VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Genetic Programming for Evolving Computer Programs with Loop Structures for Classification Tasks

Fahmi Abdulhamid

Supervisors: Mengjie Zhang and Kourosh Neshatian

Submitted in partial fulfilment of the requirements for
ENGR 489.

Abstract

Genetic Programming (GP) is an evolutionary technique that evolves computer programs to solve a problem. GP is commonly used for classification tasks where an input must be classified into a single category. The use of standard binary operators has proven effective for classification, but recent research has found that loop structures can generate notably better results. It is still unclear how characteristics of loop structures affect different classification tasks. This research proposes five novel loop structures with different characteristics and analyses their effectiveness and understandability against both image and non-image classification tasks. Results find that the proposed loop structures can outperform GP with no loops and existing loop structures. Results are also simpler and more human understandable than GP with no loops and existing loop structures.

Acknowledgements

I would like to thank Mengjie Zhang and Kourosh Neshatian, my two supervisors, for their ongoing guidance and support throughout this research. I also thank Harith Al-Sahaf, my fellow student, for his support and collaboration.

I thank my family and friends for their moral support and enthusiasm, without which I would have not put the same vigour and determination into my work. Much appreciation also goes towards my two neighbours, Kerry and Andrew, for proof-reading my work.

Lastly, I greatly appreciate the effort and generosity of the UCI Machine Learning Repository [1] for their support of the Machine Learning community by supplying free access to quality datasets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We humans have a powerful perceptual system capable of routinely identifying and categorising what is seen based on known patterns unique to different kinds of objects. Our perceptual system is apt toward identifying minute differences in objects quickly and reliably which allow us to manually classify objects as needed for different situations. Jobs such as quality inspection, categorising objects, and identifying anomalies are frequent tasks required for production processes. Despite the capability of humans to perform these jobs adequately, long periods of performing repetitive tasks can lead to mistakes and quickly diminish the reliability of categorisation performance. Furthermore, compared to machines, people are expensive to employ and can be several times less efficient which can ultimately affect the success of a production process or business. Accordingly, given the amount of data to reliably classify, computer-based classification is desirable.

1.1 Motivation

The use of machine learning techniques such as Neural Networks and Genetic Programming (GP) have proven successful for automatic classification tasks for character recognition and geographic object detection [2]. Often an increase in both performance and reliability have been found for such automatic classification tasks by using learning techniques. GP is often employed due to the expressiveness and ease of analysis for classification tasks. GP is useful where the domain is not well understood or the relationships between variables is unclear [2] which is ideal for complex domains. GP is a form of evolutionary computation where a solution is evolved over a number of generations. The evolutionary process is utilised to build increasingly better solutions that are able to identify and correlate significant details in a way that benefits object classification.

Traditionally for image classification tasks, it is not uncommon to incorporate domain specific knowledge into GP systems. For instance, it is well understood that neighbouring pixel values in images are correlated in some way. Representing image data as two-dimensional structures can help GP to more effectively target neighbouring pixel values which can make the process of understanding the final solution easier. To make better use of pixel correlations, GP can also be used on extracted features that have been created beforehand. By identifying regions in images that are known to be significant within a domain and performing aggregate operations on those regions (such as mean or standard deviation), the GP process can be more successful. When applied to extracted features, GP builds solutions based on features that are already known to be significant rather than expending significant learning effort when attempting to find correlations in seemingly random data.

The process of GP learning correlations between data values can be time consuming due to the limited expressiveness of standard GP functions. The use of basic mathematical operators ($+$, $-$, $\times$, and $\div$) only target two data values at one time and perform very simple operations on the two selected data values, which may not produce any meaningful result in isolation. The incorporation of aggregate functions using loop structures has been found to increase the expressiveness of GP to produce more accurate and more simple solutions [3, 4, 5].

Loop structures are able to target related subsets of data and perform complex aggregate functions to provide meaningful results for classification tasks. For instance, one can imagine that it is difficult to differentiate a digit by looking at any particular pixel in isolation but the task becomes easy when looking at groups of pixels that represent notable regions of digits. With discovered improvements in classification accuracy by incorporating loops in GP, it is desirable to develop further loops by applying previous lessons learned to allow GP to evolve programs that are better than GP with no loops and GP with existing loop structures.

It is believed that by better understanding the impact that loops have on GP for classification tasks, users of GP systems may be able to employ a greater degree of expressiveness to achieve better classification results. Varying characteristics of different loop structures may be beneficial for different types of problems. Understanding these characteristics is important to ensure that the correct loop structures are used in a way that maximises their effectiveness. Further, loops are able to simplify learned solutions to become more human understandable. Simplified solutions are able to more effectively express which features are important for classification, which is helpful for analysis if the classification domain is not well understood. GP with loops can produce simpler solutions than GP without loops, but there is a need for improvement, particularly as problems become more complex. The goals for this research include developing new loop structures that are simpler, but more expressive, than GP with no loops and GP with existing loop structures.

1.2 Research Goals

The goal of this research is to design and develop new loop structures that are more capable of contributing to better classification accuracies for both binary image and non-image classification problems. The development of new loop structures is a computer science problem with regard the development and analysis of novel loop structures. It is also a software engineering problem with regard to the process of identifying problems and characteristics in current classification problems and designing a solution to mitigate those problems. By developing new loop structures and understanding their impact on GP, future applications of GP may benefit with a greater insight into the advantages and disadvantages of loop structures for different classification tasks. The specific goals of this research are listed below:

1. Establish a standard, baseline, GP system with no loops for comparison against GP with loops.
2. Develop new loop structures for *image* classification tasks.
   (a) Develop loops structures that perform better than GP without loops and an existing loop structure [3].
   (b) Understand how different characteristics of image data can be exploited with loops.
   (c) Investigate whether learned programs with loops are more human understandable than programs without loops and an existing loop structure.

3. Develop new loop structures for *non-image* classification tasks.
   (a) Produce GP systems with loops that outperform GP without loops.
   (b) Understand how loops can identify and exploit relationships between data values.
   (c) Produce learned programs that are simpler and more human understandable than GP with no loops.

The first item represents a 'standard' GP system that may be commonly used for classification tasks. The standard GP system is used for comparison purposes against GP with loop structures. GP without loops has been successfully used for classification tasks which makes it ideal for comparison purposes. The second item is to add new loop structures onto the standard GP system for image classification problems to increase classification performance. Neighbouring pixel values in images have some correlation that may be significant for classification. Loop structures acting on image data must be able to easily correlated neighbouring pixels. By using GP without loops as a baseline, the impact of the addition of loops can be measured. GP with no loops has two notable issues; unnecessary program bloat and the difficulty for users to understand how solutions work without considerable effort. Simpler solutions that more clearly target important features can reduce complexity to allow users to gain more insight into how images are classified. As part of this research, the effectiveness of the new loop structures to target important features in images as well as produce human understandable solutions will also be analysed.

The third item is to develop new loop structures for non-image classification problems. Unlike image classification problems, non-image data is not assumed to have any correlation between neighbouring data values. New loop structures must be able to derive correlations between arbitrary data values. It is also hypothesised that the same reduction in learned program complexity can be found for non-image classification problems as well.

1.3 Major Contributions

My contribution to the field of GP is the design of five new loop structures for image and non-image classification problems. I have designed the Evolvable Body Loop (EB-Loop), Bar-Loop, Seam-Loop, Step-Loop, and Sort-Loop loop structures. While each loop structure may be used for image classification tasks, all but the Bar-Loop, and Seam-Loop may be used for non-image classification tasks. Although loops have been applied to GP for classification problems in previous studies, the loops I have developed provide several notable benefits:

- **Accurate Classifications** The application of the new loop structures enables GP to achieve better classification accuracies than GP with no loops and GP with existing loop structures.
- **Simple Solutions** The new loop structures can produce solutions that are smaller and more human understandable than GP with no loops and GP with existing loop structures. Solutions that are easy to analyse allow users to gain an understanding of the significant characteristics of a problem which may be beneficial if the classification task is not well understood.

In particular, the users do not need to have knowledge of any prior research on the use of loops for non-image classification problems. In fact, the use of an appropriate loop structure is able to find correlations between values in non-image data to achieve classification results notably better than those generated from GP with no loops.

In addition to the above contributions, two works produced from this research are listed below:

- "Genetic Programming for Evolving Programs with Loop Structures for Classification Tasks," in *The 5th International Conference on Automation, Robotics and Applications (ICARA)*, 2011 — Accepted.
- "Image Recognition using Genetic Programming with Loop Structures," in *The 26th International Conference on Image and Vision Computing New Zealand (IVCNZ)*, 2011 — Submitted.

1.4 Report Organisation

A brief description of Machine Learning and GP followed by an overview of previous work relating to loop structures in GP is given in Chapter 2. In Chapter 3, the proposed loop structures and loop operators are introduced. Chapter 4 describes the classification problems used to analyse GP with loops along with details on experiment setup. GP systems are tested against the classification problems and results reported in Chapter 5. GP solutions are analysed in Chapter 6. Finally, the report is concluded and future work is outlined in Chapter 7.

Chapter 2

Background and Related Work

Machine Learning is the programming of machines to make inferences or learn from their environment. In this Chapter, a description of Machine Learning will be given, followed by an introduction into GP, and concluded by an overview of previous work into the application of loop structures in GP.

2.1 Machine Learning

Animals are able to make inferences and learn from their experience through life. Their behaviour is influenced by what they do and from what they learn. This learning and expressing of behaviour is known as intelligence [6]. Intelligence is naturally the domain of biologists and psychologists but significant advances in the representation of machine intelligence has been made by computer scientists and software engineers. It is often classified as Machine Learning which is one branch of the broad spectrum of Artificial Intelligence (AI).

Just as animals learn, it is desired for machines to learn as well. Aspects such as when a machine learns and when a machine is considered 'intelligent' are always changing. It is desirable for machines to learn in order to extract information within large bodies of data, make intelligent inferences on the world, and adapt, as animals do, to a changing environment automatically.

Machine learning presents difficult problems such as "how do machines learn?" and "how do machines store knowledge?". Machines learn by changing some state of their internal representation [6]. Which states change and how states change impact on how machines can learn. Different Machine Learning algorithms store information in different ways. The storage of information is not so important as how relationships between information can be extracted [7]. Only a small model of the world can be represented at any time, and making inference upon the data in the world based on previous observations is key to Machine Learning.

Numerous Machine Learning techniques exist such as Decision Trees, Naive Bayes, Neural Networks, and GP to name a few [8].

2.2 Classification Tasks

Classification tasks involve differentiating if an example is in one of several categories, or classes. Binary classification involves only two possible classes. Typically, a collection of known data, called the *training set*, is given to a Machine Learning system. The system then undergoes a learning process to learn how to differentiate between multiple classes.

Evolutionary techniques such as Genetic Algorithms (GA) and GP will cycle multiple times to learn a classifier capable of differentiating between classes. Once learning concludes, a *test set* of unseen data is given to the learned classifier. The classification of example in the test set are known to the user, but not to the classifier. The performance of a classifier to differentiate examples in the test set into the correct classes is an indicator of how well the classifier has generalised. An illustration of the process is given in Figure 2.1.

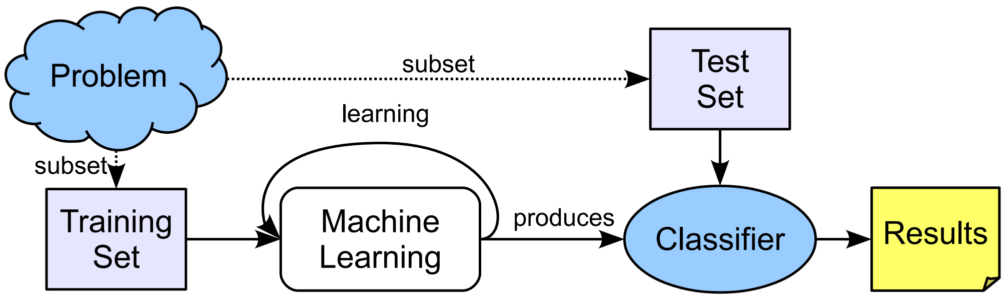


Figure 2.1: Overview of Machine Learning Process

Typically, if the training set contains a biased collection of examples or if inadequate settings were used to run the Machine Learning program, classification of the test set will be poor. While the learned classifier can preform poorly on the test set because of poor learning conditions on the training set, *overfitting* can occur on the training set if too much learning occurs. Overfitting is when the classifier targets minute features unique to elements in the training set that are not common in unseen data. In this case, the outcome is a good result for the training set and a poor result for the test set.

2.3 Genetic Programming and Classification

GP is an evolutionary computation machine learning paradigm. As a machine learning process, GP is given a training set of known (classified) data and a test set of unseen data. GP builds computer programs represented as tree structures to classify all instances in the training set before it is applied to unseen instances in the test set for validation. As an evolutionary computation process, GP learns how to build programs to solve a given problem by undergoing multiple iterations or *generations* of program construction until user defined stopping criteria are met (usually when a minimum error rate is achieved or the process runs for too long). Programs are represented as tree structures (similar to LISP expressions) built from function nodes (composite structures) and terminal nodes (leaf structures). Function nodes represent operations such as addition and subtraction while terminal nodes represent values or variables such as random numbers or input data values. Every node returns a value. Terminal nodes return their stored value and function nodes return a result based on input from their children, thereby creating an executable program. A positive or negative result returned from the root node determines the class for binary classification.

Figure 2.2 illustrates an overview of the GP process. First, a random population of programs, called *individuals*, are created (1). Every individual is then tested on the training set and a measure of performance such as accuracy or error rate is assigned (2). The measure of performance is calculated by a *Fitness Function*. A subset of individuals is then selected for breeding, with good individuals more likely to be accepted than poor individuals (3). Genetic operators such as Elitism, Crossover, and Mutation are applied on the selected individuals to evolve a new population of individuals (4, 1). Most likely, the new population will perform better than the previous population. The new population is then evaluated against

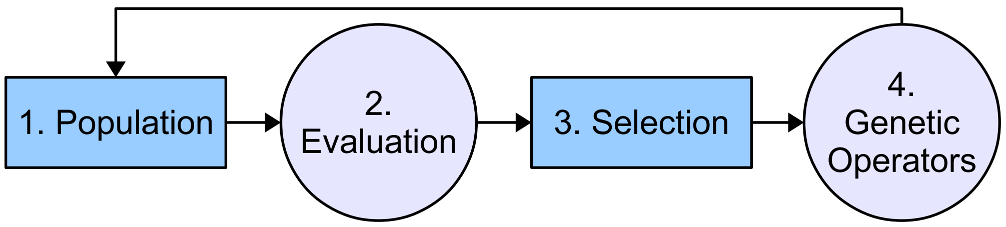


Figure 2.2: Overview of GP Evolution Process

the training set and the cycle continues until user defined stopping criteria have been met.

Elitism, Crossover, and Mutation are three common genetic operators used for breeding. The Elitism genetic operator simply copies an individual from the current population to the new population. Elitism preserves individuals to guarantee that good individuals are not lost which may have a chance of reducing the quality of new populations. Crossover takes two individuals, swaps some of their sub-trees and places the 'children' in the new population. Mutation takes a single individual, randomly changes parts of it and places the mutated individual in the new population. Crossover serves to improve individuals by swapping parts between two individuals. Mutation randomly changes a part of a single individual with a chance of improving it, and preserves the diversity of the population.

The addition of loop structures does not change the overall GP process. Loops are represented as function nodes that are subject to the GP process like any other basic mathematical operator.

2.4 Loops in Genetic Programming

The earliest work of applying loops to GP was to solve the Block-Staking and Summation problems [9]. The `(Do-Until work predicate)` loop was used as a means to provide iteration to solve the problem. The `Do-Until` loop took a body and a condition and, like the classical 'While' loop, would execute the loop until the condition was satisfied. The `Do-Until` loop was prone to infinite loops because it was not guaranteed to terminate unless the GP system enforced limited iteration. As a result the `Do-Until` loop may have been expensive to evaluate. Later work made use of Lambda abstractions to implement implicit recursion structures that were used to solve the Even-N-Parity problem [10]. The use of Lambda abstractions to generate compositions of structures (like $g(f(x))$) increased the expressiveness of learned programs to more effectively solve tasks. The use of implicit recursion allowed iteration to occur without the possibility of infinite loops. Infinite loops can increase the complexity of GP because measures must be taken to penalise inefficient programs that may contain structures of good programs. GP with Lambda abstractions observed a dramatic increase in the probability of success over GP with no iteration structures for the Even-N-Parity problem. The use of iteration through recursion not only improved results but allowed GP to solve more difficult problems.

For another iteration-intensive problem, the Lawn-Mower problem, it was found that GP with loops could find solutions for larger 'lawns' than GP without loops given the same constraints on program size [11]. It was also found that GP with loops produced simpler solutions in less time than GP without loops. Although the results were positive, the `(Loop N)` loop structure was used where `N` is set to a domain specific value. The value of `N` is the same for all loops in the GP process and is set beforehand by the user. If `N` was set to an inadequate value because the domain was not well understood, the loop structure would not contribute effectively to learned solutions.

Two of Koza's students have also contributed loops to GP for solving the Factoring and Greatest Common Factor (GCF) mathematical problems. The `Do-While` loop was applied to GP for factoring numbers [12]. Factoring numbers requires iteration and GP could successfully find a general solution to the problem. The GCF problem was also attempted using loops in GP [13]. Learned solutions consisted of two separate trees. The initialisation tree performed set-up tasks while the loop tree performed the iteration. No loop function nodes were used, instead the GP system itself would iterate the loop tree until a stopping condition was satisfied. GP with iteration could solve the GCF problem between any two numbers and still produce a Human understandable solution. Notwithstanding the application of two trees, one for setup and the other for iteration, is highly problem dependent and may not be effective for other tasks. The GP structure used only permitted one loop where some problems may require multiple loops to solve. Both students used memory slots in their GP systems in order to alias the same values. This report focuses on loops for classification and does not include the use of memory slots.

Two new loops were developed and tested against the Sante Fe Ant problem and to a number sorting problem (of length 7) [14]: The `(For-Loop1 numIterations body)` and `(For-Loop2 start end body)` loops were developed. For both problems, it was found that GP with no loops performs better than GP with loops for easy problems but worse for difficult problems. GP with no loops performs better for simple problems because of the smaller search space and the restriction of GP with no loops to solve problems smaller than a function of the GP size. GP with loops had a notably higher probability of finding perfect solutions than GP without loops. The solutions produced by GP with loops were also smaller that those generated by GP without loops and comparable to existing Human-generated solutions. A disadvantage of loops was found when a large increase in evolution time was observed. The two loop structures were later re-tested on the Visit-Every-Square problem and a modified ant problem [15]. Like the previous paper, GP with loops had a higher probability of success than GP without loops. It was also found that time to evolve programs with loops followed a double-exponential curve in relation to the maximum iteration limit which is a cost that must be taken into consideration when seeking better results. Although expensive, GP with loops could evolve faster than GP with no loops with an iteration limit of less than 200. A simplified version of the loop structure, `(For-Loop start end method)`, where `method` is one summation or subtraction, was used to reduce the search space because loop bodies no longer need to be evolved [5]. The loop structure was applied to a binary image classification problem between squares and circles. GP with loops performed better than GP with no loops for the simple problem and even better when the images were shifted off-centre. Of concern, the images used were artificially produced and contain only black and white pixels (no grey pixels). The results may not be representative of real world tasks where images may be of naturally occurring objects with grey scale pixels (0-255).

A count-controlled loop (`while start end body`) and an event-controlled loop (`while condition body`) were also tested against the artificial ant problem but also to the factorial problem (n!) [16]. Only GP with loops could solve the factorial problem and solutions with loops were more comprehensible than those without loops. GP with loops also found more solutions to the ant problem than GP without loops and evolved solutions solved the problem in fewer steps. The ability of GP with loops to identify a commonly recurring structure and nest that structure into a loop ensures that sub-trees do not have to be duplicated. Solutions can be found in fewer iterations. Three loop structures were used in conjunction to solve novice procedural programming problems such as "find the maximum number in a list" [17]. The application of loops was successful at solving such problems. To increase evolution speed, programs with loop structures were penalised. Such a penalty prompted the use of an Iterative Structure-Based Algorithm (ISBA) to escape local optima due to the fitness bias. Results using ISBA were better than those without.

More recently, Li's Ph.D thesis was published which contained the analysis of several loop structures against several problem solving tasks [3]. Of particular interest to this work, Li developed loop structures to solve the square-circle binary image classification problem. The application of the `(for-loop-1d start end method)` and `(for-loop-2d-rect start end method)` loops generated solutions that were more general and more simple than those generated by GP without loops. Both loop structures have been incorporated in the experiments discussed later for comparison purposes. For another task, akin to classification, 'For' loop structures were evaluated for learning a repeating binary pattern such as "1001001..." [18, 19]. Unlike GP without loops, GP with loops was able to learn a pattern to any arbitrary size (a perfect solution) while GP without loops was constrained by program size.

A comparison of restricted (nesting disallowed) and unrestricted (nesting allowed) loop structures in GP for binary image classification tasks demonstrated the increased utility of loops [4]. For a character recognition problem, GP with loops did only slightly better than GP without loops. But when images were shifted off-centre, GP with loops notably outperformed GP without loops. GP with loops also did better for the texture recognition problem. Comparing restricted and unrestricted loops, it was found that restricted loops perform better for simple tasks but unrestricted loops perform better for more difficult tasks. Again, an increase in evolution time was found for GP with loops over GP without loops.

The comparison of multiple loop structures against multiple mathematical problems [20] proved to be a good way of identifying which characteristics of loops were better for different types of problems. Such a methodology is employed for this paper. Results found that loops with implicit stopping criteria were more frequently used by GP than loops with explicit stopping criteria. It was also found that GP with loops is not good at solving mathematical problems with non-approximative (exact) solutions. Although 'While' loop structures had a relatively unstable usage through early GP evolution, the Memory Loop and Conditional Assignment loops proved stable for mathematical problems. A stable result indicates that the design of novel loop structures can outperform classical loop structures used in programming languages (such as the 'For' loop and 'While' loop) despite the fact that the loops used were domain specific for mathematical problems.

2.5 The Case for New Loop Structures

From Section 2.4 it is clear that the use of loops in GP has promising benefits for game-playing, pattern learning, and image classification tasks. Most of the work on applying loops to GP have been oriented towards generating iterative instruction sequences for game playing. More work needs to be carried out for classification tasks. The circle-square, character recognition, and texture recognition image classification problems only demonstrate the use of loops for a narrow range of classification problems. The utility of loops is still unclear with regard to how loops can benefit different image classification tasks.

I propose five new loop structures which are evaluated against six image datasets in order to gain a more in depth understanding of how the different characteristics of loop structures impact performance on different classification tasks. Further, I evaluate the utility of three of the five proposed loop structures for non-image classification tasks. To my knowledge, no previous research has evaluated GP with loop structures for non-image classification tasks. Non-image classification tasks are used when GP is applied to pre-extracted features or when the classification problem does not involve simple visually representable objects (such as particle identification). The five proposed loop structures will be presented and their performance analysed in the following chapters.

Chapter 3

Design and Development of New Loops

Exploiting various characteristics of different classification problems is expected to allow loop structures to both perform well and produce simple results. Suitable loop structures can allow significant features to be more easily detected to support classification. This chapter discusses five proposed loop structures developed for classification tasks. Each loop structure was designed with a different set of characteristics based on exploiting various aspects of classification problems. Three loop operators are also discussed which have been developed to support the loop structures.

3.1 New Loop Structures

Five new loop structures are proposed: The EB-Loop, Bar-Loop, Seam-loop, Step-Loop, and Sort-Loop for binary classification problems. To simplify the learning process, all but the EB-Loop use predefined loop operators. The available loop operators are Summation, Standard Deviation, and Sum-Mode. The new loop structures and the applied loop operators are discussed below.

3.1.1 Evolvable Body Loop (EB-Loop)

The Evolvable Body Loop (henceforth, EB-Loop) is a loop structure designed for 2D image data but is also applicable to 1D non-image data (provided that the data is represented as a 2D collection with one column). Where a domain is not well understood, GP with loops is useful for automatically targeting regions of significance to provide a better understanding of data to aid in classification. Just as important as knowing which regions are significant, it is necessary to understand which operators are best suited on identified regions. Basic operations such as summation are powerful, but there may exist less obvious operations that are better for specific tasks. The EB-Loop is designed to automatically learn operations that better represent the loop region acted upon. Operations that provide more expressiveness for loop

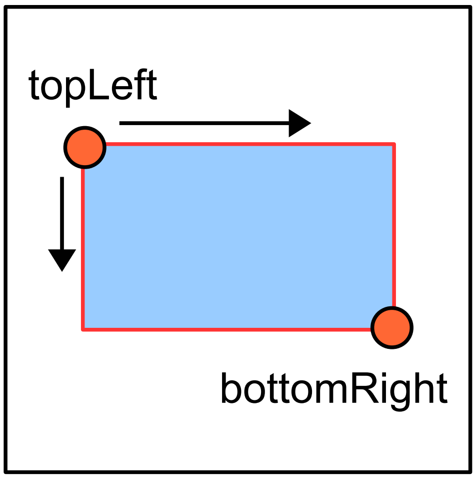


Figure 3.1: EB-Loop Method

regions may contribute to better classification accuracies. The EB-Loop structure is

```
(EB-Loop topLeft bottomRight loopBody)
```

which is illustrated in Figure 3.1. The first two arguments represent the (row, column) coordinates of the rectangular region to iterate over. The positions may be swapped to ensure that `topLeft` and `bottomRight` represent their respective corners. The `loopBody` argument represents the subtree that will compute the operation over the loop region. To allow operations to be constructed, two special loop terminals are used which only exist within the `loopBody` subtree. The first, `loopCurrentValue`, represents the current data value being iterated over and the second, `loopResult`, represents the accumulated result. The EB-Loop procedure is described below:

1. Populate the `loopCurrentValue` terminal with the current data value.
2. Evaluate the `loopBody` subtree.
3. Assign `loopResult` to the result of the subtree.
4. Advance the iteration index and repeat if not finished.

The incorporation of small predefined subtrees called 'Snippets' is used which contain both special loop terminals to promote valid loop structures and spread the two special loop terminals when genetic operators are applied.

3.1.2 Bar-Loop

The Bar-Loop is a loop structure designed for 2D image data only. It is not applicable to 1D non-image data. Th EB-Loop is free to evolve any rectangle loop regions within the bounds of the data which allows it to identify small and precise regions, but might serve to increase the search space. A larger search space decreases the probability of finding good solutions because more paths may be poor. GP must exhaust more generations before a good solution can be found which increases evolution time and decreases solution accuracy if a good solution is not found. The Bar-Loop is constrained to evolving rectangles that *must* touch two opposing sides of an image. A shape restriction reduces the search space to improving learning performance and may enforce loop regions to iterate both inside and outside objects for better classification accuracy. Iterating inside objects can extract features from object content while iterating outside objects can extract features based on object shape and size. The Bar-Loop structure is

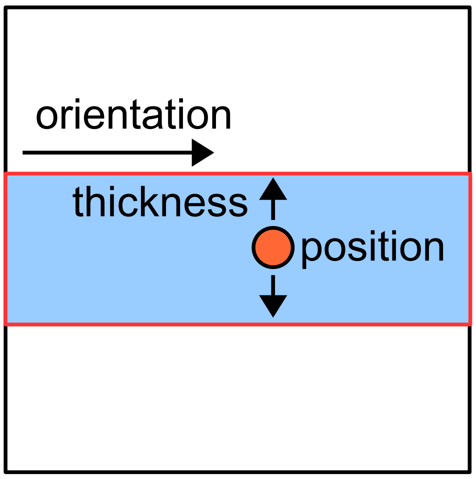


Figure 3.2: Bar-Loop Method

```
(Bar-Loop position thickness orientation loopMethod).
```

which is illustrated in Figure 3.2. The `position` argument determines a middle (row, column) coordinate that the loop region must iterate over while `thickness` determines how thick the loop region is. The `orientation` argument determines wheather the loop region is horizontal or vertical and the `loopMethod` terminal is a predefined operation. Only the EB-Loop evolves a loop operation.

3.1.3 Seam-Loop

The Seam-Loop is designed for 2D image data only and is not applicable to 1D non-image data. Large search spaces can occur as a result of the flexibility of loop structures. Loop structures may have too many 'options' that increase the possible combinations of options available to learn. Restricting loop flexibility can reduce the search space at the cost of a less expressive loop structure. The Seam-Loop uses a method inspired by Seam Carving [21] to automatically find image seams that may be of significance. Seam Carving identifies seams of low energy for removal while the Seam-Loop identifies seams of high energy that may be significant to particular object classes. GP only needs to specify a start position and the Seam-Loop automatically identifies important pixels. For performance reasons, the Seam-Loop only finds vertical seams. Further, a simplified Seam Carving algorithm is used that performs a greedy hill-climbing traversal with no preprocessing steps. The Seam-Loop structure is

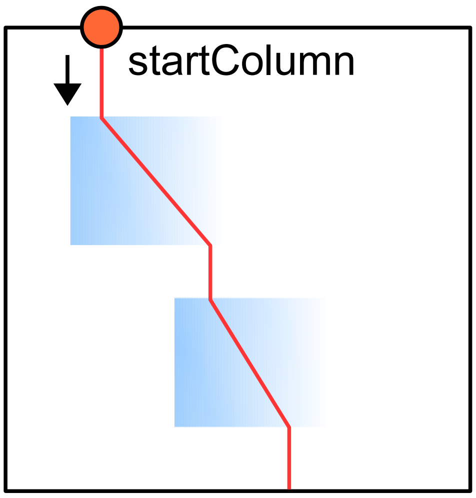


Figure 3.3: Seam-Loop Method

```
(Seam-Loop startColumn loopMethod)
```

which is illustrated in Figure 3.3. The argument `startColumn` represents a column number such that the seam starts at (row, col) position (0, `startColumn`) and traverses downwards. For every position visited, the left, middle, and right positions directly below are inspected. The position with the highest intensity (pixel value) is chosen as the next position to visit (note how the Seam slants to the light regions of the boxes in Figure 3.3). If all values are equal, the center position is chosen by default.

3.1.4 Step-Loop

The Step-Loop is primarily designed for 1D non-image data but it is applicable to 2D image data as well. For 2D image data, the Step-Loop operates on a 1D representation of the image where each row is appended onto the first row. It is well understood that neighbouring pixels in images are correlated because important features typically span multiple pixels in size. Loop structures can exploit correlations between neighbouring pixels by performing aggregate functions on nearby pixels. Correlation between neighbouring values is not a characteristic of non-image data. There may be no correlation between data values in tabular data (a logical database table/relation representation for example). The previous rectangular loop structures may not be able to find good correlations.

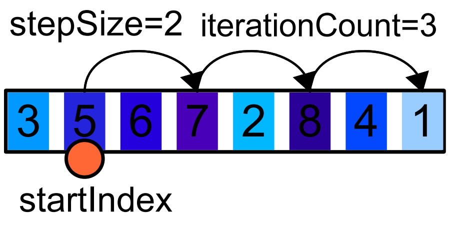


Figure 3.4: Step-Loop Method

The Step-Loop is designed to 'step' over data values in order to iterate over distant data values. By targeting distant data values, the Step-Loop is more likely to identify correlations in a non-image dataset. GP is able to learn good step distances to skip over insignificant data values. The Step-Loop structure is

```
(Step-Loop startIndex stepSize iterationCount loopMethod)
```

which is illustrated in Figure3.4. Iteration begins at `startIndex` and increments by `stepSize` for each iteration. Iteration stops after `iterationCount` cycles. The `loopMethod` is evaluated for each data value iterated over. Algorithm 3.1.1 displays the procedure of the Step-Loop as a 'For' loop.

Algorithm 3.1.1: STEP-LOOP(*start, step, count, data*)

```
$index \leftarrow start$
$result \leftarrow 0$
for $i \leftarrow 0$ to $count - 1$
    do { $index \leftarrow index \,\%\, data.length$
         $result \leftarrow result + data[index]$
         $index \leftarrow index + step$
return $(result)$
```

The Step-Loop also 'wraps' around the bounds of the collection to avoid array bounds errors and increase the expressiveness of the loop structure for 2D image data. Without wrapping, only one row or column can be targeted in image data. For 2D image data, the Step-Loop is able to target distant features in images. For example, the Step-Loop may be able to better identify if two features change together (like eyes) which can be useful for identifying relationships, such as symmetry, between features in images.

3.1.5 Sort-Loop

Like the Step-Loop, the Sort-Loop is designed for 1D non-image data but is applicable to 2D image data represented in a 1D structure. Because non-image data is not considered to have correlations between neighbouring data values, it is important to design a means to identify correlations between data values regardless of location. The Step-Loop identifies correlations by stepping over independent data values to identify correlated data values. The Sort-Loop sorts the entire collection into ascending order. Small values are low-indexed, large values are high-indexed, and thus values of similar magnitude become neighbours. It is assumed that values of similar magnitude may be related in some way. Performing a loop function over a sub-sequence of the data may yield good results because neighbouring data values are correlated and can produce significant results for classification. The Sort-Loop structure is

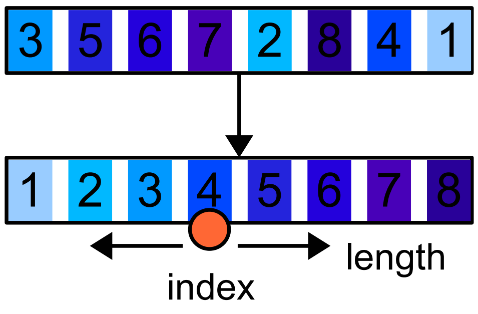


Figure 3.5: Sort-Loop Method

```
(Sort-Loop index length loopMethod)
```

which is illustrated in Figure 3.5. The `index` argument represents the centre of the sub-sequence to iterate over, `length` represents the number of items be iterated over, and the `loopMethod` argument is one of the pre-determined loop aggregate operators. Unlike the Step-Loop, no wrapping occurs. Instead, the loop region is restricted to within the bounds of the collection. Restriction is performed because values at opposite ends of a record are not similar in magnitude and may not be correlated.

3.2 Loop Operators

Loop operators are predefined aggregate functions that are applied to a collection of data. The `loopMethod` terminal represents one of the three loop operators. Loop operators produce aggregate results over all values covered by a loop region. Predefined loop operators are used to reduce the GP search space. Loop operators can be complex and difficult to learn. By defining loop operators, GP only has to select one to use. Supplied loop operators are expected to be correct and expressive for classification tasks, increasing the probability of good solutions learned by GP with loops.

The three predefined loop operators are Summation, Standard Deviation, and Sum-Mode. The three loop operators evaluate on numerical data (which is the case for all ten problems). Each is described below.

3.2.1 Summation

The Summation loop operator simply adds all data values together. The Summation operator is well understood to produce good results for image classification problems [3]. The Summation operator can identify whether the magnitude of values is large or small which can be useful when targeting areas that are more intense for one class and less intense for another. The formula used for Summation is displayed in Equation 3.1, where n is the number of data values.

$$Summation = \sum_{i=0}^{n} data(i) \tag{3.1}$$

3.2.2 Standard Deviation

The Standard Deviation loop operator gives the variance between data values that have been iterated over. Standard deviation is good at identifying how 'smooth' or 'rough' the values within a given region are. A smooth region for one class may be rough for another, providing a means for differentiation. The formula used for standard deviation is displayed in Equation 3.2. The variable n is the number of data values and the variable μ is the mean of the collection.

$$StandardDeviation = \sqrt{\frac{\sum_{i=0}^{n}(data(i) - \mu)^2}{n-1}} \tag{3.2}$$

3.2.3 Sum-Mode

The Sum-Mode operator is a combination of Mode and Summation. The Mode operator gives the numbers that are found with the highest frequency. For example, the Mode of $(7,2,7,2,3,1)$ is $(7,2)$ because both seven and two occur with the highest frequency of two. If all values were unique, then the Mode would consist of the entire collection of numbers. The Mode operator is useful for non-image data because it can detect frequency relationships between data values. The GP system used in this research used mono-valued types, not multi-values types such as sets and vectors. It is possible to return just one of the Mode values, but which one may be specific to different problems. Further, it would not be clear which Mode is returned when analysing learned solutions, increasing the complexity of learned classifiers.

Instead, in this project, the sum of all Mode values is returned. By returning the sum, the operator becomes more expressive because it is giving more information about frequency values. The Sum-Mode is also easy to understand when analysing learned solutions, which makes learned solutions more Human understandable. A positive consequence of the Sum-Mode operator is that small values will be returned if few Mode values exist and large values will be returned of many Mode values exist (all values are unique) which can prove beneficial when identifying frequency correlations between data values. Algorithm 3.2.1 displays the procedure of evaluating the Sum-Mode operator.

Algorithm 3.2.1: GETSUMMODE(*data*)

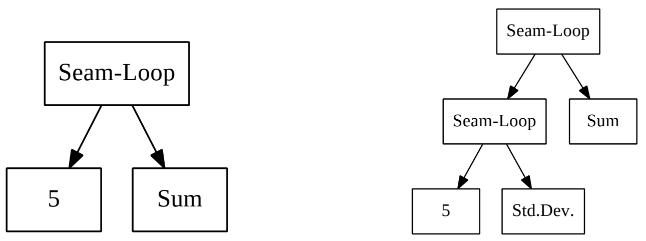
```
$modeSet \leftarrow getModes(data)$
$sumMode \leftarrow 0$
**for each** $value \in modeSet$
  **do** $sumMode \leftarrow sumMode + value$
**return** $(sumMode)$
```

3.3 Restricted and Unrestricted Loops

Programming languages typically contain 'For' loops and 'While' loops. These loops contain multiple statements in their body which are repeated until some stopping criteria are met. As well as simple statements such as arithmetic, assignment, and message calling, loops can contain other loops in their body. Loop nesting is expressive when working with multi-dimensional data or when the parent loop relies upon an aggregate operation in a child loop. Despite the expressiveness of nested loops, time complexity increases as the number of nested loops becomes larger. One loop incurs $O(n)$ time complexity, two loops $O(n^2)$, and etc. Such an increase in time may decrease procedure performance.

Loop structures in GP can be categorised into two broad categories, *restricted* and *unrestricted*. Restricted loops are not allowed to house nested loops. Unrestricted loops are allowed to house nested loops to evaluate at least one argument. As is the case with programming languages, restricted loops in GP are notably faster than unrestricted loops but do not perform as well for difficult tasks [4].

The EB-Loop is the only restricted loop proposed. The EB-Loop is restricted due to the increased difficulty of learning valid loop bodies. Making the EB-Loop unrestricted would further increase the difficulty of learning valid loop bodies, which would negatively impact on classification performance. The remaining loop structures are all unrestricted in order to maximise classification accuracy. Below, in Figure 3.6, are two examples of the Seam-Loop, one restricted (Figure 3.6a) and one unrestricted (Figure 3.6b).



(a) Restricted Seam-Loop Program

(b) Unrestricted Seam-Loop Program

Figure 3.6: Restricted and Unrestricted Seam-Loop Programs

Chapter 4

Experimental Configuration

In order to gain an understanding of how different characteristics of loop structures impact classification performance for different types of image problems, several classification problems have been selected. Problems are a mix of image and non-image classification problems which represent a wide range of different characteristics. The selected problems are described below followed by a description of experiment setup to analyse performance on the problems.

4.1 Classification Problems

To gain a good insight into the affect of loop structures on classification tasks, ten binary classification problems were tested. Each problem contains of a unique set of characteristics that will be useful when evaluating and comparing the effectiveness of loop structures. Classification tasks include classification of naturally occurring patterns, post-processed patterns, artificially derived patterns, measurement, and game playing. The large variety of tasks is ideal for comparison and contains more problems than prior experiments with loops in GP in the literature.

Of the ten problems, six are image classification problems and four are non-image classification problems. A description of the classification problems and essential modifications are given below.

4.1.1 Image Classification Problems

Six image datasets were used: Cell, MNIST, Face, Pedestrian, Box-Vert, and Box-Horz. Sample images are displayed in Figure 4.1.

Cell The Microscopical Cell Image Database (Serous cytology) [22, 23] represents a collection of eighteen different types of Cells. Experiments perform a binary classification of Lymphocytes non activés and Mésothéliales. The original images are in colour and have been converted to grey scale images to simplify classification. In grey scale, each pixel is a natural number between 0 and 255 (inclusive) which allows pixels to be represented as single numbers. Images are also cropped to only include the cell by removing data not included in the nuclear segmentation of the cell. The images are cropped to avoid GP classifying cells based on artefacts outside each cell which ensures that GP systems are actually distinguishing between cells. The original cell images were of different dimensions so all images were scaled to 25×25 pixels to ensure consistency. The cells have a relatively bland internal content which means that the shape/structure of the cells play a large role in classification.

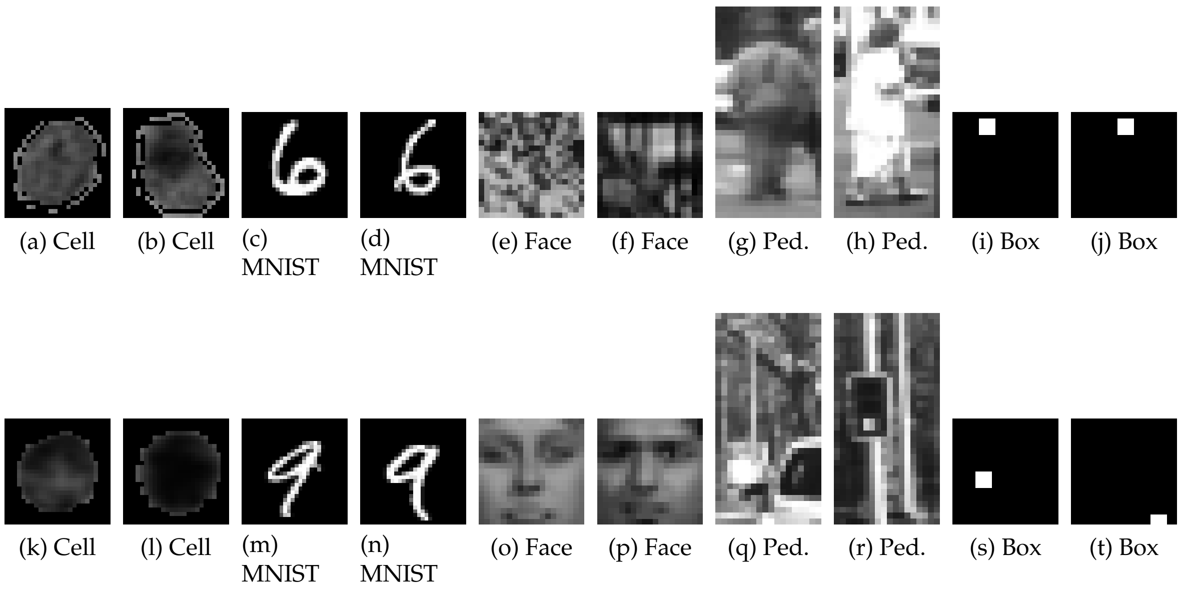


Figure 4.1: Examples of images from the image datasets. The top row are class 1 images and the bottom row are class 2 images.

MNIST The MNIST Database of Handwritten Digits [24] is a collection of images representing hand written digits from 0 to 9 (inclusive). All images are 28×28 pixels. Experiments perform a binary classification between the "6" and "9" digits. All images are gray scale and digits are centred on each image. Because the actual digits are filled with a solid colour, GP systems must classify digits based on shape alone.

Face The Face dataset [25] contains a collection of images which are either a face or an unknown object. Faces occupy the entire space of the image and hair is not displayed (though facial hair is). Experiments perform a binary classification between "Face" and "Not Face" images. All images are in grey scale and 19×19 pixels. Faces have well known significant regions that can be used for differentiation: Eyes, nose, and mouth. Because the border of the face is not clearly displayed in images, GP systems must perform classification based on the content of the face, not the shape of the head.

Pedestrian The Pedestrian dataset [26] contains images with pedestrians and images without pedestrians. There are three main difficulties with classifying pedestrians. First, pedestrians are not guaranteed to be in the centre of images or occupy the same area, which means that GP systems must evolve general solutions. Without general solutions, GP will suffer from overfitting on the training data. The second is the presence of background objects. Objects in the background are not consistent because pictures of pedestrians are taken at different crossings, at different times, and with different camera angles. Background objects may lead to classifiers not correctly classifying images. The third is clothing. Different people wear different clothes which can hide or augment Human features. All images are 18×36 pixels.

Box The Box dataset is a new dataset created for this research. It contains images of a white 5×5 pixel box on a black 32×32 pixel background. A colour depth of two is used (0 for black and 1 for white). The dataset is 100% computer generated. The algorithm for constructing Boxes is provided in Appendix A.1. Each image contains a box at a certain position. The task is to classify on which side the Box is on. There are two variations, Box-Vert for top-bottom classification and Box-Horz for left-right classification. The purpose of the Box dataset is to analyse the ability of a machine learning method to identify changes in a particular axis. Thye Box dataset also serves to analyse how accurate machine learning methods are when identifying small objects in a large image.

4.1.2 Non-Image Classification Problems

Forest The Forest Cover Type dataset [27] is used to predict forest cover type from cartographic variables only, not remotely sensed data. The classification task used in experiments aims to classify between Cottonwood/Willow and Douglas-fir. Each record contains 53 attributes and attribute values are represented as integers.

Spam The Spambase dataset [1] is used to classify personal emails as either Spam or Not-Spam. Each record contains 57 attributes and attribute values are represented as floating point numbers.

Poker The Poker Hand dataset [1] contains combinations of five-card Poker hands to classify Poker hands from empty hands to Royal Flush. Experiments perform binary classification of Poker hands between empty hands and one pair. Unlike the other non-image datasets, the values used to classify Poker hands do not stay in one place. For example, a pair may be found as the first two cards or the third and fifth cards which can make classification difficult due to the unpredictability of the order of the cards. Each record contains 10 attributes (for number and suite) and attribute values are represented as integers. The format of the Poker Hand dataset is described in Table 4.1.

Table 4.1: Format of the Poker Hand Dataset

Index Position	0	1	2	3	4	5	6	7	8	9
Attribute (s = suite, c = card)	s1	c1	s2	c2	s3	c3	s3	c4	s5	c5
Empty hand	1	11	3	9	3	1	3	5	1	6
Empty hand	1	4	3	9	4	8	3	7	2	6
Single pair	3	8	1	7	3	7	2	1	2	9
Single pair	2	8	1	8	3	3	2	13	3	7

MiniBooNE The MiniBooNE Particle Identification dataset [1] is used to classify between electron neutrinos (signal) and muon neutrinos (background). Each record consists of 50 real-valued numbers.

4.2 Comparison GP Systems

In order to derive a relative measure of performance for the new loop structures, GP with no loops was implemented as a baseline. The Terminal and Function set used for GP with no loops is displayed in Table 4.2. It is equally important to compare the new loop structures against existing loop structures to gain a better understanding of how the characteristics of different loop structures affect classification performance. Two of Li's loop structures were also implemented and evaluated in experiments; the `(for-loop-1d start end method)` was implemented for 1D non-image data and the `(for-loop-2d-rect start end method)` was implemented for 2D image data [3, Ch. 4].

Table 4.2: Terminals and Functions for GP with No Loops

Node	Type	Description
Drand	Terminal	Generates a double value between 0.0-100.0 denoted by 'drand-x'.
Pixel-Value	Terminal	Generates a random position within the bounds of the data, denoted by 'pixel-value-[r,c]'. It returns the value at position (row, col) and the value is cast to a double.
Addition	Function	Takes two double values and returns the sum.
Subtraction	Function	Takes two double values, subtracts the second from the first and returns the result.

All GP systems with loops use a superset of the terminals and functions used for GP with no loops. The base set of terminals and functions are derived from Li's work in order to facilitate an accurate comparison [3, Ch. 4]. Table 4.3 displays the additional terminals and functions used by each loop structure. The only notable exception is the use of the three proposed loop operations. Li's GP system used Addition and Subtraction only. In this research the three proposed loop methods are substituted into Li's loops [3, Ch. 4]. The substitution of loop operators ensures that results are comparable so that an accurate analysis and discussion can be made.

Table 4.3: Extra Terminals and Functions for GP with Loops

Applicability	Node	Type
EB-Loop	EB-Loop	Function
	Loop Current Value	Terminal
	Loop Result	Terminal
Bar-Loop	Bar-Loop	Function
	Vertical Orientation	Terminal
	Horizontal Orientation	Terminal
Seam-Loop	Seam-Loop	Function
Step-Loop	Step-Loop	Function
Sort-Loop	Sort-Loop	Function
Li's Loops	for-loop-1d	Function
	for-loop-2d-rect	Function
All Loops	Summation	Terminal
	Standard Deviation	Terminal
	Sum-Mode	Terminal

4.3 Experiment Configuration

Experiment configuration is displayed in detail in Table 4.4. The GP system configuration is designed to be similar to Li's work in order to facilitate an accurate comparison of GP systems. GP deployment configuration is designed to produce accurate aggregate results which are not biased in terms of dataset selection, or GP program construction.

Each GP system was evaluated against every applicable classification problem. Exactly 100 runs were evaluated for each combination of GP system and problem in order to produce accurate aggregate results. All datasets contain 1000 examples with a perfect 50%/50% split

Table 4.4: GP Experiment Variables

Variable	Value
Experiment runs	100
Generations	50
Population size	1024
Crossover (%)	70
Mutation (%)	28
Elitism (%)	2
Termination	100% accuracy or max generations reached
Training Set size	500 (250 class 1 and 250 class 2)
Test Set size	500 (250 class 1 and 250 class 2)
Tree Depth	Min: 1 Max: 8

between the training and test sets. The datasets were fixed for every experiment run to ensure consistency for comparison. The only variablity between runs was the random seed used for terminals that require the creation of random numbers and probabilitic selection and augmentation of individuals for breeding.

The fitness function used was classification error rate. For each unsuccessful image classification of a GP candidate program, the program is penalised by one point. Program accuracy is based on a single threshold. For any image to be classified, if the result is < 0 then a "class 1" classification is produced and if the result is ≥ 0 then a "class 2" classification is produced. Program fitness directly corresponds to classification accuracy. A fitness of zero indicates perfect classification on the training or test set. The GP learning process terminates when a perfect solution is found for the training set or fifty generations have been performed with no perfect solution. When the learning process terminates, the best solution (determined by the lowest fitness/highest accuracy) is evaluated on the test set. The fitness of the test set is a measure of how well a learned classifier has generalised on unseen data.

All GP systems were implemented with Java v1.6 using the ECJ v20 library [28]. ECJ supports *Strongly Typed GP (STGP)* which is a requirement for maintaining the integrity of learned programs. Not all nodes return the same data type, STGP ensures that programs are constructed correctly so that no type errors occur. Due to the large number of experiment runs and long GP evolution time, all experiments were deployed to the ECS Grid which is controlled by the Sun Grid Engine (SGE).

Although the Grid saves time by running experiments in parallel, there are consequences that must be noted. Machines in the ECS Grid are also student workstations; evaluation time between experiments may not be consistent due to the inconsistent workload each machine experiences. Different workloads may increase the variance of evolution time between runs. Network load may affect file transfer times which can also affect GP execution time because multiple files must be loaded. Network load has been mitigated in the experiments by copying all necessary files to target machines before GP execution starts. The hardware specification of the average computer in the ECS Grid is displayed in Table 4.5.

Table 4.5: Grid Typical Hardware for GP Experiments

Component	Value
OS	GNU/Linux
CPU	i686 Intel® Core^{TM} 2 Duo CPU E8400 @ 3.00GHz
RAM	3GB
JVM	1.6.0_03-p4

Chapter 5

Evaluation of Results

All five proposed loop structures as well as GP with no loops, GP with Li's 2D loop, and GP with Li's 1D loop have been evaluated against all applicable problems. Results are discussed in light of the different characteristics between loops and problems, from which comparisons are made. Different loop structures are either more powerful or inhibited by the characteristics of different problems. Analysis is made to understand which loops are useful under specific conditions in order to help users of GP systems to create more expressive function sets to solve problems displaying certain properties.

5.1 Image Classification

All GP systems were evaluated against all six image problems. Li's 2D loop was used for 2D image data. Results were aggregated from 100 independent runs of each experiment. Between all image problems, it was evident that GP with loops was able to achieve better classification accuracy on unseen data than GP without loops. Results for the Cell, MNIST, and Face problems are presented in Table 5.1 and results for the Pedestrian, Box-Horz, and Box-Vert problems are presented in Table 5.2. Training and test set mean accuracies from GP with loops are marked with $\uparrow$ or $\downarrow$ if statistically better or worse than GP without loops and are marked with $\Uparrow$ or $\Downarrow$ if statistically better or worse than GP with Li's loops. A paired two-tail t test was performed [29, 30].

5.1.1 Results for Images: Part 1

Results for the Cell, MNIST, and Face problems are displayed in Table 5.1. Regarding the Cell problem, Li's loop structure produced better results than GP with no loops for both the training and test sets. Li's loop also generalised better on unseen data by producing a smaller difference between training set and test set mean accuracy. Better generalisation is a sign of a more reliable classifier that has identified better features for classification. The Seam-Loop produced the highest training and test set accuracies for cell images. The restriction on search space by applying a loop structure to automatically find good seams had enabled the Seam-Loop to find the best solution within the same learning period. The Bar-Loop also performed well, producing results better than GP without loops and better than Li's loop, but not as good as the Seam-Loop. Again, the reduction on search space, this time by enforcing bar shape, had enabled better results for both training and test sets. The Bar-Loop also produced the best overall result for the cell problem. The EB-Loop produced results similar to GP without loops. Poorer results were achieved because most evolved loops were invalid and could not contribute effectively to classification. The Step-Loop performed slightly better than GP without loops. The ability to target distant features may have contributed to increased program expressiveness, but the lack of being able to target localised regions may have limited classification performance. The Sort-Loop achieved the worst results on the cell problem because it sorted pixel data out of order, removing significant correlations between neighbouring pixel values. Poor results by the Sort-loop were not surprising as the Sort-Loop was not designed for image data (although it can be applied).

For the MNIST problem, Li's loop, the Step-Loop, and the Bar-Loop outperformed GP with no loops. It is also clear that the Step-Loop, and the Bar-Loop achieved results better than the best result produced by Li's loop. Li's loop and the Bar-Loop were the only loop structures that achieved 100% accuracy on the training set, demonstrating the expressivness of rectangular shaped loop regions for digit recognition. Although the Seam-Loop performed the best for the Cell problem, it performed poorer than GP with no loops for the MNIST problem. Seams were effective at targeting signifcant parts of cells, which have lots of internal content, but were not able to effectivly target changes in the shape of digits. The extra search space produced by an almost useless loop structure increased learning effort so that the Seam-Loop could not learn a good solution within the learning period. The EB-Loop performed better relative to the Cell dataset. The simple structure of digits may have meant that simple loop structures are adequite for classification, leading to small loops that are less likely to change during evolution.

For the Face problem, the Bar-Loop was the only loop structure that achieved better mean classification accuracy on the test set than GP with no loops. The Bar-Loop also achieved the overall best result for the test set. The mean accuracy for the training set was about equal to Li's loop, GP with no loops, and the Seam-Loop which indicates that the Bar-Loop could better generalise on unseen data because of the smaller differences between training and test accuracies. The Step-Loop, Seam-Loop, and Sort-Loop performed no better than GP with no loops, indicating that non-rectangular loop regions may not be suitable for classifying localised regions in face images, such as the eyes, nose, and mouth. The EB-Loop, although producing rectangular regions, could not outperform GP with no loops. The increased search space, coupled with a difficult problem, may have required more learning effort to evolve a good classifier.

Observing the results for the Cell, MNIST, and Face problems, only the Bar-Loop could consistently outperform GP with no loops on unseen test data. Li's loop and the Step-Loop achieved results better than GP with no loops for the Cell and MNIST problems, but not for the the Face problem. The larger search space of Li's loop and the Step-Loop may have hindered learning performance when considering the restriction on the number of generations permitted to learn a solution.

Table 5.1: GP Results for Image Problems: Part 1

Data Set	**Fitness Set**	**Variables**	**No-Loop**	**Li**	**EB-Loop**	**Step-Loop**	**Sort-Loop**	**Bar-Loop**	**Seam-Loop**
Cell	Training	Mean (%)	95.92	96.14	95.17↓⇓	95.92	89.95↓⇓	96.40↑⇑	97.43↑⇑
		Std. Dev. (%)	0.93	0.59	2.06	1.26	0.86	0.80	0.30
		Best (%)	98.20	97.80	97.80	98.00	92.40	98.00	98.60
		Duration (s)	18.29	50.17	106.59	437.03	243.77	953.45	214.16
	Testing	Mean (%)	92.60	94.02↑	92.68⇓	93.42↑⇓	86.58↓⇓	94.53↑⇑	94.78↑⇑
		Std. Dev. (%)	1.36	0.91	2.41	1.96	1.39	1.23	0.45
		Best (%)	95.60	96.00	96.00	96.40	90.00	96.80	96.00
MNIST	Training	Mean (%)	99.34	99.52↑	98.83↓⇓	99.46↑⇓	66.37↓⇓	99.38⇓	98.52↓⇓
		Std. Dev. (%)	0.38	0.14	0.85	0.21	0.64	0.26	0.70
		Best (%)	100.00	100.00	99.80	99.80	68.40	100.00	99.80
		Duration (s)	53.20	73.32	155.46	434.82	376.30	1082.61	118.15
	Testing	Mean (%)	96.70	98.87↑	97.95↑⇓	98.46↑⇓	61.68↓⇓	98.63↑⇓	95.77↓⇓
		Std. Dev. (%)	0.84	0.42	1.30	0.63	1.03	0.81	1.14
		Best (%)	99.00	99.40	99.60	99.60	63.80	99.60	97.80
Face	Training	Mean (%)	96.51	96.11↓	93.50↓⇓	95.77↓	84.99↓⇓	96.18	96.09↓
		Std. Dev. (%)	1.00	1.65	2.23	2.06	2.82	1.66	1.32
		Best (%)	98.40	99.20	97.20	99.00	89.60	99.00	98.60
		Duration (s)	13.83	31.38	95.08	300.28	285.27	549.25	53.97
	Testing	Mean (%)	91.14	90.69	87.86↓⇓	91.56⇑	86.71↓⇓	92.87↑⇑	90.97
		Std. Dev. (%)	1.65	2.70	2.74	3.07	2.43	2.11	2.25
		Best (%)	94.60	96.20	93.00	96.40	92.80	97.00	96.00

5.1.2 Results for Images: Part 2

Results for the Pedestrian, Box-Horz, and Box-Vert problems are displayed in Table 5.2. For the Pedestrian problem, only Li's loop and the Sort-Loop could achieve a better mean training accuracy than GP with no loops, but the increase is not statistically significant. Poor performance from GP with loop structures may be attributed to the difficult nature of pedestrian identification. Background objects, camera angle, posture, and clothing all serve to add complexity to identifying pedestrians. The loop structures were inadequate for targeting obscured features. This may be because large loop regions readily classify background objects as pedestrians. GP with no loops was better able to find a function that mitigates background issues. Although GP with no loops performed better on average, top results achieved by GP with Li's loop, Sort-Loop, and, in particular, the Bar-Loop were better than GP with no loops, indicating that loop structures can achieve better results than GP with no loops if used correctly.

The Box-Horz and Box-Vert problems are generalisations designed to measure the performance of GP when targeting small moving objects on a particular axis. The Box problems were easily solved by most loop structures. All GP systems with loops (except the Sort-Loop) performed better than GP with no loops for the Box-Horz problem. The Sort-Loop being the exception because pixel values are sorted, removing box location information. It is clear that the loop structures can differentiate between boxes on the left and on the right. GP with no loops performed worse because only individual pixel values must be inspected while loop regions are able to easily cover an entire side of an image. Besides GP with no loops and GP with the Sort-Loop, the Step-Loop loop performed the worst. The Step-Loop's 1D representation of data has made horizontal feature changes difficult to analyse. Poorer results on average may be an indication of the difficulty of learning vertical loop regions that do not 'spill' into an adjacent side. Li's loop and the EB-Loop did not perform as well as the Bar-Loop and Seam-Loop. The specification of both corners of loop regions may have increased the search space and made it difficult to target an entire width or height of an image side. The Bar-Loop and Seam-Loop performed well with a smaller search space.

Similar results were found for the Box-Vert problem with two notable differences. The Step-Loop performed notably better than it did for horizontal changes in position. The Step-Loop is more easily capable of producing horizontal loop regions that can better target when a feature is at the top or bottom of an image. The Seam-loop performed worse than GP with no loops because only vertical seams are generated. Without horizontal seams, changes in the vertical position of features becomes a difficult task. It is assumed that GP with the Seam-Loop resorted to analysing individual pixel values, like GP with no loops.

For the Box problems, GP with the Bar-Loop consistently outperformed the remaining GP systems. The vertical and horizontal orientation settings for the Bar-Loop were undoubtedly one reason for the success. Li's loop and the EB-Loop also performed well for the two Box problems. Regarding the Pedestrian problem, GP with the Bar-Loop, Li's loop, and the Sort-Loop achieved more accurate best results than GP with no loops, but no loop structure could achieve better average results than GP without loops for the test set.

Table 5.2: GP Results for Image Problems: Part 2

Data Set	Fitness Set	Variables	No-Loop	Li	EB-Loop	Step-Loop	Sort-Loop	Bar-Loop	Seam-Loop
Pedestrian	Training	Mean (%)	83.07	84.58↑	80.60↓⇓	82.05↓⇓	82.35↓⇓	81.20↓⇓	82.06↓⇓
		Std. Dev. (%)	1.41	2.29	1.60	1.50	2.58	1.87	1.43
		Best (%)	87.60	90.40	84.20	86.20	88.40	88.20	85.00
		Duration (s)	13.57	32.45	169.63	363.34	268.28	1367.48	88.33
	Testing	Mean (%)	77.81	78.12	76.75↓⇓	76.56↓⇓	78.38	76.51↓⇓	76.87↓⇓
		Std. Dev. (%)	1.48	3.02	1.95	2.26	3.28	3.03	1.91
		Best (%)	81.60	85.40	80.60	81.00	85.20	88.20	81.20
BoxHorz	Training	Mean (%)	86.14	95.27↑	97.79↑⇑	89.19↑⇓	51.97↓⇓	99.99↑⇑	99.33↑⇑
		Std. Dev. (%)	2.80	1.86	3.12	5.15	0.16	0.07	1.41
		Best (%)	91.60	99.40	100.00	100.00	52.20	100.00	100.00
		Duration (s)	19.29	111.87	427.34	228.63	213.18	163.45	97.87
	Testing	Mean (%)	81.87	92.24↑	96.59↑⇑	86.35↑⇓	48.42↓⇓	99.93↑⇑	99.03↑⇑
		Std. Dev. (%)	3.43	2.59	3.89	6.16	0.12	0.27	1.73
		Best (%)	88.20	97.80	100.00	100.00	49.00	100.00	100.00
BoxVert	Training	Mean (%)	85.98	99.42↑	98.00↑⇓	99.84↑⇑	50.96↓⇓	99.98↑⇑	84.85↓⇓
		Std. Dev. (%)	2.73	0.79	2.48	0.44	0.29	0.18	3.03
		Best (%)	91.60	100.00	100.00	100.00	51.20	100.00	90.80
		Duration (s)	11.85	104.19	460.10	147.77	229.93	146.42	103.20
	Testing	Mean (%)	80.85	98.32↑	96.60↑⇓	99.12↑⇑	48.97↓⇓	99.85↑⇑	79.93⇓
		Std. Dev. (%)	3.12	1.55	3.40	1.15	0.29	0.43	4.01
		Best (%)	87.80	100.00	100.00	100.00	49.80	100.00	89.60

5.1.3 Discussion

Considering all six image problems, the Bar-Loop consistently achieved results better than the remaining loop structures, followed by Li's loop and the Step-Loop. The reduced search space of the Bar-Loop improves the probability of finding good solutions while the enforcement that bars must touch two opposing sides of an image ensure that the Bar-Loop can target important features for structural analysis and background features for analysis of the shape and size of objects. The Seam-Loop performed well for objects where features are on a horizontal axis as evidenced by good performance on circular cells and the Box-Horz problem. But, the Seam-Loop performs poorly for features on a vertical axis such as digits in the MNIST problem, faces (eyes are above nose for example), and the Box-Vert problem. The Sort-Loop, being designed for 1D non-image data where neighbouring values are not considered correlated, is not suitable for 2D image data because pixels are shuffled in a way that removes correlations between neighbouring pixels and remove feature location information.

5.2 Non-Image Classification

All GP systems except for the Bar-Loop and Seam-Loop were evaluated against all four non-image datasets. Li's 1D loop was used for 1D image data. Results are aggregated from 100 independent runs of each experiment. Results for the Forest, Spam, Poker, and MiniBooNE problems are displayed in Table 5.3. Training and test set mean accuracies from GP with loops are marked with $\uparrow$ or $\downarrow$ if statistically better or worse than GP without loops and are marked with $\Uparrow$ or $\Downarrow$ if statistically better or worse than GP with Li's loops. A paired two-tail *t* test was performed [29, 30]. Considering all non-image problems, the Step-Loop and Li's 1D loop achieved better results than GP with no loops consistently. The Sort-Loop achieved notable results as well for the Poker and MiniBooNE problems.

5.2.1 Results for Non-Images

Regarding the Forest Cover Type problem, it is clear that Li's loop and the Step-Loop are notably outperforming GP with no loops on the test set by 6.84% and 11.07% respectfully. The capability of Li's loop to iterate over a sequence of data has identified nearby features significant for classification while the Step-Loop was able to perform even better by targeting distant features that may have more significant correlations. By stepping over data values, the Step-Loop also produced the best overall result, a good indication of the increased expressiveness of the Step-Loop over the remaining GP structures. The EB-Loop performed slightly better than GP without loops but not as good as Li's loop. Although sequences of values can be iterated over, the lack of ensured validity of loop structures had hindered classification performance. The Sort-Loop performed the worst, possibly by shuffling data to remove any existing correlations between neighbouring data values.

Li's loop and the Step-Loop performed equally well for the Email Spam problem, although GP with no loops achieved the best overall result. This indicates that the significant features of spam in the dataset are not ordered sequentially or at regular intervals. GP with no loops can target any combination of values given a large enough tree size, which was the case. Notwithstanding, GP with no loops had the largest standard deviation between runs so it is not as reliable and consistent as the remaining loop structures.

For the Poker Hand dataset, the Step-Loop achieved perfect classification accuracy on both the training set and test set for the majority of runs. The Step-Loop was able to step over suite values and only target card values. Suite values have no influence between an empty

Table 5.3: GP Results for Non-Image Problems

Data Set	Fitness Set	Variables	No Loop	Li	EB-Loop	Step-Loop	Sort-Loop
Forest	Training	Mean (%)	79.39	80.15↑	77.05↓⇓	84.08↑⇑	66.85↓⇓
		Std. Dev. (%)	1.90	2.02	2.44	2.54	1.56
		Best (%)	82.00	83.20	82.20	89.20	72.80
		Duration (s)	15.22	50.59	47.58	144.53	65.03
	Testing	Mean (%)	68.12	74.96↑	69.69↑⇓	79.19↑⇑	62.62↓⇓
		Std. Dev. (%)	3.36	6.23	5.66	6.95	2.81
		Best (%)	88.40	89.80	90.20	94.20	70.60
Spam	Training	Mean (%)	89.34	91.41↑	88.31↓⇓	91.20↑	72.78↓⇓
		Std. Dev. (%)	2.52	1.47	2.13	1.63	1.56
		Best (%)	93.80	93.80	91.80	93.80	77.40
		Duration (s)	9.95	46.46	42.04	67.66	57.14
	Testing	Mean (%)	88.25	89.43↑	86.88↓⇓	89.19↑	74.31↓⇓
		Std. Dev. (%)	2.85	1.35	2.49	1.90	1.58
		Best (%)	93.60	92.00	91.40	92.20	78.80
Poker	Training	Mean (%)	58.22	71.11↑	56.69↓⇓	100.00↑⇑	84.95↑⇑
		Std. Dev. (%)	0.70	2.14	0.83	0.00	0.76
		Best (%)	60.60	74.80	59.00	100.00	87.80
		Duration (s)	12.83	59.69	24.14	12.77	46.26
	Testing	Mean (%)	52.63	66.14↑	52.65⇓	99.93↑⇑	87.56↑⇑
		Std. Dev. (%)	1.83	2.01	1.87	0.18	0.96
		Best (%)	56.80	69.20	57.00	100.00	89.80
MiniBoo	Training	Mean (%)	83.26	84.71↑	82.70↓⇓	88.72↑⇑	87.85↑⇑
		Std. Dev. (%)	0.37	1.43	0.53	1.95	1.54
		Best (%)	84.20	88.60	84.40	92.40	90.00
		Duration (s)	14.36	62.92	37.73	149.83	76.80
	Testing	Mean (%)	76.30	78.71↑	75.89↓⇓	81.80↑⇑	83.68↑⇑
		Std. Dev. (%)	0.91	1.34	1.13	2.24	1.93
		Best (%)	79.60	82.60	83.00	85.20	86.60

hand and a pair. The Sort-Loop was also able to target card values by sorting each had such that pairs became neighbouring values. Iterating over neighbouring pairs produced significant classification results. Results were not as good as the Step-loop because suite values were still iterated over. Li's loop also faired well while the EB-Loop and GP with no loops could not classify between Poker hands. A 50% classification accuracy is no better than a random guess.

All loops but the EB-Loop outperformed GP with no loops for the MiniBooNE Particle Identification problem. The Sort-Loop achieved the best results followed by the Step-Loop and Li's loop structure. Sorting records to group values of similar magnitude allowed the Sort-Loop to identify stronger correlations between data values. Similarly, the Step-loop was able to correlate distant values while Li's loop and the EB-Loop found it difficult to find sequential correlated values in records.

5.2.2 Discussion

For Non-Image problems, both Li's loop and the Step-Loop consistently outperformed GP with no loops, indicating that loop structures produce more accurate classifiers than GP with no loops. Further, for all but the Spam problem, the Step-Loop notably outperformed Li's loop. This provides evidence that loop structures must be able to target distant values to achieve better results for non-image classification tasks because neighbouring values are not considered correlated. As further evidence, the Sort-Loop achieved good results for both the Poker Hand and MiniBooNE problems by sorting data values to find correlations between values of similar magnitude. The EB-Loop could not outperform GP with no loops for any of the four non-image datasets. The complexity of the EB-Loop coupled with the ability to only target sequential data values is not effective for non-image classification tasks.

5.2.3 Comparison with Alternative Classifiers

Non-Image data is suitable for classifiers such as Decision Trees and Naïve Bayes because feature separation and feature correlation in extracted features are easily identified and used for classification. It is interesting to compare GP with and without loops to the Decision Trees and Naïve Bayes classifiers in order to understand the suitability of the methods when compared to alternative classifiers. Table 5.4 displays the results of the Decision Tree and Naïve Bayes classifiers on the test set for each non-image classification problem. The best results for GP with no loops and GP with loops are also displayed for comparison. Weka v3.6.5 [31] was used. The Decision Tree classifier was J48 and the standard Naïve Bayes classifier was used.

Table 5.4: Classifier Comparison on Non-Image Problems

Data Set	Decision Tree (%)	Naïve Bayes (%)	GP: Best No Loop (%)	GP: Best Loop (%)
Forest	84.4	89.4	88.4	94.2
Spam	90.8	81	93.6	92.2
Poker	48.4	51.4	56.8	100
MiniBooNE	83	83.8	79.6	86.6

For the Poker Hand and Spam problems, GP achieved more accurate classification performance on unseen data. The distribution of features in the Spam dataset may not have been ideal for the Decision Tree and Naïve Bayes classifiers which would have lead to worse results than GP. GP was able to automatically learn a function to separate classes, regardless of feature distribution.

Regarding the Poker Hand problem, as the pairs in a Poker hand can occur between any two cards, only GP with loops could perform classification. Further, while GP without loops could not out perform the Decision Tree and Naïve Bayes classifiers for the Forest and MiniBooNE problems, GP with loops achieved the best results that notably outperformed the alternative classifiers. It is evident that GP with loops could utilise standard GP functions and terminals to separate classifications and was able to use loop structures that could target subsets of data that the remaining classifiers could not target. Thus, GP with loops could find significant correlations in a large search space to achieve better classification results.

5.3 Comparison of Loops

Considering the observed results as a whole, it is apparent that some trends have been identified. Li's loop structures performed better than GP with no loops on unseen data for four out of the six image problems and for every non-image problem. Although Li's loop performed better than GP with no loops on average, the Bar-Loop was able to achieve slightly better results than Li's loop on unseen data and more accurate best results. The Bar-Loop also outperformed GP with no loops for five out the six image problems. Like Li's loop, the Step-Loop performed consistently better than GP with no loops for non-image datasets, even outperforming Li's loop for three out of four non-image problems. It is clear from the experiments that Li's loop and GP with no loops takes shorter time to evolve good solutions but are not able to achieve the same accuracy as some loops that take longer to evaluate, despite similarity in loop structure.

For image data, using predefined loop operators and rectangular regions demonstrated to be a good combination for achieving consistently good results. Neighbouring pixel correlations are used and the search space is reduced by not learning new loop operators. The probability of learning good solutions increases. For non-image classification tasks, the notable performance of the Sort-Loop and, in particular, the Step-Loop demonstrates that loop structures must be able to target any combination of values to find significant correlations. This has been achieved by reordering data values with the Sort-Loop and 'stepping' over data values with the Step-Loop.

In the Chapter 6, learned classifiers are analysed to derive reasoning behind any good or poor performance observed.

Chapter 6

Analysis of Evolved Classifiers

Simplicity and understandability of learned solutions can be an important factor for consideration when developing a GP system to solve a problem. Simple solutions that are easily interpretable provide insight into which features are important for classification and how those features can be used together to achieve good results. In Section 6.1, the identified loop regions of the best programs are analysed and compared to understand how well identified loop regions reflect the nature of the problem being solved. In Section 6.2, some good programs are analysed to measure human understandability and problem solving capability. Note that only image problems are used for loop region analysis as regions for non-image data are not easily visualisable. As loop regions in images represent features, loop regions in non-images represent correlations between data values which does not offer significant insight if visualised.

6.1 Analysis of Loop Regions

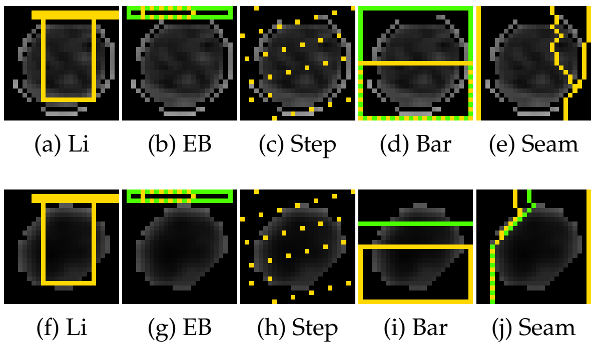
The following Figures describe the loop regions of the best solutions for each combination of loop method and image problem. Due to the inadequacy of the Sort-Loop for image data, the loop regions of the Sort-Loop are not described. Image regions are highlighted with a yellow border. For clarity, only significant loop regions are displayed[1] and a green border is used to distinguish between overlapping loop regions. Border colour does not signify any difference of loop regions.

Cell Figure 6.1 displays the loop regions of the best solutions for the Cell classification problem. It is clear that each loop structure used a different method to target cell features.

Li's loop generated two notable loop regions. The first analyses the cell body or internal structure while the second has produced a narrow rectangle which detects changes in cell size. Looking at both classes of cell images, it is apparent that the internal structure of class 2 cells (bottom) is smoother than that of class 1 cells. Class 2 cells are also slightly smaller due to a less prominent border. Li's loop effectively targeted these features. The Bar-Loop performed slightly better by using a pixel value to determine the size of one of the loop regions (Figures 6.1d and 6.1i). Due to the restriction of bars to touch two opposing sides of the image, the bottom loop (in yellow) is able to target both the internal content and size of the cell. The top loop (in green) enlarges if the cell is large and contracts when the cell is small, providing a clear adaptation to cell size for better classification results. The EB-Loop generated two valid loop structures, both of which analyse cell size based on changes in cell

[1]Based on author's judgement by analysing the problem and evolved program.

height. Although only cell size is targeted, the use of two loops coupled with a wider loop region enabled the EB-Loop to achieve the same results as Li's loop. Although, only addition was learned in loop bodies which is more easily expressed by the Summation loop operator. The Step-Loop produced a good result by targeting the entire area of the images. Changes in cell size and internal structure are more likely to be analysed. If cells were differentiated by localised features instead, the Step-Loop would find it difficult to learn a good loop structure. The Step-Loop can target lines on any axis but not rectangles, which is a limitation for image data. The Seam-Loop was able to automatically target seams of significance, which appear to be the cell border. Each class has a slightly different border which the Seam-Loop is using for classification. Size was also analysed by the right-most seam. In the class 1 cell (top) the seam was near enough to target the larger cell. In the class 2 cell (bottom), the cell was too small to be detected by the right-most seam.


(a) Li (b) EB (c) Step (d) Bar (e) Seam

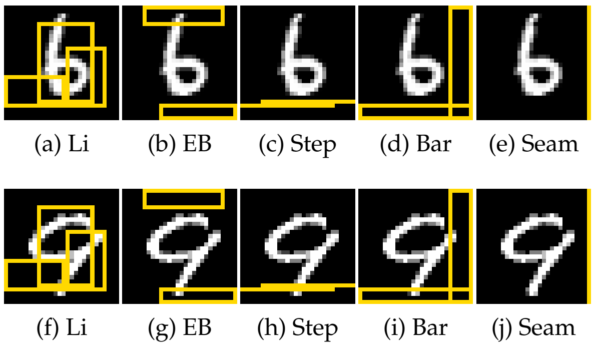
(f) Li (g) EB (h) Step (i) Bar (j) Seam

Figure 6.1: Evolved loops of the best solutions for the Cell problem. The top row are class 1 images and the bottom row are class 2 images.

MNIST Figure 6.2 displays the learned loop regions for the MNIST problem. Comparing all loop structures, Li's loop and the Seam-Loop were the 'odd ones out'.

As mentioned in Section 5.1.2, the Seam-Loop is not readily capable of identifying vertical changes in features. Thus neither the top or bottom features that differentiate sixes and nines can be targeted. The Seam-Loop used one loop (right-most column in Figures 6.2e and 6.2j) which may have either identified peculiarities in the drawing of digits in the used dataset, produced a necessary zero value for an equation, or was part of a redundant subtree. Li's loop targeted the base of digits for the bottom of a six or the 'stalk' of a nine. The largest centre-most loop analysed the mass of the digit for classification.

The remaining loop structures, the EB-Loop, Step-Loop, and Bar-Loop, produced simple solutions that target either the top or bottom of the digits which is clearly a notable difference between sixes and nines. The Bar-Loop produced a loop running down the right of the image which may have targeted peculiarities in the drawing of either a six or a nine. For instance, sixes tend to be slanted to the left while nines tend to be slanted to the right.

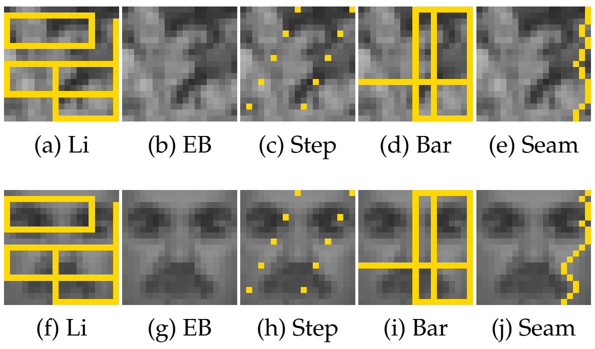

(a) Li (b) EB (c) Step (d) Bar (e) Seam

(f) Li (g) EB (h) Step (i) Bar (j) Seam

Figure 6.2: Evolved loops of the best solutions for the MNIST problem. The top row are class 1 images (6s) and the bottom row are class 2 images (9s).

Face The evolved loop regions for the Face problem are displayed in Figure 6.3. Li's loop and the Bar-Loop successfully targeted the significant features of the Face: eyes, nose, and mouth by using rectangular loop regions.

Li's loop targeted broad areas of the eyes, nose, and mouth regions on both sides of the face while the Bar-Loop targeted significant facial features on one side of the face only, exploiting the symmetric nature of faces. The Seam-Loop also performed well because seams could automatically identify facial features. In Figure 6.3j, the Seam-Loop targeted features around the eyes and mouth which may be distinguishable from random features as found in Figure 6.3e. The EB-Loop did not learn any valid loop structures which had limited its classification accuracy. Learning valid loop bodies in parallel with identifying precise localised regions may have been difficult given the restricted number of generations to evolve programs. The Step-loop iterated over the eyes, nose, and mouth, but could not focus on any localised regions in particular due to the linear nature of the iteration algorithm.

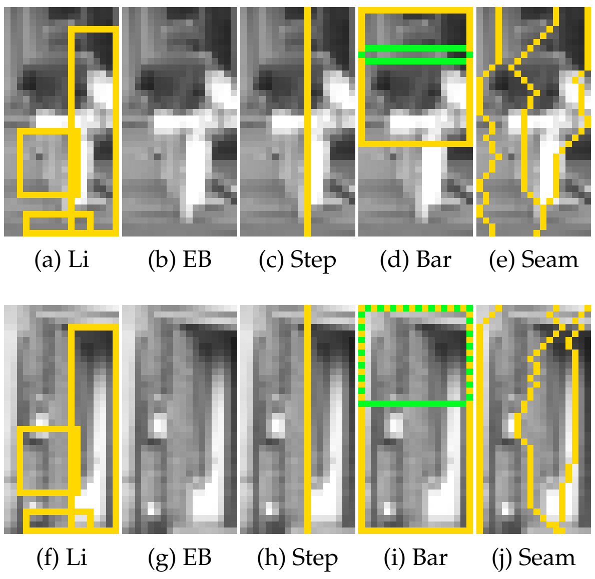

(a) Li (b) EB (c) Step (d) Bar (e) Seam

(f) Li (g) EB (h) Step (i) Bar (j) Seam

Figure 6.3: Evolved loops of the best solutions for the Face problem. The top row are class 1 images and the bottom row are class 2 images.

Pedestrian Evolved loop regions for the Pedestrian problem are displayed in Figure 6.4. It is clear that despite a complex background filled with numerous different objects, the loop structures were still able to identify pedestrians from background. The task is made more difficult due to inconsistent camera angles and pedestrian posture which has clearly effected the EB-Loop and the Step-Loop.

GP with Li's loop structure identified several loop regions around the lower body area. Despite the three loop regions displayed, loop regions were duplicated several times. There were 16 loops in total, which indicates a high level of redundancy and a large and complex classifier. Although complex, the produced classifier was more effective than GP with no loops. It may be common for lower body clothing such as pants, shorts, or skirts to be of one colour and have plain designs. This uniformity may be a clear difference from the seemingly random background and, thus, reliable for classification. The Bar-Loop instead targeted the upper body with only two significant loops regions. Pixel values are used to vary the thickness of both loop regions. The yellow loop region targeted the upper body of pedestrians and the entire image for non-pedestrians while the green loop region targeted the neck or head for pedestrians and the upper image for non-pedestrians. It may well be the case that the Bar-Loop solution classifies true for pedestrian


(a) Li (b) EB (c) Step (d) Bar (e) Seam

(f) Li (g) EB (h) Step (i) Bar (j) Seam

Figure 6.4: Evolved loops of the best solutions for the Pedestrian problem. The top row are class 1 images and the bottom row are class 2 images.

if the aggregate result from both loop regions is small.

The EB-Loop did not evolve any valid loop regions due to the complexity of the loop, the difficulty of the problem, and the restriction on the number of generations to evolve a classifier. The Step-Loop generated a single loop structure that iterates vertically down the centre of the image. This loop region maximises the amount of time iteration occurs over a pedestrian relative to background objects. Although helpful, the loop region does not provide as much insight as the Bar-loop or Li's loop into which features are useful for classifying pedestrians. The Seam-Loop achieved a slightly better result because seams could automatically 'steer' towards the pedestrian if one exists. Although effective for pedestrians, the randomness of background objects would certainly be difficult to distinguish because the next-most significant object may be targeted by the Seam-Loop instead and possibly be incorrectly classified as a pedestrian.

Box Loop regions learned for the Box-Horz and Box-Vert problems are displayed in Figures 6.5 and 6.6 respectfully. From both Box problems, it is clear that the loop structures that require the specification of both the start and the end coordinates of rectangles produce disorganised results. Li's loop and the EB-Loop both targeted the left side of the image

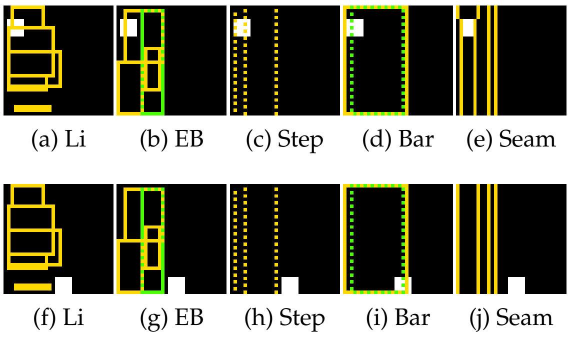


Figure 6.5: Evolved loops of the best solutions for the Box-Horz problem. The top row are class 1 images and the bottom row are class 2 images.

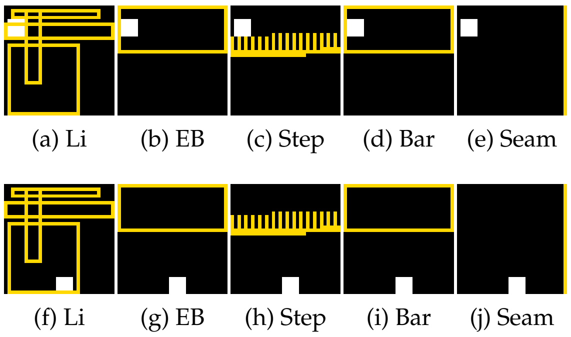


Figure 6.6: Evolved loops of the best solutions for the Box-Vert problem. The top row are class 1 images and the bottom row are class 2 images.

for the Box-Horz problem (Figure 6.5) but produce multiple overlapping loop regions. The evolved classifier will not be efficient and it can be difficult to understand how the classifier works. The disorganisation of loops is a byproduct of the flexibility of the loop structures. The flexibility of more precisely defining loop regions increases the search space such that it becomes difficult to find good solutions. To compensate for poor loop regions, *more* loop regions are added to produce good results which incurs bloat and duplication of subtrees. Program bloat is a result of the randomness of the crossover and mutation genetic operators. However, the Bar-Loop, Step-Loop, and Seam-loop all produced simple results that can be understood with less effort in comparison. Only a few distinct loops that target just one side of the image for classification had been produced. Li's loop was the only loop structure that developed loop regions that could not achieve a 100% accuracy for the Box-Horz problem.

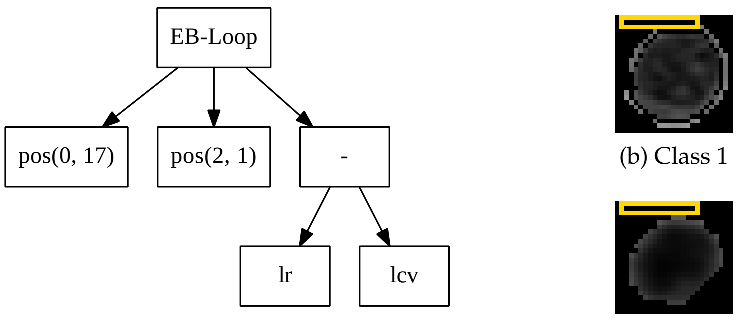
The best solution with Li's loop for the Box-Vert problem (Figure 6.6) produced a complex result that contained more loop structures than the remaining loops. Furthermore, the loop regions have 'spilled' into an opposing side of the image. Although this can be useful for classification by checking if a box is on the opposite side, it makes analysing the solution more difficult. When analysing the solution, it may seem like targeting the opposing side of an image is useful, but the usefulness may be negligible. The EB-Loop produced several perfect classifiers (on the test set), the displayed solution contained only one loop while the remaining solutions where as complex as Li's loop. Due to the increased flexibility of the EB-Loop and Li's loop, the probability of learning simple solutions is smaller than that of the remaining loop structures. The Step-Loop and Bar-Loop could clearly solve the Box-Vert problem in a human understandable manner. The Seam-Loop can only generate vertical seams so it could not adequately solve the Box-Vert problem. Only one seam was used (right-most column in Figures 6.6e and 6.6j). The seam clearly cannot effectively identify which side the box is on.

6.2 Analysis of Learned Solutions

It is essential to understand how learned solutions work for analysing which features are being targeted, how different features are used in conjunction, and which operators are significant in order to gain a better understanding of the problem domain. The features that are targeted can be easily visualised (as in Section 6.1) to provide insight into which features are important for classification. What is left unclear is how the identified features are used together to generate an output for classification. Understanding how combinations of features are utilised allows GP programs to be interpreted into simpler programs that can be turned into executable instructions. This analysis of identified features along with how features are utilised provide insight into the problem domain which may not have been well understood before analysis. Simple solutions as a result of expressive GP nodes facilitate in the easier analysis of solutions. Below, some typical solutions are presented and discussed. The Key below describes the meaning of abbreviations used:

- **pos(r, c)** 2D position terminal with coordinates (row, column). Describes a 2D position.
- **pos(i)** 1D position terminal with index i. Describes a 1D position.
- **pv(r, c)** Pixel-Value terminal with coordinates (row, column). Returns the pixel value at coordinates (r, c).

EB-Loop The EB-Loop was able to outperform GP with no loops for the Cell dataset. One good program learned by GP with the EB-Loop is illustrated in Figure 6.7.



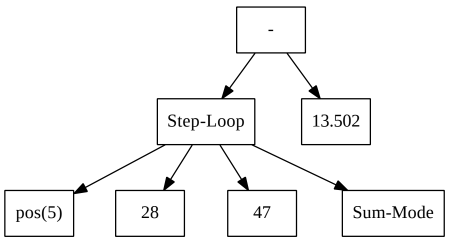
(a) Program Tree (b) Class 1 (c) Class 2

```
(EB-Loop pos-[row:0_col:17] pos-[row:2_col:1] (- lr lcv))
```

Figure 6.7: Example EB-Loop Solution for the Cell Problem

It is clear that the learned solution is simple, consisting of a tree depth of three. The EB-Loop performs a subtraction operation over most of the first three rows. The `lcv` abbreviation indicates `loopCurrentValue` and the `loopResult` terminal is indicated by `lr`. For every pixel value within the rectangle `pos(0, 17)` and `pos(2, 1)`, `lcv` is populated with the pixel value and subtracted by `lr` (accumulated result). The `lr` terminal is then populated with the result ($lr = lr - lcv$). There is no redundancy or unnecessary bloat in the program, leading to a simple solution that can be quickly analysed to identify how cells are being differentiated. It is clear that if the cell is big, a negative value will be returned for a class one classification; and if the cell is small, a zero value will be returned for a class two classification. It is also clear that due to the simple loop body, consisting of subtraction, a loop structure that accepted the Summation loop operator terminal would be simpler for GP to learn and for users to analyse.

Step-Loop The Step-Loop notably outperformed every other GP system for the Poker Hand problem, achieving a 100% accuracy on both the training and test sets majority of the time. Analysing how the Step-Loop classified Poker hands can allow us to undertstand why such good performance was learned. A perfect solution learned by GP with the Step-Loop for the Poker Hand problem is illustrated in Figure 6.8.



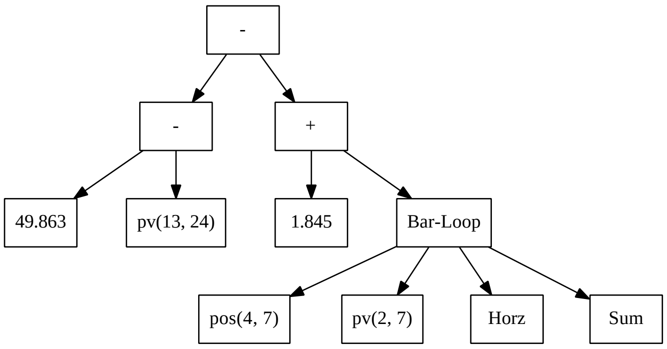
```
(- (Step-Loop pos-5 drand-28.498 drand-47.275 Sum-Mode) drand-13.502)
```

Figure 6.8: Example Step-Loop Solution for the Poker Hand Problem

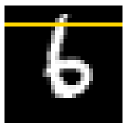
The Step-Loop iterates over the following pattern nine times: `(5, 3, 1, 9, 7)`. The values are produced because iteration begun at `pos(5)` and the index was incremented by 28 each cycle. The pattern was duplicated because iteration terminated at 47 cycles (see Algorithm 3.1.1). It is obvious that every index iterated over is an odd numbered index. In the Poker Hand dataset, suits are at even indices and cards are at odd indices (see Table 4.1). The Step-Loop only iterates over cards because suits have no influence on whether two cards make a pair. The Sum-Mode operator is applied over all card values. For an empty hand, all card values are unique so they all occur with the same frequency. The Sum-Mode operator will return the sum of every card value. If every card is unique, the Sum-Mode operator will return a value that is *no smaller* than 15 ($1 + 2 + 3 + 4 + 5$). Subtraction by 13.502 will yield a positive value that correctly classifies a hand as empty. If a single pair exists (two cards of the same value), the result of the Sum-Mode operator will be *at most* 13 (two Kings). Subtraction by 13.502 will yield a negative value that correctly classifies a hand as having a single pair. It is clear that the simple nature of the solution has left no ambiguity as to how the classification problem was solved.

Bar-Loop The Bar-Loop had been able to achieve more accurate results than most other loop structures for the majority of image classification problems. One good solution learned by the Bar-Loop for the MNIST problem is illustrated in Figure 6.9.

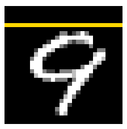
The displayed Bar-Loop solution uses pixel values which has hindered the understandability of the solution because the loop region is dependent on digit shape rather than being consistent across images. Although a hindrance, with some effort, one can find that the Bar-Loop region is a one pixel horizontal line across the top of the image. The pixel value



(a) Program Tree



(b) Class 6



(c) Class 9

```
(- (- drand-49.863 pixel-value-[row-13_col-24]) (+ drand-1.845 (Bar-Loop
pos-[row:4_col:7] pixel-value-[row-2_col-7] Horz Sum)))
```

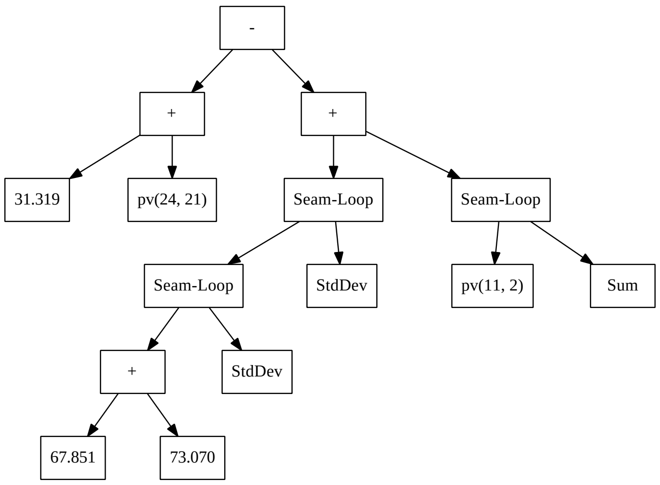
Figure 6.9: Example Bar-Loop Solution for the MNIST Problem

used for the thickness of the Bar-Loop is located in a position that is almost certainly void (pixel value of zero). The value is constrained to be at least one pixel. The loop region is targeting the top stalk of the six which protrudes further than the head of the nine due to digit positioning. Digits are positioned based on centre of mass. Therefore, bottom heavy sixes appear slightly higher than top heavy nines. If a six is found, a value greater than zero is produced due to the Summation operator and a value of zero is produced for a nine.

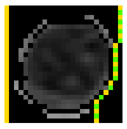
The left subtree will produce a value of about 50 if the pixel value is zero and about 205 (255 colour depth - 50) otherwise. The left subtree therefore operates as a threshold for the right subtree. If a prominent stalk was found, the value of the right subtree would be greater than the value of the left subtree, producing a negative value — a 'six' classification. If no stalk was found or only a part of a stalk was found (possibly the top edge of the head of a nine), the right subtree result will be larger, producing a positive result — a 'nine' classification. The solution is simple, but it does contain redundancy. The left subtree could be replaced with a constant value and the right subtree does not need the addition with 1.845. Of course, the simplicity of the solution promotes understanding to make simplifications easy.

Seam-Loop The Seam-Loop performed best for the Cell classification problem. An illustration of a good solution with the Seam-Loop is displayed in Figure 6.10.

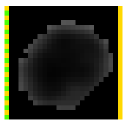
Clearly, the displayed classifier is larger than the other programs. The program illustrated targets either the left or right border of cells. If the cell is small, the right-most Seam-Loop will target the left border. If the cell is large, the right-most Seam-Loop will target the right border. A pixel-value terminal is used to test cell size. There is also a nested Seam-Loop. The nested Seam-Loop will always target the right of images ($67 + 73 = 140$ is restricted to the image width). If the cell is large, the seam will detect the cell and maneuver to it which means the parent Seam-Loop targets the right of the image. If the cell is small, a zero value is produced and the parent Seam-Loop targets the left side of the image. The program is heavily utilising cell size for classification. Like the Bar-Loop program in Figure 6.9, the left subtree behaves like a threshold.



(a) Program Tree



(b) Class 1

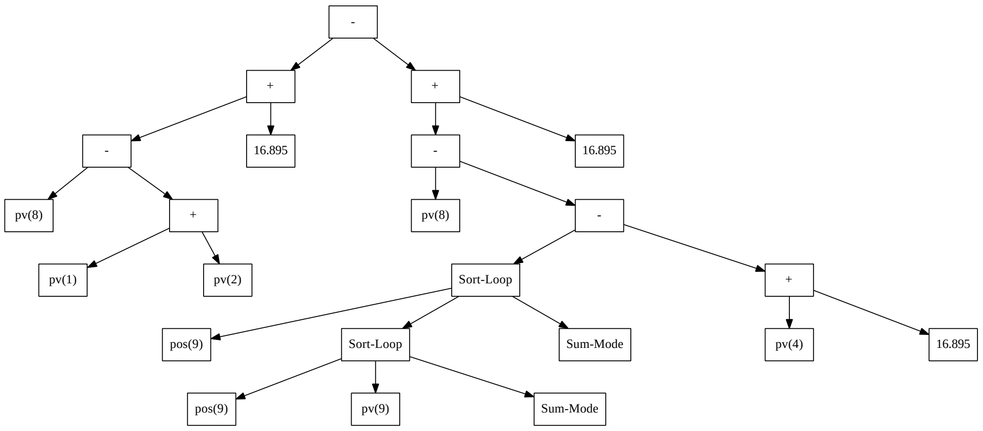


(c) Class 2

```
(- (+ drand-31.319 pixel-value-[row-24_col-21]) (+ (Seam-Loop (Seam-Loop (+
drand-67.851 drand-73.070) StdDev) StdDev) (Seam-Loop pixel-value-[row-11_col-2]
Sum)))
```

Figure 6.10: Example Seam-Loop Solution for the Cell Problem

Sort-Loop The Sort-Loop did better than all but the Step-Loop for the Poker Hand classification problem. It is interesting to analyse a Sort-Loop solution to better understand why it did not do better. A good example of a Sort-Loop solution for the Poker Hand problem is displayed in Figure 6.11.



```
(- (+ (- pixel-value-8 (+ pixel-value-1 pixel-value-2)) drand-16.895) (+ (-
pixel-value-8 (- (Sort-Loop pos-9 (Sort-Loop pos-9 pixel-value-9 Sum-Mode) Sum-Mode)
(+ pixel-value-4 drand-16.895))) drand-16.895))
```

Figure 6.11: Example Sort-Loop Solution for the Poker Hand Problem

The large size of the program tree coupled with duplicate nodes (16.895 for instance) is a good sign of bloat and redundancy which has hindered the understandability of the solution. Taking the effort to focus only on the two Sort-Loops reveals two notable characteristics for classifying Poker hands. First, `pos(9)` indicates that only the high values are iterated over (as values are sorted in ascending order). Second, the Sum-Mode operator is used. Suits range from 1–4 while cards range from 1–13. By targeting only high values, suite values (which do not influence if two cards are a pair) can be avoided at the expensive of ignoring low value cards. By using Sum-Mode, a large value is returned if no pair exists (all values are unique and, thus, summed together) and a low value is returned if a pair exists (maximum of 13 — a King).

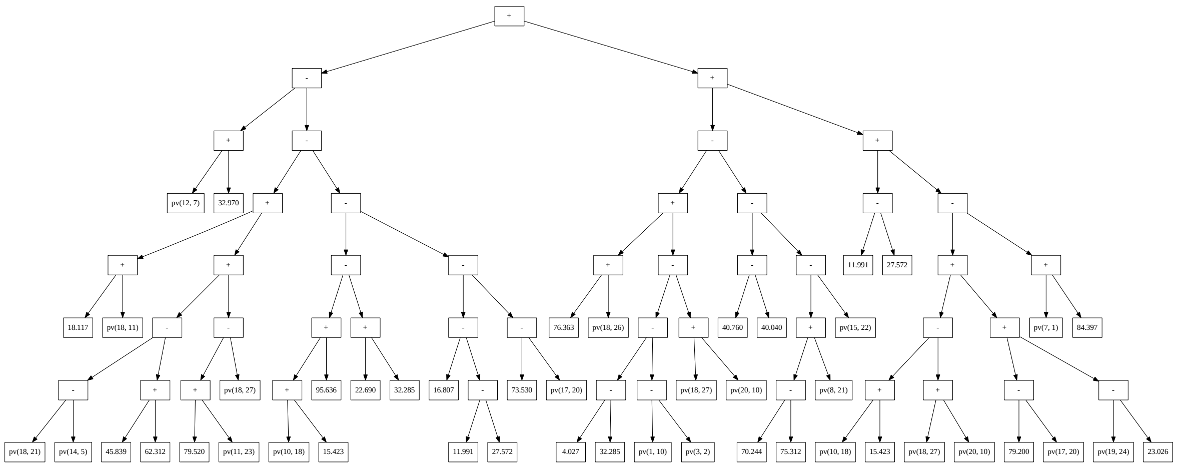
The inability to target all card values and exclude all suite values has reduced the classification performance of the Sort-Loop on the Poker Hard problem. Targeting high values means that low values are ignored. Large programs may have been learned in an attempt to identify low valued cards.

6.3 Further Discussions

It is evident that the GP systems with loops were able to reliably identify significant features for classification on a range of image problems. While Li's loop achieved good classification results, the identification of significant features is supported by numerous ill-positioned loop regions. Several loop regions to target one feature makes evolved classifiers difficult to interpret and it can be difficult to identify the bounds of the feature being targeted. The Bar-Loop produced good, simple, loop regions by evolving few overlaying or duplicate loop structures. The use of pixel-values also allowed the Bar-Loop to change which loop regions are being used based on pixel intensity between images. The analysed classifier also proved relatively simple. The program was human understandable, but it required more effort to analyse than the EB-Loop or the Step-Loop. The Step-Loop, although designed for 1D non-image data, demonstrated that it can identify good loop regions for the MNIST and Box problems. An analysis of a classifier for the Poker Hand problem revealed that it could develop a prefect solution for the task. The solution was simple and contained no program bloat.

The EB-Loop, while being able to produce a good human understandable solution for the Cell problem, could not find valid loop regions for the Face or Pedestrian problems, indicating that it is not powerful for complex features. The Seam-Loop produced good results for the Cell problem and could learn a simple program, but the unpredictability of where the Seam-Loop is going to iterate for different images clearly restricts how understandable learned classifiers are. The Seam-Loop works best for consistent images with horizontally aligned features. Vertically aligned features cannot be effectively targeted by vertical seams.

GP with no loops produces large programs made up of various pixel-value terminals which are not easily interpretable. A typical program evolved by GP with no loops for the MNIST dataset is displayed in Figure 6.12. Undoubtedly, the ability to visualise loop regions as well as produce small classifiers made up of expressive loop structures leads to learned programs that are simple to analyse and are human understandable. Where a problem is not well understood, GP with loops can evolve programs that provide insight into which features are important. Any unnecessary complexity in learned programs is expected to be manually identified and remedied which is an important topic for future work.



```
(+ (- (+ pixel-value-[row-12_col-7] drand-32.970) (- (+ (+ drand-18.117
pixel-value-[row-18_col-11]) (+ (- (- pixel-value-[row-18_col-21]
pixel-value-[row-14_col-5]) (+ drand-45.839 drand-62.312)) (- (+ drand-79.520
pixel-value-[row-11_col-23]) pixel-value-[row-18_col-27]))) (- (- (+ (+
pixel-value-[row-10_col-18] drand-15.423) drand-95.636) (+ drand-22.690
drand-32.285)) (- (- drand-16.807 (- drand-11.991 drand-27.572)) (- drand-73.530
pixel-value-[row-17_col-20]))))) (+ (- (+ (+ drand-76.363
pixel-value-[row-18_col-26]) (- (- (- drand-4.027 drand-32.285) (-
pixel-value-[row-1_col-10] pixel-value-[row-3_col-2])) (+
pixel-value-[row-18_col-27] pixel-value-[row-20_col-10]))) (- (- drand-40.760
drand-40.040) (- (+ (- drand-70.244 drand-75.312) pixel-value-[row-8_col-21])
pixel-value-[row-15_col-22]))) (+ (- drand-11.991 drand-27.572) (- (+ (- (+
pixel-value-[row-10_col-18] drand-15.423) (+ pixel-value-[row-18_col-27]
pixel-value-[row-20_col-10])) (+ (- drand-79.200 pixel-value-[row-17_col-20]) (-
pixel-value-[row-19_col-24] drand-23.026))) (+ pixel-value-[row-7_col-1]
drand-84.397)))))
```

Figure 6.12: Example No-Loop Solution for the MNIST Problem

Chapter 7

Conclusions and Future Work

This research has analysed the use of loop structures in GP for binary image and binary non-image classification tasks. Five new loop structures were developed: The Evolvable Body Loop (EB-Loop), Bar-Loop, Seam-Loop, Step-Loop, and Sort-Loop. Each loop presented different characteristics which were analysed against ten binary classification problems. For each problem, results were compared against a standard GP system with no loops and Li's proposed loop structures. Experiments found that GP with the proposed loops structures can more accurately differentiate between classes in both image and non-image problems than GP with no loops and often GP with Li's loops.

7.1 Project Conclusions

Of the five proposed loop structures, two were able to consistently outperform GP with no loops and GP with Li's loops. The Bar-Loop was able to achieve good classification accuracies for most image problems and the Step-Loop was able to achieve good classification accuracies for most non-image problems. The Bar-Loop and Step-Loop are, therefore, good all-round loop structures for classification tasks due to the simple nature of their iteration process.

The characteristics of the Seam-Loop and Sort-Loop allowed them to achieve the best results for select problems but poor results for others. The automatic targeting of seams by the Seam-Loop and the grouping of values of similar magnitude by the Sort-Loop is not adequate for most problems, but can be highly expressive for some. It is evident in this research that good loop structures are not overly flexible yet still expressive. Good loop structures must be as simple yet as expressive as possible. The EB-Loop, being the most complicated loop, could not reliably learn good solutions in the training period.

This research also found that GP with loops can be simpler and easier to understand. The ability to visualise loop regions provides insight into which features are targeted and considered important for classification. Further, the expressive nature of the loop structures allows small classifiers to be produced that can be manually interpreted with little effort.

No previous study had analysed the effect of loop structures for non-image classification tasks. This study has found that loops can notably increase classification accuracy over GP with no loops for non-image classification problems. The improvement of classification accuracy is more pronounced than for image classification tasks. Better results are evidenced by the Step-loop and Sort-Loop, which can target distant data values to find and exploit correlations to generate accurate classifiers.

7.2 Future Work

Despite the positive results for the application of loops in GP, there are several avenues and problems that must be addressed in the future:

1. **Learning Efficiency** GP with loop structures took multiple times longer to evolve than GP without loops. Work must be carried out to identify optimisations to minimise the difference in learning time between GP with loops and GP without loops to increase the utility of loops for industry applications. Possible avenues to explore may include new loop structures that are cheaper to evaluate, program simplification during evolution for GP with loops, restrictions on the number of loops, and effective parsimony pressure.

2. **Parsimony Pressure** Research into how best to apply parsimony pressure into the fitness function could help to further simplify learned programs to increase human understandability of learned solutions. Understanding whether parsimony pressure should take into account the number of loops, number of iterations, or nested loops may be helpful for reducing evolution time.

3. **Program Understandability** Although good results were produced by the proposed loop structures when evaluating program simplicity, more expressive loop structures may be able to further simplify learned classifiers.

4. **Mixing Loop Structures** No known prior work has mixed loop structures for classification tasks such that more than one loop structure exists in the function set. Mixing loop structures to achieve a collection of good all-round loops and domain specific loops may enable GP to produce even better classification results.

Bibliography

[1] A. Frank and A. Asuncion. (2010) UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences. [Online]. Available: http://archive.ics.uci.edu/ml

[2] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Published via `http://lulu.com`, 2008, (With contributions by J. R. Koza). [Online]. Available: http://www.gp-field-guide.org.uk

[3] X. Li, "Utilising restricted for-loops in genetic programming," Ph.D. dissertation, School of Computer Science and Information Technology, RMIT University, Feb. 2007.

[4] J. Larres, M. Zhang, and W. N. Browne, "Using unrestricted loops in genetic programming for image classification," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, Jul. 2010, pp. 1–8.

[5] X. Li and V. Ciesielski, "Using loops in genetic programming for a two class binary image classification problem," in *AI 2004: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, G. Webb and X. Yu, Eds. Springer Berlin / Heidelberg, 2005, vol. 3339, pp. 239–250. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30549-1_77

[6] N. J. Nilsson, "Introduction to machine learning: An early draft of a proposed textbook," 1996, http://ai.stanford.edu/~nilsson/mlbook.html.

[7] L. Goldschlager and A. Lister, *Computer Science A Modern Introduction*. Prentice/Hall International, 1982.

[8] E. Alpaydin, *Introduction to Machine Learning*, ser. Adaptive Computation and Machine Learning. MIT Press, 2004.

[9] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[10] T. Yu and C. Clark, "Recursion, lambda abstractions and genetic programming," *Cognitive Science Research Papers-University Of Birmingham CSRP*, pp. 26–30, 1998.

[11] Y. Qi, B. Wang, and L. Kang, "Genetic programming with simple loops," *Journal of Computer Science and Technology*, vol. 14, pp. 429–433, 1999, 10.1007/BF02948747. [Online]. Available: http://dx.doi.org/10.1007/BF02948747

[12] J. R. Finkel, *Using Genetic Programming to Evolve an Algorithm for Factoring Numbers*. Stanford Bookstore, 2003, pp. 52–60. [Online]. Available: http://www.genetic-programming.org/sp2003/Finkel.pdf

[13] T. Lai, *Discovery of Understandable Math Formulas Using Genetic Programming*. Stanford Bookstore, 2003, no. Rosen, pp. 118–127. [Online]. Available: http://www.genetic-programming.org/sp2003/Lai.pdf

[14] V. Ciesielski. and X. Li, "Experiments with explicit for-loops in genetic programming," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 1, Jun. 2004, pp. 494–501.

[15] X. Li and V. Ciesielski, "An analysis of explicit loops in genetic programming," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, Sep. 2005, pp. 2522–2529.

[16] G. Chen and M. Zhang, "Evolving while-loop structures in genetic programming for factorial and ant problems," in *AI 2005: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, S. Zhang and R. Jarvis, Eds. Springer Berlin / Heidelberg, 2005, vol. 3809, pp. 1079–1085. [Online]. Available: http://dx.doi.org/10.1007/11589990_144

[17] N. Pillay, "A genetic programming system for the induction of iterative solution algorithms to novice procedural programming problems," in *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, ser. SAICSIT '05. , Republic of South Africa: South African Institute for Computer Scientists and Information Technologists, 2005, pp. 66–77. [Online]. Available: http://portal.acm.org/citation.cfm?id=1145675.1145683

[18] G. Wijesinghe and V. Ciesielski, "Experiments with indexed for-loops in genetic programming," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ser. GECCO '08. New York, NY, USA: ACM, 2008, pp. 1347–1348. [Online]. Available: http://doi.acm.org/10.1145/1389095.1389357

[19] ——, "Evolving programs with parameters and loops," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, Jul. 2010, pp. 1–8.

[20] M. Wan, T. Weise, and K. Tang, "Novel loop structures and the evolution of mathematical algorithms," in *Genetic Programming*, ser. Lecture Notes in Computer Science, S. Silva, J. Foster, M. Nicolau, P. Machado, and M. Giacobini, Eds. Springer Berlin / Heidelberg, 2011, vol. 6621, pp. 49–60, 10.1007/978-3-642-20407-4_5. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20407-4_5

[21] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," in *ACM SIGGRAPH 2007 papers*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1275808.1276390

[22] O. Lezoray, A. Elmoataz, and H. Cardot, "A color object recognition scheme: Application to cellular sorting," *Machine Vision and Applications*, vol. 14, pp. 166–171, 2003.

[23] O. Lezoray. (2011) List of image databases. [Online]. Available: http://users.info.unicaen.fr/~lezoray/Databases.php

[24] Y. LeCun and C. Cortes. The MNIST database of handwritten digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[25] M. cbcl. (2000) Face data. [Online]. Available: http://cbcl.mit.edu/software-datasets/FaceData2.html

[26] D. M. Gavrila. (2006) Daimler pedestrian classification benchmark dataset. Intelligent Systems Lab Amsterdam, Faculty of Science, University of Amsterdam. [Online]. Available: http://www.gavrila.net/Research/Pedestrian_Detection/Daimler_Pedestrian_Benchmark_D/Daimler_Mono_Ped__Class__Bench/daimler_mono_ped__class__bench.html

[27] S. Hettich and S. D. Bay. (1999) The UCI KDD archive. University of California, Department of Information and Computer Science. Irvine, CA. [Online]. Available: http://kdd.ics.uci.edu

[28] S. Luke. (2011) ECJ 20: A java-based evolutionary computation research system. [Online]. Available: http://cs.gmu.edu/~eclab/projects/ecj/

[29] R. Lowry. (2011) Concepts & applications of inferential statistics. [Online]. Available: http://faculty.vassar.edu/lowry/webtext.html

[30] J. F. Box, "Guinness, gosset, fisher, and small samples," *Statistical Science*, vol. 2, pp. 45–52, 1987.

[31] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, November 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

Appendix A

Dataset Construction Algorithms

A.1 Box Dataset Construction

The process for constructing boxes in the Box-Vert and Box-Horz datasets is given in Listing A.1.

```
// Struct for defining a box image
typedef struct {
    int imageWidth;
    int imageHeight;
    int boxWidth;
    int boxHeight;
} ImageDetail;

int** createImage(ImageDetail *imageDetail, int row, int col) {
    // Declare the variables that will be used
    int **image;
    int i;
    int rowIndex;
    int colIndex;
    int rowStart = row - imageDetail->boxHeight/2;
    int colStart = col - imageDetail->boxWidth/2;
    int rowEnd = row + imageDetail->boxHeight/2;
    int colEnd = col + imageDetail->boxWidth/2;

    // Clamp the box start and end indicies
    if(rowStart < 0) rowStart = 0;
    if(colStart < 0) colStart = 0;
    if(rowEnd >= imageDetail->imageHeight) rowEnd = imageDetail->imageHeight - 1;
    if(colEnd >= imageDetail->imageWidth) colEnd = imageDetail->imageWidth - 1;

    // Initialise the image 2D array
    image = calloc(sizeof(int), imageDetail->imageHeight);
    for(i = 0; i < imageDetail->imageHeight; i++)
        image[i] = calloc(sizeof(int), imageDetail->imageWidth);

    // Add the box to the image
    for(rowIndex = rowStart; rowIndex <= rowEnd; rowIndex++) {
        for(colIndex = colStart; colIndex <= colEnd; colIndex++)
            image[rowIndex][colIndex] = 1;
    }

    return image;
}
```

Listing A.1: Box Dataset Construction