selection. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1131–1138. ACM, 2005.

[34] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. TCGA Report No. 89003, University of Alabama, 1989.

[35] M. R. Leuze C.B. Pettey and J. J. Grefenstette. A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 155–161, 1987.

[36] David J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[37] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In *Advances in Genetic Programming 2,* chapter 6, pages 111–134. MIT Press, Cambridge, MA, USA, 1996.

[38] Riccardo Poli and William B. Langdon. On the search properties of different crossover operators in genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 293–301, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[39] Walter Alden Tackett. Greedy recombination and genetic search on the space of computer programs. In *Foundations of Genetic Algorithms 3*, pages 271–297. Morgan Kaufmann, 1994. Published 1995.

[40] Hitoshi Iba Takuya Ito and Satoshi Sato. Non-destructive depth-dependent crossover for genetic programming. In *Proceedings of the First European Workshop on Genetic Programming*, volume 1391, pages 71–82. LNCS, 1998.

[41] Kevin J. Lang. Hill climbing beats genetic search on a boolean circuit synthesis of Koza's. In *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, 1995.

[42] Mengjie Zhang, Xiaoying Gao, and Weijun Lou. A new crossover operator in genetic programming for object classification. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(5):1332–1343, 2007.

[43] Patrik D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, 1994.

[44] S. Hengproprohm and P. Chongstitvatana. Selective crossover in genetic programming. In *In ISCIT International Symposium on Communications and Information Technologies*, 2001.

[45] Alan Piszcz and Terence Soule. Dynamics of evolutionary robustness. In *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 871–878. ACM Press, 2006.

[46] Riccardo Poli, Nicholas F. McPhee, and Leonardo Vanneschi. Elitism reduces bloat in genetic programming. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. ACM Press, 2008.

[47] David Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In K. E. Kinnear Jr, editor, *Advances in Genetic Programming*. MIT Press, 1994.

[48] Jurgen Wakunda and Andreas Zell. A new selection scheme for steady-state evolution strategies. In *In Proceedings of the Genetic and Evolutionary Computation Conference*, pages 794–801. Morgan Kaufmann Publishers, 2000.

[49] J. H Holland and J. S. Reitman. *Cognitive Systems based on Adaptive Algorithms*. Academic Press, 1978.

[50] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.

[51] Jeroen Eggermont, Agoston E. Eiben, and Jano I. van Hemert. A comparison of genetic programming variants for data classification. In *Advances in Intelligent Data Analysis, Third International Symposium, IDA-99*, volume 1642 of *LNCS*, pages 281–290. Springer-Verlag, 1999.

[52] Kenneth Alan De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[53] J.R. Parker. *Algorithms for Image Processing and Computer Vision*. Wiley & Sons, 1996.

[54] E.R. Davies. *Machine Vision: Theory, Algorithms, Practicalities. 3rd Edition*. Morgan Kaufmann, 2005.

[55] E.R. Davies. On the noise suppression and image enhancement characteristics of the median, truncated median and mode filters. *Pattern Recognition Letters*, 7:87–97, 1988.

[56] K.K. Ma T. Chen and L.H. Chen. Tri-state median filter for image denoising. *IEEE Transactions in Image Processing*, 8:1834–1838, 1999.

[57] Nemanja Petrovic and Vladimir Crnojevic. Impulse noise detection based on robust statistics and genetic programming. In *Advanced Concepts for Intelligent Vision Systems, 7th International Conference, ACIVS 2005, Proceedings*, volume 3708 of *LNCS*, pages 643–649. Springer, 2005.

[58] Craig W. Reynolds. Evolution of obstacle avoidance behaviour:using noise to promote robust solutions. In *Advances in Genetic Programming*, chapter 10, pages 221–241. MIT Press, 1994.

[59] Roongroj Nopsuwanchai and Prabhas Chongstitvatana. Improving robustness of robot programs generated by genetic programming for dynamic environments. In *In Proceedings of the Asia-Pasific Conference on Circuits and Systems (APCCAS98)*, pages 523–526, 1998.

[60] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[61] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *9th European Conference on Computer Vision*, 2006.

[62] Ching Yang Wang. *Edge Detection Esing Template Matching*. PhD thesis, Duke University, 1985.

[63] D. Marr and E. Hildreth. Theory of edge detection. In *Proceedings of the Royal Society*, pages 187–217, 1980.

[64] F. John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[65] F. Bergholm. Edge focusing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):726–741, 1987.

[66] C.A. Rothwell, J.L Mundy, W. Hoffman, and V.D. Nguyen. Driving vision by topology. In *International Symposium on Computer Vision*, pages 395–400, 1995.

[67] Christopher Harris and Bernard Buxton. Evolving edge detectors with genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 309–315. MIT Press, 1996.

[68] Christopher Harris and Bernard Buxton. Low-level edge detection using genetic programming: performance, specificity and application to real-world signals. Technical Report RN/97/7, University College London, 1997.

[69] C.H. Chao and A. P. Dhawan. Edge detection using Hopfield neural network. In *Applications of Artificial Neural Networks V*, volume 2243, pages 242–251, 1994.

[70] S.M. Bhandarkar, Y.Q. Zhang, and W.D. Potter. An edge-detection technique using genetic algorithm-based optimization. *Pattern Recognition*, 27(9):1159–1180, 1994.

[71] Byatia P. Srinivasan V. and S.H. Ong. Edge detection using a neural network. *Pattern Recognition*, 27(12):1653–1662, 1994.

[72] Leonardo Trujillo and Gustavo Olague. Synthesis of interest point detectors through genetic programming. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 887–894. ACM, 2006.

[73] P.R. Beaudet. Rotationally invariant image operators. In *International Joint Conference on Pattern Recognition*, 1978.

[74] S.M. Smith. Susan - a new approach to low level image processing. Technical report, Defence Research Agency, 1995.

[75] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66, 1979.

[76] R.B. Ohlander. *Analysis of Natural Scenes*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, USA, 1975.

[77] B. Bhanu and B.A. Parvin. Segmentation of natural scenes. *Pattern Recognition*, 20(5):487–496, 1987.

[78] S. Beucher and C. Lantuéjoul. Use of watersheds in contour detection. In *In Proc. International Workshop on Image Processing, Real-Time Edge and Motion Detection/Estimation*, 1979.

[79] D. Martin, C.Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of the 8th International Conference on Computer Vision*, volume 2, pages 416–423, 2001.

[80] N.W. Campbell, B.T. Thomas, and T.Troscianko. Automatic segmentation and classification of outdoor images using neural networks. *International Journal of Neural Systems*, 8:137–144, 1997.

[81] Steven P. Brumby, James Theiler, Simon Perkins, Neal R. Harvey, and John J. Szymanski. Genetic programming approach to extracting features from remotely sensed imagery. In *FUSION 2001: Fourth International Conference on Image Fusion*, 2001.

[82] Y.L. Chang and X. Li. Adaptive image region growing. *IEEE Transactions on Image Processing*, 3(6):868–873, 1994.

[83] Riccardo Poli. Genetic programming for image analysis. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 363–368. MIT Press, 1996.

[84] Mark E. Roberts and Ela Claridge. An artificially evolved vision system for segmenting skin lesion images. In *Proceedings of the 6th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 2878 of *LNCS*, pages 655–662. Springer-Verlag, 2003.

[85] K. Shanmugan Haralick, R.M. and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3:610–621, 1973.

[86] Andy Song and Victor Ciesielski. Texture analysis by genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2092–2099, 2004.

[87] K.I. Laws. Texture energy measures. In *In Proceedings of Image Understanding Workshop*, pages 47–51, 1979.

[88] Marcos I. Quintana, R. Poli, and E. Claridge. Genetic programming for mathematical morphology algorithm design on binary images. In *Artificial Intelligence, Proceedings of the International Conference KBCS-2002*, pages 161–171, 2002.

[89] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Orlando, Florida, USA, 1983.

[90] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991.*, pages 586–591, 1991.

[91] Shumeet Baluja Henry Rowley and Takeo Kanade. Rotation invariant neural network-based face detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1998.

[92] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[93] Jay F. Winkeler and B. S. Manjunath. Experiments with genetic programming for active vision tasks, 1997.

[94] Walter A. Tackett. Genetic programming for feature discovery and image discrimination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 303–309, 1993.

[95] Simon C. Roberts and Daniel Howard. *Evolution of Vehicle Detectors for Infra-red Linescan Imagery*, volume 1596 of *Lecture notes in computer science*, pages 110–125. Springer-Verlag, 1999.

[96] Daniel Howard, Simon C. Roberts, and Richard Brankin. Evolution of ship detectors for satellite SAR imagery. In *Genetic Programming, Proceedings of EuroGP'99*, volume 1598 of *LNCS*, pages 135–148. Springer-Verlag, 1999.

[97] M. Roberts and E. Claridge. Cooperative coevolution of image feature construction and object detection, 2004.

[98] Mengjie Zhang and Victor Ciesielski. Genetic programming for multiple class object detection. In Norman Foo, editor, *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, pages 180–192. Springer-Verlag, 1999.

[99] Peter Andreae Mengjie Zhang and Mark Pritchard. Pixel statistics and false alarm area in genetic programming for object detection. In *Applications of Evolutionary Computing*, 2003.

[100] Polina K. Spivak. Discovery of optical character recognition algorithms using genetic programming. In John R. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford 2002*, pages 223–232. Stanford Bookstore, Stanford, California, 2002.

[101] David Andre. Learning and upgrading rules for an ocr system using genetic programming. In *In Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pages 27–29. IEEE Press, 1994.

[102] Ankur Teredesai, J. Park, and Venugopal Govindaraju. Active handwritten character recognition using genetic programming. In *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 371–379. Springer-Verlag, 2001.

[103] Michael P. Johnson, Pattie Maes, and Trevor Darrell. Evolving visual routines. In *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 198–209. MIT Press, 1994.

[104] Hu M.K. Visual pattern recognition by moment invariants. *IRE Trans. Info. Theory*, 8:179–187, 1962.

[105] J Sklansky. Measuring concavity on a rectangular mosaic. *IEEE Transactions on Computers*, pages 1355–1364, 1972.

[106] Ossama El Badawy1 and Mohamed Kamel. Matching concavity trees. In *Structural, Syntactic, and Statistical Pattern Recognition*, volume 3138 of *LNCS*, pages 556–564. Springer Verlag, 2004.

[107] Hector Montes and Jeremy Wyatt. Cartesian genetic programming for image processing tasks. In *Proceedings of IASTED International Conference on Neural Networks and Computational Intelligence (NCI 2003)*, 2003.

[108] I. Fujinaga M. Droettboom, K. MacMillan. The gamera framework for building custom recognition systems. In *Symposium on Document Image Understanding Technologies*, pages 275–286, 2003.

[109] Robotcub: An international project on humanoid cognitive systems, 2008. http://www.robotcub.org.

[110] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[111] J. R. Quinlan. Generating production rules from examples,. In *Proc. Tenth Int. Joint Conf. on Artificial Intelligence*, pages 304–307, 1987.

[112] L. Breiman, J. H. Friedman, R. A. Olshen, , and C. J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, 1984.

[113] Martijn C. J. Bot and William B. Langdon. Application of genetic programming to induction of linear classification trees. In *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 247–258. Springer-Verlag, 2000.

[114] Will Smart and Mengjie Zhang. Using genetic programming for multiclass classification by simultaneously solving component binary classification problems. Technical Report CS-TR-05-1, Computer Science, Victoria University of Wellington, New Zealand, 2005.

[115] Durga Prasad Muni, Nikhil R Pal, and Jyotirmay Das. A novel approach to design classifier using genetic programming. *IEEE Transactions on Evolutionary Computation*, 8(2):183–196, 2004.

[116] Yun Zhang and Mengjie Zhang. A new program structure in genetic programming for object classification. In *Proceeding of Image and Vision Computing NZ International Conference*, pages 459–465, 2004.

[117] Thomas Loveard and Victor Ciesielski. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1070–1077. IEEE Press, 2001.

[118] Mengjie Zhang, Victor B. Ciesielski, and Peter Andreae. A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Applied Signal Processing*, 2003(8):841–859, 2003. Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis.

[119] Will Smart and Mengjie Zhang. Probability based genetic programming for multiclass object classification. Technical Report CS-TR-04-7, Computer Science, Victoria University of Wellington, New Zealand, 2004.

[120] J. N. Kapur. *Maximum Entropy Models in Science and Engineering*. Wiley-Interscience, 1990.

[121] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[122] W. B. Langdon and B. F. Buxton. Genetic programming for combining classifiers. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 66–73. Morgan Kaufmann, 2001.

[123] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[124] John Koza. Genetic programming, 2007. http://www.genetic-programming.com/.

[125] Olly Oechsle. Ecj2java, 2005. http://vase.essex.ac.uk/software/ecj2java/.

[126] Kumar Chellapilla. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1(3):209–216, 1997.

[127] Sean Luke and Lee Spector. A revised comparison of crossover and mutation in genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 208–213, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[128] T. Blickle and L. Thiele. A mathematical analysis of tournament selection, 1995.

[129] Tatsuya Motoki. Calculating the expected loss of diversity of selection schemes. *Evolutionary Computataion*, 10(4):397–422, 2002.

[130] Gearoid Murphy and Conor Ryan. A simple powerful constraint for genetic programming. *Genetic Programming, Lecture Notes in Computer Science*, 4971, 2008.

[131] Sara Silva and Jonas Almeida. Dynamic maximum tree depth. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1776–1787. Springer-Verlag, 2003.

[132] Thomas Loveard and Victor Ciesielski. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1070–1077. IEEE Press, 2001.

[133] Been-Chian Chien, Jui-Hsiang Yang, and Wen-Yang Lin. Generating effective classifiers with supervised learning of genetic programming. In *Data Warehousing and Knowledge Discovery*, pages 192–201, 2003.

[134] Gianluigi Folino, Clara Pizzuti, and G. Spezzano. Improving cooperative gp ensemble with clustering and pruning for pattern classification. In *GECCO 2006*, pages 791–798. ACM, 2006.

[135] Wei-yin Loh Tjen-sien Lim. An empirical comparison of decision trees and other classification methods. Technical report, University of Wisconsin Madison, 1997.

[136] R.A Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[137] P. McCullagh and John A. Nelder. *Generalized Linear Models, Second Edition*. Springer, 1989.

[138] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[139] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[140] W.H. Wolberg W.N. Street and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *International Symposium on Electronic Imaging: Science and Technology*, volume 1905, pages 861–870, 1993.

[141] F. C. Crow. Summed-area tables for texture mapping. *Journal of Computer Graphics*, 18(3):207–212, 1984.

[142] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd Edition*. Boston: Addison-Wesley, 1993.

[143] B.A Wandell. Analysis of the retinex theory of color vision. *Journal of the Optical Society of America*, 3:1651–1661, 1986.

[144] G. Buchsbaum. A spatial processor model for object colour perception. *J. Franklin Ins.*, 310(1):337–350, 1980.

[145] E.H. Land. Recent advances in retinex theory and some implications for cortical applications: Colour vision and the natural image. In *Proceedings of the National Academy of Science*, volume 80, pages 5163–5169, 1984.

[146] T. Gevers and A. W. M. Smeulders. Colour based object recognition. *Pattern Recognition*, 32:453–464, 1999.

[147] Fuhui Long Hanchuan Peng and Chris Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[148] Ron Kohavi and George John. Wrappers for feature subset selection. *Artificial Intelligence Journal, special issue on relevance*, 97(1):273–324, 1997.

[149] R. Fergus. Caltech leaves dataset, 2003.

[150] M. Roberts. The effectiveness of cost-based subtree caching mechanisms in typed genetic programming for image segmentation. In *Applications of Evolutionary Computation, Proceedings of EvoIASP 2003, Colume 2611 of LNCS*, pages 444–454, 2003.

[151] Mark E. Roberts and Ela Claridge. A multistage approach to cooperatively coevolving feature construction and object detection. In *Applications of Evolutionary Computing*, volume 3449 of *LNCS*. Springer Verlag, 2005.

[152] Thad Starner, Joshua Weaver, and Alex Pentland. Real-time american sign language recognition using desk and wearable computer based video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1371–1375, 1998.

[153] Y. LeCun, L. Bottou, Y. Bengio, , and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[154] S. K. Nayar S. A. Nene and H. Murase. Columbia object image library (coil-100). Technical Report CUCS-006-96, Columbia University, 1996.

[155] Visionsystems, 2009. http://www.visionsystem.com/.

[156] Cognex vision systems, 2009. www.cognex.com.

[157] M. Koppen and B. Nickolay. Design of image exploring agent using genetic programming. In *Proc. IIZUKA'96*, pages 549–552, 1996.

[158] Yang Zhang and Peter I. Rockett. Evolving optimal feature extraction using multi-objective genetic programming: a methodology and preliminary study on edge detection. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 1, pages 795–802. ACM Press, 2005.

[159] Cristopher T. M. Graae, Peter Nordin, and Mats Nordahl. Stereoscopic vision for a humanoid robot using genetic programming. In *Real-World Applications of Evolutionary Computing*, volume 1803 of *LNCS*. Springer-Verlag, 2000.

[160] Ankur Teredesai and Venu Govindaraju. Gp-based secondary classifiers. *Pattern Recognition*, 38(4):505 – 512, 2005.

[161] Fernando E. B. Otero, Monique M. S. Silvia, and Alex A. Freitas. Genetic programming for attribute construction in data mining. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, page 1270, 2002.