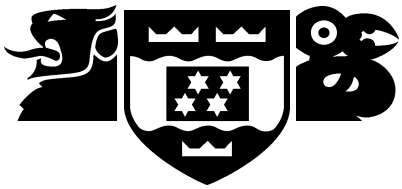
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



Computer Science

PO Box 600
Wellington
New Zealand

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

Improving Crossover in Genetic Programming for Image Recognition

Weijun (Normann) Lou

Supervisors:
Dr. Mengjie Zhang and Dr. Xiaoying (Sharon) Gao

Friday, 18 Feb 2005

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

Crossover operator is the predominant operator in most of Genetic Programming (GP) system. The empirical evidence shows that along with building blocks are constructed bigger and bigger as GP evolution proceeds, the crossover operator tends to disrupt those building blocks rather than preserve them. The traditional GP crossover primarily acts as macromutation. Looseness is used for representing how "sticky" two nodes at the two ends of the link should be. We use looseness to control the selection of crossover points in our extended Looseness Controlled Crossover (hereafter referred as eLCC) approach. The eLCC approach somehow eliminates the disruptive effect of the crossover operator explicitly. On the other hand, brood recombination crossover is the analogy of biological crossover, which mainly focus on modelling animal species breeding and preserves the homology in an implicit way. There exists a brood-diversity point for brood recombination crossover. The brood-diversity point stops further performance improvement while increasing brood size.

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisors, Dr. Mengjie Zhang and Dr. Xiaoying (Sharon) Gao, for directing me on right track and giving invaluable ideas, and for investigating a significant amount of their time to supervise the process, review the text and all other countless helps in this project.

I am further indebted to Mr. Will Smart, a PhD student in Victoria University of Wellington and the developer of the VGP (Victoria GP) package, for contributing ideas to the project and tutoring on how to efficiently use VGP package.

I also would like to acknowledge the course co-coordinator, Dr. Neil Leslie, for his effort on controlling the procedure in various stages and providing templates for writing the project proposal and project report.

I should finally give my thanks to my family — my wife, Maggie Lee and my son, Richard Lou for their logistic support and energy supplemented whenever I need.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Goals . . . 2
- 1.3 Contributions . . . 2
- 1.4 Structure . . . 3

2 Background **4**
- 2.1 Overview of Machine Learning . . . 4
  - 2.1.1 What is Machine Learning . . . 4
  - 2.1.2 Important Issues in Machine Learning . . . 4
  - 2.1.3 Search Strategies in Machine Learning . . . 5
- 2.2 Overview of Genetic Programming . . . 7
  - 2.2.1 The Primitives of GP . . . 7
  - 2.2.2 Fitness Function . . . 9
  - 2.2.3 Program Generation and Initialization a GP Population . . . 10
  - 2.2.4 Genetic Operators in GP . . . 10
  - 2.2.5 Selection . . . 11
  - 2.2.6 Main Parameters . . . 11
  - 2.2.7 The Basic GP Algorithm . . . 13
- 2.3 Related Work . . . 14
  - 2.3.1 Recent GP Approaches On Result Designation and Probability-based Fitness Function . . . 14
  - 2.3.2 Crossover — The Center of the Storm . . . 15
  - 2.3.3 Standard GP Crossover — A Type of Macromutation . . . 15
  - 2.3.4 Some Other Improved Crossover Operators Reported . . . 15
- 2.4 Clues from Biological Crossover . . . 16
- 2.5 Chapter Summary . . . 16

3 Tasks and Datasets **18**
- 3.1 *Shape* Dataset . . . 18
- 3.2 *Coin* Dataset . . . 19
- 3.3 *Texture* Dataset . . . 19
- 3.4 Summary on Characteristics of Datasets . . . 20

4 Basic Approach **21**
- 4.1 Chapter Goals . . . 21
- 4.2 Feature Selection, Extraction and Terminal Set . . . 21
  - 4.2.1 Mean as the 1st Order Moment . . . 22
  - 4.2.2 Standard Deviation as 2nd Order Moment . . . 22
  - 4.2.3 Skewness as 3rd Order Moment . . . 22

4.2.4 Kurtosis as 4th Order Moment . . . 23
4.3 Function Set . . . 23
4.4 Parameters Selected and Termination Criteria . . . 23
4.4.1 Parameters . . . 23
4.4.2 Termination Criteria . . . 24
4.5 Fitness Function . . . 25
4.6 Experimental Results . . . 25
4.6.1 Abbreviations . . . 25
4.6.2 Feature Combinations and Experimental Arrangement . . . 26
4.7 Analysis and Discussion . . . 27
4.7.1 Analysis for *Shape* Task . . . 27
4.7.2 Analysis for *Coin* Task . . . 27
4.7.3 Analysis for *Texture* Task . . . 28
4.8 Chapter Summary . . . 28

5 Headless Chicken Crossover **29**
5.1 Chapter Goals . . . 29
5.2 Revisit of Headless Chicken Crossover . . . 29
5.3 Experimental Results and Analysis . . . 30
5.4 Chapter Summary . . . 31

6 Looseness-controlled Crossover **33**
6.1 Introduction . . . 33
6.2 Chapter Goals . . . 33
6.3 Basic Idea . . . 33
6.4 Methodology . . . 35
6.4.1 Version 1 . . . 35
6.4.2 Version 2 and Version 3 . . . 35
6.5 Experimental Results . . . 36
6.6 Chapter Summary . . . 37

7 Extended Looseness-controlled Crossover **38**
7.1 Introduction . . . 38
7.2 Chapter Goals . . . 38
7.3 Theory Behind . . . 39
7.4 Methodology Extended . . . 39
7.5 Experimental Results and Analysis . . . 41
7.6 Chapter Summary . . . 41

8 Brood Recombination Crossover **43**
8.1 Introduction . . . 43
8.2 Chapter Goals . . . 44
8.3 Experimental Results and Analysis on fixed Brood Size . . . 44
8.3.1 Curves: Accuracy vs N and Generations vs N . . . 45
8.3.2 Brood-diversity Point . . . 46
8.3.3 Effect of Brood-diversity Point . . . 47
8.4 Experimental Results and Analysis on Varied Brood Size . . . 47
8.4.1 Four Functions We Used . . . 47
8.4.2 Results and Analysis . . . 48
8.5 Chapter Summary . . . 50

9 Conclusions **51**
9.1 Comprehensive Comparison 51
9.2 Conclusions 52
9.3 Future Work 53

List of Figures

2.1 Three fundamental genetic operators in GP: (a) Reproduction (b) Crossover (c) Mutation 11

3.1 Sample image in dataset *Shape* 18
3.2 Sample image in dataset *Coin* 19
3.3 Sample images in dataset *Shape* 20

4.1 Pixels distribution of images: (a) larger mean vs smaller mean; (b) larger standard deviation vs smaller standard deviation; (c) positive skewness vs negative skewness; (d) platykurtic vs leptokurtic. 22

5.1 The illustration of headless chicken crossover 30

6.1 The illustration of looseness-controlled crossover 34
6.2 The illustration of 3 versions for basic looseness-controlled crossover: (a) Version 1 (b) Version 2 (c) Version 3 35

7.1 The illustration of building block growing 39
7.2 The illustration of extended looseness-controlled crossover 40
7.3 The illustration of the way in that building blocks grow: (a) link-by-link (b) subtree-by-subtree 42

8.1 The illustration of brood recombination crossover 43
8.2 Improvement by brood recombination crossover for the three tasks (a) Accuracy (b) Generation 46
8.3 Dynamic N vs fixed N 49

List of Tables

3.1 Main characteristics of datasets 20

4.1 Main parameter values for applying basic GP approach on *shape*, *coin* and *texture* datasets 24
4.2 Results comparison of different feature combinations on the *shape*, *coin* and *texture* datasets using basic GP approach. 25

5.1 Results of headless chicken crossover and basic GP approach on *shape*, *coin* and *texture* datasets 31

6.1 Results of three versions of LCC and basic approach on the *shape*, *coin* and *texture* datasets 36

7.1 Results of three versions of eLCC and basic approach on *shape*, *coin* and *texture* datasets 41

8.1 Results of brood recombination crossover with fixed different brood size and basic approach on the *shape*, *coin* and *texture* datasets 45
8.2 Results comparison : basic GP vs brood recombination crossover with fixed N (N=4) vs brood recombination crossover with dynamic N on the *Shape*, *Coin* and *Texture* datasets 49

9.1 Best experimental results of each approach on the *shape*, *coin* and *texture* datasets 51

Chapter 1

Introduction

1.1 Motivation

Information exchange is more and more in form of graphics, "animated gif is worth one mega-word" [7]. The rapid increase on visualized information results in the high cost of manipulating information by human. This trend calls for automated programs. GP is one of the approaches to automatic programming.

GP is a relatively new and fast developing technique in evolutionary computation since 1992, when John R. Koza set GP as a individual part in Machine Learning by publishing the book "Genetic Programming: On the Programming of Computers by Natural Selection" and other three proceedings in various conference.

Search techniques are among the most important aspects of any machine learning system, because they define the manner in which the system moves through the space of candidate solutions. The crossover operator is the predominant search operator in most GP systems. The crossover operator applies the most of searching time, whereas mutation and reproduction operators apply the least of search time. GP's heavy use of crossover is more than just a preference, because it mimics the process of sexual mating. It is expected that good building blocks of a GP program are likely to be constructed and spread as they are exchanged among individuals. The remarkable amount of biological energy that goes into maintaining species and homologous crossover suggests that crossover may well be an effective search operator in population-based machine learning systems like GP [4]. It should be more effective than systems based on random transformations (such as mutations).

However, the randomness of crossover point selection in GP makes that the traditional GP crossover acts as macromutation. A lot of empirical evidence shows the traditional GP crossover operator is a disruptive force as well as a constructive force - putting building blocks together and than tearing them apart [4, 23, 22] and does not consider much about preserving the homology.

When candidate programs contain small good building blocks, these building blocks tend to be combined together by crossover, because the probability that the crossover points are selected inside the blocks is small; but when candidate programs contains large good building blocks, these building blocks tend to be broken by crossover, because the probability that the crossover points are selected inside the blocks is big. Along with the GP evolutionary process is more and more successful in assembling small building blocks into larger and larger blocks, the whole structure becomes more and more fragile because it is more prone to being broken up by subsequent crossover.

Additionally, principal constraints on biological crossover are noticed [4]:

- Biological crossover takes place only between members of the same species.
- Biological crossover occurs with remarkable attention to preservation of semantics.
- Biological crossover is homologous.

These constraints give us the clues that the traditional GP crossover could be improved in the way that eliminates randomness and adds more control over crossover operation.

1.2 Goals

The overall goal of the project is to investigate various new approaches to improve the conventional GP crossover operator so as to improve the performance for solving image recognition problems. These approaches will be examined and compared with the basic GP crossover on image recognition problems of increasing difficulty. Specifically, the following research goals will be investigated:

- to find out what features will be relatively suitable for specific tasks of multi-class object classification, which will be used as a baseline for comparison purpose in the basic GP approach.
- to investigate the crossover-is-macromutation theory by applying basic GP crossover and headless chicken crossover on image recognition.
- to design and develop new crossover operator to avoid good building blocks from being disrupted and preserve homology.
- to investigate one of the most promising GP crossover operator - brood recombination crossover in detail, justify the relation between brood size and the improvement of performance as well as the strategy of the dynamic brood size.

1.3 Contributions

In applying the basic GP approach, the fact that different features selected for the tasks with different difficulty level lead to the different classification performance reveals that the feature selection and extraction is still a very important aspect for image recognition problem. Although more ”powerful” features are extracted as inputs to GP system, the unsatisfactory results on hard task by basic GP approach give the need for improving GP system in other aspects, rather than from feature selection and extraction. The project has the following major contributions:

- The investigation on hill climbing crossover — headless chicken crossover on image recognition domain adds more weights to the crossover-is-macromutation theory. The investigation also predicts the directions for further efforts on improving traditional GP crossover, and turns on the light of stagnation of GP runs, that occurs as the result of what is referred as ”bloat” — the exponential growth of GP ”introns”.

- The new crossover operator — extended Looseness Control Crossover (eLCC) is introduced, which directly makes effort on the selection of crossover points and the preservation of good building blocks. Giving more control over the selection of crossover points, eLCC finds a new way to reduce the disruptive effect of crossover. By proposing the concept of link looseness or stickiness, eLCC also finds a way to identify good building blocks from other parts in a tree.
- The investigation on brood recombination crossover on image recognition problem makes the trends of improving crossover by homologous crossover more persuadable. Also, this investigation issues a new concept — brood-diversity point, which is believed to be an important aspect to reveal the relationship between the brood size and the improvement of performance.

1.4 Structure

In this report, Chapter 2 briefly reviews GP and machine learning theory, summarises research problem concerning GP. We describe the three tasks of increasing difficulty and the corresponding three datasets used in this project in details in Chapter 3. In Chapter 4, we apply the basic GP approach to the three tasks. In details, we assigns values to the various experimental parameters for three tasks respectively, gives termination criteria, explains the details of the features selected and extracted for each task, provides the results based on different combination of features, summarizes the most suitable features which will be used in the chapters that follow. In Chapter 5, headless chicken crossover is investigated on the three tasks, we discuss its advances and shortages based on comparison with basic GP approach. Chapter 6 introduces the basic concept of link looseness and how we attempt to control crossover operation by it, i.e. the Looseness Control Crossover (hereafter referred as LCC). We describes the methodology we used, gives the experimental results of this first attempt in this direction and analyses further possible improvement in this direction. In Chapter 7, we review and justify the hypothesis we made in LCC, extend LCC to eLCC, improve the methodology, give the experimental results and analyse the results. in Chapter 8, we investigate one of the most promising crossover operator — Brood Recombination, we also compare the results with basic approach and analyse some important and interesting points in this approach. The last chapter comprehensively compares the four approaches we investigated in this project, draws our final conclusions, and indicates future work.

Chapter 2

Background

2.1 Overview of Machine Learning

2.1.1 What is Machine Learning

Since the 1950, computer scientists have made varying degrees of success to give computers the ability to learn. The most contemporary definition of machine learning is the study of computer algorithms that improve automatically through experience [12]. All machine learning systems try to solve, at the highest level, a similar task — how to learn from its experience of the environment. The learning method has to be told precisely *how to learn* rather than *how to perform* the task required in a specific problem [5]. Machine learning is a process that defines the environment and the techniques for letting the machine study the environment by training and evaluate the learning by testing. This process begins with the identification of learning domain and ends with evaluating and using the results of the learning.

2.1.2 Important Issues in Machine Learning

The key parts of machine learning process are the ”learning domain”, the ”training set”, the ”learning system”, ”testing” the results and ”monitoring” the learning procedure [22, 4].

A learning domain is set of problems or facts where the ”features” of the domain can be identified, selected and extracted. For example, for image object recognition problems, the features, such as brightness, contrast, texture or appearance, could be measured by the images pixel statistic data and the various calculations on the data.

Machine learning occurs by training on examples, each of which contains data representing one instance of relationship between the chosen features and a result or results. For example, in bank loan decision system, two examples can be:

ID	Income	Marriage Status	Num. of Children	Credit Record	Result
56	$60,000	Married	2	Excellent	Approved
66	$45,000	Single	0	Bad	Conditional Approval

According to the usage of the examples in a machine learning system, the examples can be splitted in three sets: training set, test set and validation set.

The collection of examples from which the learning knowledge or rules are induced is called the training set. A machine learning system goes through the training set and attempts to learn from the examples.

The collection of examples for measuring the quality of the learnt knowledge or rules is called the test set. The examples in test set are from the same domain the system trained upon and is never used during the system training time.

The ability of a system to predict the results or outputs of the test set is often referred to as ”generalization” [4]. Generalization occurs when learning knowledge or rule also is valid on test set. Good learning system would learn the true relationship between the features (inputs) and results (outputs) and ignore the noise existed in training set. However, many machine learning systems can and do continue to improve its fitness to the training set by leaning the noise unique to the training example. This results in that the knowledge or rules learnt highly fits into the training set but has very bad performance in test set. The problem is called overfitting or overtraining. In causing overfitting, three factors may play a part:

- complexity of the learnt solution: The simpler the model being learnt, the higher the probability of being generalized.
- amount of time spent on training: too much time spending on training tends to make learnt knowledge overfit the training set.
- size of training set: the smaller the training set is, the small the reliability of prediction on test set will be, and the more difficult to get good generalization

To solve the overfitting problem, another collection of examples is used for monitoring and terminating the training procedure. This set is called validation set, it is independent from training set and test set and is not directly used in training.

2.1.3 Search Strategies in Machine Learning

A machine learning system targets at find a final solution that can be translated into the knowledge or rules modelled. The candidate solutions comprise the search space. This search space is usually huge for non-trivial problems. Thus, each learning system must define a search scheme operator which decides how and in what order the system will search through a limited portion of huge search space, i.e. which candidate solution will be evaluated first, which next and when to stop the search [4]. There are different types of search operators, and here we only have a quick look at four of them: blind search, hill climbing search, simulated annealing search and beam search.

Blind Search

A system with blind search strategy goes through the search space and picking up a candidate solution without knowledge of the search space and the results from the previous step in the search. The search is directed by no heuristics.

Blind search usually moves a tree structure to represent the search space, in which each node represents a candidate solution and the edges represent the ”jumps” through the search space among nodes [4].

Blind search traverses the tree either in breadth-first or depth-first manner. Breadth-first search means searching each level of the tree until a good solution is found or the whole tree is finished. Depth-first search means searching the maximum depth of the tree down the first path dictated by the tree. If it reaches a leaf, it backtracks up the tree until it sees a branch

which has not been visited yet. It searches down this unvisited branch to the maximum depth. This exploration continues until an acceptable solution is found or the whole tree is completed.

Both breadth-first and depth-first search are exhaustive type search, and almost all interesting learning domains in machine learning have far too large search spaces to using the blind search.

Hill Climbing Search

Hill climbing search have one candidate solution evaluated and transformed to make new candidate solution, the new solution is evaluated. It is kept if it is better than the old one. Otherwise, it is discarded and the old one is kept and transformed again to make another candidate solution, the newly formed solution is evaluated, kept or discarded until a termination criterion is met [4].

The key of hill climbing search is that the system always currently keeps best solution seen so far. It guarantees that all the solutions previously kept are worse than the one currently kept, and the one next kept will be better than the one currently kept. Also, it is usually easier to form a better solution (which is the one next kept) from a good solution (which is the one currently kept) than from a worse solution (which is the one previously kept). Thus, hill climbing probably limit its search on a small portion of search space of all candidate solutions. Hill climbing therefore is not exhaustive type of search.

However, this search strategy suffers from the problem so called ”local maxima” (or ”local minima” if switching our view to gradient descent — targeting at minimum cost) [16], because it can stunk on a solution which is locally best.

Simulated Annealing Search

Simulated annealing is quite similar to hill climbing, except it avoids local maxima (minima) problem by allowing the search to continue after it reaches the local peak (in case of hill climbing) or the local bottom (in case of gradient descent). We can view our tasks as getting a ping-pong ball into the deepest crevice in a bumpy surface [16]. Once the ball gets into a local pit, we need to shake the surface so as to bounce the ball out of the pit. The key lies in that we should shake the surface hard enough for bouncing the ball out of the local minima but not enough for bouncing the ball out of the globally deepest crevice [16].

Keeping the imagination above in mind, when the search moves to the next solution, it is not necessarily a best move. It allows random moves. That is, if the move finds a better solution, it is always acceptable. Otherwise, depending on the stage which the search is in and how ”bad” the move is, it is accepted or discarded. The ”bad” moves are more likely to be allowed at the start and more likely to be refused at the end. The likelihood of being accepted is decreased along with the search proceeds or if ”badness” of the moves gets bigger.

Beam Search

Unlike blind search and hill climbing which are single point-to-point searches in the search space, beam search maintains a population of search points. In a beam search, some ”evaluation metric” is used to select out a certain number of the most promising solutions, which

comprise the so-called "beam", for further transformation. All others solutions are discarded. In another word, a beam search limits the points it can find in search space to all possible transformations that result from applying the search operators to the individuals in the beam [4]. We will address this search strategy in a little detail in Section 2.2

2.2 Overview of Genetic Programming

Genetic programming (GP) is the automated learning of computer programs [4], which can be solutions to problems.

Based on the concept of biologically inspired operations (crossover, mutation and reproduction, etc) and the principles of Darwinian natural selection (survival of the fittest), GP iteratively transforms a population of randomly generated computer programs into a new generation of programs by applying these operators to individuals selected according to their fitness. The core of Darwinian natural selection reveals that "The fittest survive", GP follows the same theory, i.e. the individual program with the best fitness are selected to participate those operations and form the next population. The iteration of transformation goes on until the termination criteria are satisfied and the final single best program probably solves the problems under the problem domain.

GP systems can be designed for solving problems by finding the best solution, which can be presented in many forms, such as linear string, tree structure or graph. GP dealing with tree presentation of program is called tree-based GP system, we will only address tree-based GP system in this project.

Most computer programs usually can be presented as a tree structure, in which the root and the internal nodes are called functions and the leaves are called terminals in GP. Functions are categorized as general functions and domain specific functions, whereas terminals are usually the attributes/features of a problem and some random selected real numbers.

The fitness function is used to select out a population of the most promising solutions for further evolution, thereby, GP employs a beam search algorithm form in term of machine learning. GP retains a population of candidate solutions that is smaller than the set of all possible solutions, in which the evaluation metric for the beam is the "fitness function" and the beam is referred as the "population" [17]. Machine learning systems have search operators that regulate both the contents and ordering of the beam. In GP, the contents of the beam are regulated by the genetic operators: reproduction, crossover and mutation, and the ordering is regulated by fitness-based selection.

2.2.1 The Primitives of GP

The functions and terminals are the primitives with which a program in GP is built. In a tree presentation of programs, terminals give valued to the system, whereas functions process those value provided by terminals. Functions and terminals are regarded as nodes of the tree presentation of programs.

The Terminal Set

Definition: *The terminal set is comprised of the inputs to the GP program, the constants supplied to the GP program, and the zero-argument functions with side-effects executed by the GP program.* [4]

The terminals, by name, terminate a branch of a tree structure in GP. They are inputs to the program, constants or functions without argument, and they return an actual numeric value.

In Section 2.1, we spoke the learning domain and the features that is selected, extracted to represent the problems under the learning domain, the selected features become the input data for system to be trained and tested. In the same way, the selected features are put into training and test sets, and then are input into terminal set.

The terminal set also includes constants with is randomly select in a range at the beginning of the run and keep unchanged. These constants in terminal can construct other constants within programs by arithmetic function in GP.

The Function Set

Definition: *The function set is composed of the statements, operators and functions available to the GP system.* [4]

The function can be selected in a very broad range and be application-specific and learning domain specific. In general, it may use any programming construct that is available in any programming language. For example, they can be arithmetic functions (PLUS, MINUS, MULTIPLY or DIVIDE), Boolean functions (AND, OR, NOT or XOR), standard triangle functions (SIN, COS, TAN) assignment statement (ASSIGN X 3), conditional statement (IF...THEN...ELSE or SWITCH) or loop statement (WHILE...DO, FOR...DO), etc.

Selection of Terminal and Function Set

The selection of terminals and functions, sometimes, can be critical to the success to solve problems by GP. In general, the abstraction of terminals and functions might be too high or too low to allow for a sensible evolution of individuals [22]. If two low, the system will find the task too complicated to move in the various directions necessary to improve a solution. If two high, the task can not be broken down appropriately.

GP have a significant ability to form new constants by applying function to terminal constants within a program, we don't need to assign the same number of constants to all nodes that labels used for constant. For example, if a program has 120 nodes in its tree presentation, among which 40 nodes are labelled for functions and other 80 are labelled for constants. We only need to assign constants to the terminal set with a number much less than 80. The more the constants assigned, the bigger the search space.

For function set, if it contains too less or too simple functions, it will probably not solve many very interesting problems. On the other hand, if a function set is too large, it will increasing the difficulty for finding the solution because of the increased search space.

A good start point for function set might be some simple combinations of arithmetic and logic operations.

While selecting functions for function set, the closure property must be considered. The closure property of function set is that each function should be able to accept gracefully all values it might receive as input. For example, the normal division can not handle zero as the divisor, so when selecting division as one member of function set, it has to be changed into sort of "protected" division which can also handle zero as divisor by explicitly coding this ability. The closure property of terminal set needs to be considered in the same way.

In practice, GP has surprising ability to take simple functions and create what it needs by combining them. In fact, those sophisticated functions are often ignored during GP evolution. It is advised one should not spend too much time designing specific functions and terminals that seems perfectly attuned to the problem.

The following heuristic guidelines for design terminal set and function set are given [26]:

- Sufficiency and closure.
- As small as possible, to shorten search time
- For relative easy problems, use standard functions
- For relative difficult problems, add more customised functions.
- Have functions permitting non-linear behaviour, such as conditional branch statements (eg. IF...THEN...ELSE), triangle functions (eg. SIN, COS) and exponent factions (eg. X^Y) for hard tasks.
- If the problem is expected to be solved by smooth curves, functions that can generate smooth curves should be used, and vice versa.

2.2.2 Fitness Function

Purpose of Fitness Function

Definition: *Fitness is the measure used by GP during simulated evolution of how well a program has leaned to predict the output(s) from the input(s) — that is, the features of the learning domain*. [4]

The examples which contains the features (inputs) and results (output) in term of machine learning are called fitness cases or patterns. As we mentioned earlier in this section, the GP's evaluation metric is called a fitness function, which is used to evaluate candidate solutions upon fitness cases in training set.

Fitness of a program generated by the evolutionary process is evaluated according to the fitness function defined for the GP system.

Fitness Function Design

Fitness function is designed to give feedback on which individuals should be selected to participate in the transformation to form the next population and which individuals should be discarded. So fitness controls the procedure of evolutionary process.

The fitness function should be designed to give graded and continuous feedback about how well a program performs on the training set. Usually, the fitness function is application-specific. For example, for image matching, the fitness of a program can be the number of matched pixels; for classification task, fitness of a program can be the number of correctly classified examples, error rate or classification accuracy; for GP-controlled agent in a betting gram, fitness can be the amount of money won. Fitness function is the core of any fitness-based selection system, like GP.

For complex GP system, there might be more than one goal, instead, multiple goals. Accordingly, the design of the fitness function needs to consider those multiple goals. For

example, there are two main goals in object detection system: high detection rate or recall and low false alarm rate or high precision. A function f(recall, precision), which combines the two goals, need to be considered correspondingly.

2.2.3 Program Generation and Initialization a GP Population

One of the important parameter of a GP run is the maximal program size permitted. In tree-based GP, it is usually expressed by maximum depth of a tree, where depth of node is defined as the minimal number of nodes that must be traversed to get from the root node to the selected node.

There are several ways of generation programs and initialising GP population: *full, grow* and *ramped half-and-half*.

By *full* generation method, functions are assigned to the nodes of tree until a given depth is reached, then the terminals are assigned to form the leaf nodes. This method generates full, entirely balanced trees. However it results in a uniform set of structures in the initial population.

By *grow* generation method, nodes are selected from either functions or terminals. When a terminal is selected, this branch with the terminal is terminated and the construction goes on to the next non-terminal branch in the tree.

By *ramped half-and-half* method, the population is divided equally among individuals to be initialized with trees having depth 2, 3, ..., n, supposing the maximum depth parameter is n. For each depth group, half of the trees are generated in *full* method and the other half in *grow* method. *Ramped half-and-half* method adds the diversity to programs in GP populations which is very valuable.

2.2.4 Genetic Operators in GP

The initial population usually has very bad fitness. Evolution proceeds by transforming the initial population by the use of genetic operators. These operators are search operators in term of machine learning. The most fundamental genetic operators in GP are *reproduction*, *crossover* and *mutation*.

Reproduction simply copies a selected program from the current generation to next generation, as show in Figure 2.1 (a). This operator allows good program to survive.

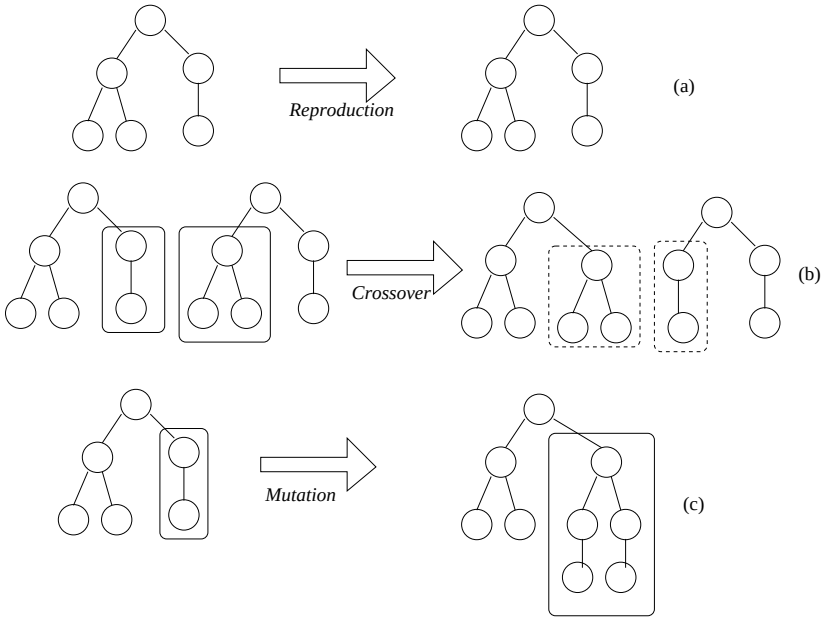


Figure 2.1: Three fundamental genetic operators in GP: (a) Reproduction (b) Crossover (c) Mutation

By *crossover* operation, the genetic material of two selected parent programs is combined by swapping a subtree of one parent with a subtree of the other, then the new two programs are put into next generation, as shown in Figure 2.1 (b). This operator tries to mimic the biology sexual mating behaviour to take advantage of different selected programs with a population, and integrate the useful attributes from them.

Mutation operates on a single selected program. As shown in Figure 2.1 (c), a random subtree of the program is replaced by another subtree, which is generated by one of the *full*, *grow* and *ramped half-and-half* methods. This operator adds the valuable diversity to the population.

2.2.5 Selection

The most GP systems employ one of the two selection methods for choosing individuals to participating to the next generation: proportional selection and tournament selection.

The proportional selection specifies probabilities for individuals to be given a chance to pass offspring into the next generation. The probability is proportionally mapping with the individual's fitness.

The tournament selection is based on competition within only a subset of the population rather than the whole population. The size of the subset is called tournament size. The individuals in the subset are randomly selected from the whole population. The winner of the tournament participates the genetic operations (mainly mutation and crossover), and its offspring replaces the loser.

2.2.6 Main Parameters

There are very important parameters to be chosen appropriated before starting to run a GP system. Usually, bad choice of important parameters leads to very slow convergence, and even can not find a satisfied solution. These parameters are [26]:

- population size: the number of individuals in population.
- maximum number of generations: the maximum index of generation for the evolutionary process to be terminating.
- reproduction rate: the percentage of individuals in next generation that are to be produced by reproduction.
- crossover rate: the percentage of individuals in next generation that are to be produced by crossover.
- mutation rate: the percentage of individuals in next generation that are to be produced by mutation.
- maximum program size: the limitation on how big individual programs in a population can be.

Population Size

Population size has two sides of influence in GP run. On one hand, bigger populations take more time for transforming a generation. On the other hand, because bigger populations have more genetic diversity and make more areas of search space being explored, it tends to reduce the number of evaluations for finding a solution. Banzhaf [2] claimed that in range of 10 to 1,000,000, increasing population size gives a nearly linear improvement in performance of the system. The selection of population size is also related to number of training examples. In general, the more the training examples, the larger the population size required. A common setting is 500 supposed by Koza.

Maximum Number of Generations

The argument on maximum number of generations is boiling in GP community. One point of view is that 50 generations are enough, because it is believed that what has not been achieved in a GP run in 50 generations will never be achieved. Another view supports that interesting evolution can be delayed to generation 1000, even more. Some researchers have made their efforts on the study of how to combine population size with maximum number of generations, i.e. small population size with big maximum number of generations vs. large population size with small maximum number of generations and how to trade off. Some researchers set the strategy: Start with large maximum number of generations and make early stop when constructive crossover falls to below 10% of all crossover events. A well recognized starting point for maximum number of generations is between 50 - 100.

Reproduction Rate vs. Crossover Rate vs. Mutation Rate

If a basic GP system uses only reproduction, crossover and mutation operator, then the sum of reproduction rate, crossover rate and mutation rate should be 100% according to their definitions. The discussion on how to allocate the value amongst the three operators has been hot since early 1990's.

The goal of reproduction is to allows the best individuals in a population survives, crossover is for combining the advantage of different selected individuals in a population

and establish, whereas mutation is to keep the diversity of a population. Form their importance of view, crossover and mutation are more critical than reproduction in simulating the biological evolution.

Comparing the importance between crossover and mutation, Koza [10] proposed that mutation is useless in 1992. Wolfgang Banzhaf, Frank D. Francone and Peter Nordin [3] found that a balance of crossover and mutation (50% vs. 50%) led to better results for difficult problems in 1996. Harries argued that mutation is more useful than crossover in 1997, and Matthias Fuchs, Dirk Fuchs and Marc Fuchs [6] claimed in 1998 that mutation is more useful when they used GP to solve the problems of combinatory logic. However, the most up-to-date common sense is that crossover plays a dominant role, especially in the early stage of the evolutionary process. In most GP system, GP crossover has been used for the claim that GP search is more effective than system based on random transformations/mutation, like annealing search. So a common setting is high rate of crossover and low rate of mutation; when the acceptable results cannot be obtained, mutation rate can be increased.

Maximum Program Size

The program size can be limited in two ways: maximum number of nodes allowed in an individual program and maximum depth of tree structure. The definition for depth of a node is referred to Section 2.2.3. The most commonly used one in tree-based GP is the maximum depth.

GP system usually uses different values for initial generation and the further evolved generation. Initial maximum depth is defined as the maximum depth of randomly generated programs in the initial population. Maximum depth is defined as the maximum depth permitted for programs resulting from crossover and mutation operations [26]

Typical initial program size should be very small to allow the system build up good individuals piece by piece. Giving small value to initial program size eases the process that building blocks grow from small size to large size. For difficult problems, the initial program size needs to be tried with larger value if no acceptable results can be obtained. This will allows the GP system to start with pre-matured complexity and avoids early local minima. In general, the initial program size should be less 10 and 4 - 6 is a good starting point [22].

when giving a value to maximum program size, the general rule is that this size should be large enough so as that the GP system has a sufficient search space. Maximum program size is task specific, and it should become bigger along with increasing difficulty of tasks. Starting point should less 10 for easy tasks and up to 25 or more for hard tasks.

2.2.7 The Basic GP Algorithm

Holding all the definitions of main aspects and key parameters of GP in hand, now we can describe the basic GP algorithm by steps that a GP system goes through during a run as follow [8]:

- Initialize a population of individuals programs composed of function set and terminal set at generation 0
- Go through evolution loop, which iteratively performs the following sub-steps:

1. Measure the fitness of each individual program in the population against the training set composed of fitness cases
2. Select 1 or 2 individual programs based on best fitness to participate new individual program(s) generation
3. Transform individual program(s) and insert the transformed individual program(s) to new population by applying one of the following operations:
   (a) Reproduction: duplication of one selected individual program, see Section 2.2.4
   (b) Crossover: recombination of randomly selected parts of two selected individual programs, see Section 2.2.4
   (c) mutation: random mutation of a randomly selected parts of one selected individual program, see Section 2.2.4
4. Designate the best individual program as execution result if one of the termination criteria is satisfied, otherwise goes to step 2

2.3 Related Work

The parameters for controlling the run can be optimized for a specific task, if we run many different experiments with different combination of different parameters' values. Then the optimized parameters usually improve the performance of the GP system for that specific problem. But as the difficulties of the tasks increase, the performance can not longer meet the requirement only by optimized parameters, new improved approaches are called for [24].

2.3.1 Recent GP Approaches On Result Designation and Probability-based Fitness Function

The researchers have got great consequence on investigation of improving GP approaches. Amongst them, SCBD (Slotted dynamic class boundary determination) [18], CDCBD (centered dynamic class boundary determination) [18] and probability based GP [19] are very recent reported approaches on multiple classification task.

The output of a numeric expression GP classifier is a numeric value, which should be further translated into a class label. Traditionally, this translation is made by falling each output in the region separated by man-made predefined thresholds/boundaries. However, firstly, it is very difficult to find the appropriate threshold/boundaries value for multiple classification problems and involve some hand-craft work; secondly, value inappropriately set will often make long training time and poor effectiveness, consequently, resulting complicated computer programs.

Both CDCBD and SCBD approaches use new translation strategies to allow each genetic program to use a set of dynamically determined class thresholds, the value of these thresholds will be automatically determined and regulated along with the evolutionary process. Probability-based GP is an even more creative approach. It evolves not only the translation strategy but also fitness measurement.

However, these improved approaches did not consider the disruption of good "building block" by crossover operator, though they have a great step ahead for overcoming the shortage due to the static man-made predefined thresholds and fitness measurement.

2.3.2 Crossover — The Center of the Storm

Crossover is the predominant search operator in most GP systems [25] [22] [4]. In Genetic Algorithm (GA), it is proved that there are good building blocks which can improve the fitness of individuals. Theoretically, GP should work faster than systems which just base on mutations, because good building blocks can grow larger and larger to form better individuals by crossover operation. However, the randomness of crossover point selection increases the risk that good building blocks are disrupted, particularly in the later stage of evolutionary process.

The famous Gedanken Experiment has a look at the probability of disruption issue of crossover. It is concluded that there might be two different phases in a GP run: in the first phase, the building blocks are relatively small in early stage of evolution process, the probability of the building blocks being disrupted by crossover are small too, where the evolution completely depends on fitness; in the second phase, the building blocks are large enough in later stage of evolution process, the probability of building blocks being disrupted are large, too, where the evolution depends on both fitness and individual structure. As an extreme example, when a final best program is evolved, the whole program is probably a perfect building block, if we apply the crossover operator to this program, then the probability of building block being disrupted is 100%.

Additionally, the effect of crossover on the relative fitness of parents and their offspring was measured in 1995 on tree-base GP system[13] [14]. The measurement shows that the fitness of the children is less than half the fitness of the parents in about 75% of all crossover events. It is concluded that crossover has an overwhelmingly negative effect on the fitness of the offspring of the crossover.

2.3.3 Standard GP Crossover — A Type of Macromutation

Lang [11] lunched his argument that traditional GP crossover did not perform nearly as well as a macromutation operator in 1995. His whimsically dubbed crossover operator — headless chicken crossover is a macromutation operator and in the form of hill climbing search. Lang claimed that headless chick crossover was much better than traditional GP operator. But his conclusion is only based on fairly easy task (Boolean 3-multiplexer problem) and therefore questionable. Because every machine learning technique has a bias - a tendency to perform better on certain types of problems than on others [4], whereas Boolean 3-multiplexer problem has no local minima, one can conclude that the bias of hill climbing algorithm is particularly well for Boolean 3-multiplexer problem. The mutate-simulated annealing and hill climbing crossover should be applied to more suites of problems to extend this theory.

2.3.4 Some Other Improved Crossover Operators Reported

The two most promising approaches developed recently are brood recombination by Tackett [20] and smart crossover by Iba and Garis [9]. Tackett create a ”brood” each time crossover was performed to model the observed and only two best children survives and all others die. This is an analogy of the observed fact that many animal species produce offspring far more than the expected to live. This approach will be addressed in Chapter 8. In general, this approach focus on making better crossover, but it did not consider intelligent selection of the crossover point, though it dose preserve the building block and homology in

a sort of hidden way. Iba and Garis add intelligence to the crossover operator, which becomes "smart crossover". Smart crossover evaluates performance for subtrees and selects the crossover point in the individual program based on these evaluations. This approach focus on determining the crossover points and constructing good building block, but it did not consider the homology. Eg. assume that a good building block is built up by smart crossover, it might be moved into a new individual that does not use it as a good building block, in another word, it might be moved into an inhospitable context.

The authors of [4] proposed a homologous crossover operator (unnamed) in their works. This crossover operator measures the structural similarity functional similarity of each subtrees by edit distances, between two programs. The two measurements are interpreted as probabilities and then use it to select crossover points. The building block hypothesis is still weak today, this is the main weakness of the proposal, especially when the proposal is lack of support of experimental results.

2.4 Clues from Biological Crossover

The fact that the traditional GP crossover reduces the fitness of offspring substantially comparing to their parents, stands in stark contrast to biological crossover. There are very interesting facts in biology that are sighted by GP researchers and this project will seek/investigate improving crossover by following some of these clues:

- In biological crossover, the gene is swapped only with the corresponding gene in the other parent. For example, the skin color gene never swapped with tallness. Furthermore, the different alleles in the swapped genes make only minor changes on the basic function after crossover due to homology. However, there is no requirement either on the relation between the two swapped context nor on the similarity between two factions that two swapped subtrees fulfil due to randomness of crossover points selected.
- The biology crossover swaps only between similar genetic material. This keeps gene function with only minor change. For example, the hair color gene crossed over in the new chromosome still decides hair color rather than other aspects of the offspring and has relatively independent function from other gene. However, once a good build block is constructed by GP crossover, it would be likely disrupted by crossover rather than preserved.
- The different species rarely mate, eg. a butterfly rarely crossover its DNA with a dragon fly. But, in GP system, it is very hard to say that the individual programs in a population are in the same specie — they are just randomly initialized.

These hints suggest there might still be room for improving the traditional GP operator.

2.5 Chapter Summary

GP, as another machine leaning system, represents a problem as the set of all possible computer programs, uses the genetic operators: reproduction, crossover, mutation to transform candidate solution into new candidate solutions in its evolutionary process. GP uses a beam search, where the population size constitutes the size of the beam and the fitness

function serves as the evaluation metric. Empirical evidence shows applying crossover to program structures often destroy valuable ”building blocks” of sub-programs, randomness in crossover point selection reduces the ability to mimic the biological crossover. Crossover operator becomes the center of the storm. Although there are lots of newly developed crossover operators, there are still rooms worthy to make further effort on crossover. New approaches on crossover operators need to be developed to minimize its disruptive effect as well as preserve the homology.

Chapter 3

Tasks and Datasets

To compare the performance of various approaches investigated in this project, we design three tasks under image recognition problem domain. They are all multi-class object classification problems but varied in difficulty.

3.1 *Shape* Dataset

The first task is to classify a group of shapes on images, which are computer generated with a certain level of noise. Correspondingly, the first set of images is generated to give different shapes with some certain noise in each pixel against a noisy background, as shown Figure 3.1. Besides the shape difference, the different noise setting also makes brightness and contrast for each class to be different too. The three classes are named as *black circle*, *grey square* and *white circle*. There are 120 objects cutouts from these images distributed evenly among the 3 classes, i.e. there are 40 objects cutouts for each class. This set is referred as *shape* for the convenience of presentation.

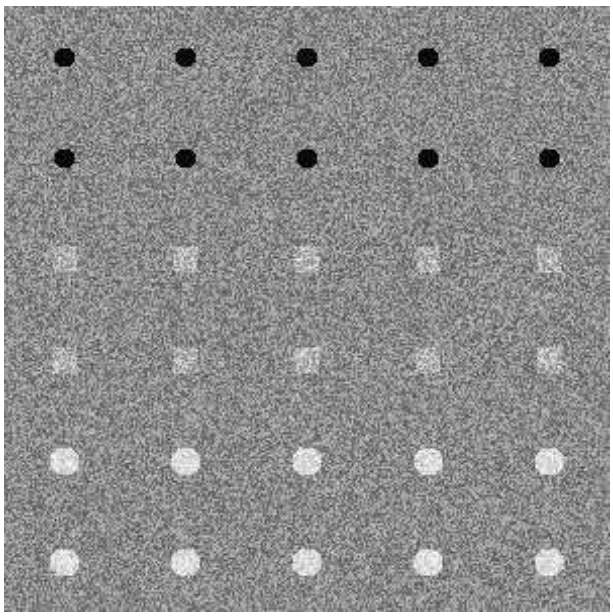


Figure 3.1: Sample image in dataset *Shape*

.

3.2 *Coin* Dataset

The second task is to classify coins appeared in different sides (heads and tails) in images, which are photographed with relatively large noise. Accordingly, the second set of images contains a bunch of New Zealand 10-cents coins against a highly noisy background, as shown Figure 3.2. They are placed in different orientations and appeared in different sides, i.e. head and tail. Thus, if the highly cluttered background is treated as a class, totally 3 classes are formed:*head*, *tail* and *background*. There are totally 500 objects cutouts from these images are made. Among the 500 cutouts, there are 160 cutouts for *head*, 160 cutouts for *tail* and 180 cutouts for *background* respectively. This set is referred as *coin*.

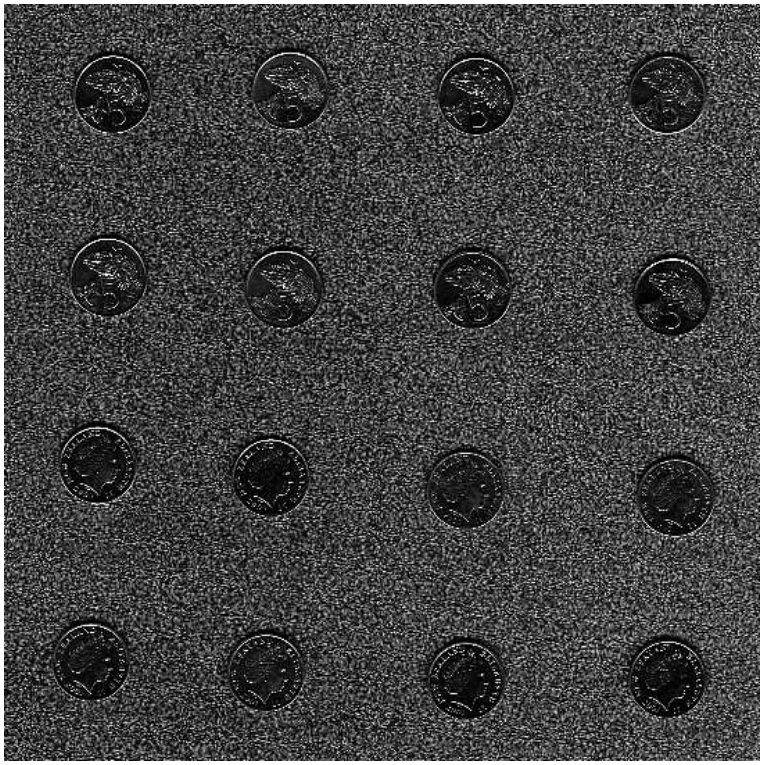


Figure 3.2: Sample image in dataset *Coin*

.

3.3 *Texture* Dataset

The third one to classify different texture, which are also photographed and looked very similar. So, the third set of images is a series of 4 similar textures images, which are taken by a camera under the natural light, as shown in Figure 3.3 (c). The images are taken out from a web-based image database held by SIPI of USC [21]. The four texture classes are named *woollen cloth*, *wood grain*, *raffia* and *herringbone weave* respectively. Because they are quite similar in every aspect and character, this set is the most difficult one to be classified. There are 900 sample cutouts from 4 large images, each class has 225 samples. This dataset is referred as *texture*.

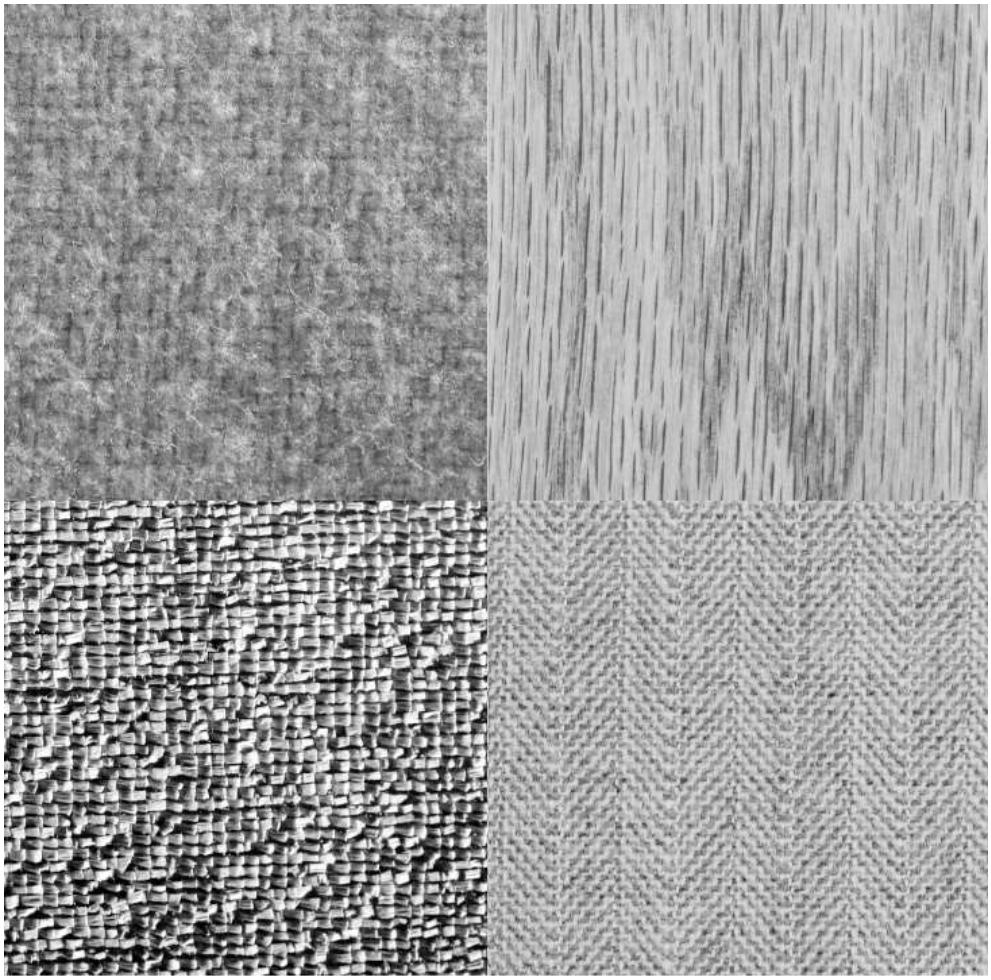


Figure 3.3: Sample images in dataset *Shape*

3.4 Summary on Characteristics of Datasets

The main characteristics of the datasets corresponding to each task are summarised in Table 3.1.

Table 3.1: Main characteristics of datasets

Dataset	Num. of Classes	Similarity Degree	Class Name	Mean	Noise Degree or Std. Deviation	Num. of cutouts
Shape	3	low	*black circle*	10	5	40
			grey square	180	50	40
			white square	220	220	40
Coin	3	middle	*head*	unknown	high	160
			tail	unknown	high	160
			background	unknown	high	180
Texture	4	high	*woollen cloth*	unknown	middle	225
			wood grain	unknown	middle	225
			raffia	unknown	middle	225
			herringbone weave	unknown	middle	225

Chapter 4

Basic Approach

In applying the GP basic approach to the three tasks described in 3, we mainly focus on feature selection and extraction.

4.1 Chapter Goals

The feature selection and extraction are always specific down to the task level. Even under the same problem domain, different tasks require different features input into GP system. We target at the following goals in this chapter:

- to identify suitable features, particularly for *Texture* classification.
- to provide a baseline for later chapters for the purpose of comparison and analysis.

4.2 Feature Selection, Extraction and Terminal Set

For the simple object classification tasks, the commonly used features under spatial domain are good enough to construct fitness cases. However, for some difficult tasks, such as classification of similar texture, the features under spatial domain are no longer sufficient. Some features under other domains might be needed to compose the terminal set.

For those image objects which do not have uniformity of intensities in small local regions, such as texture and medical X-Ray images, we need to extract some features which can reflect the variability in grey scale as well as capture the coarseness or fineness of surface of an object. This requires that the features under frequency domain are selected for putting into GP terminal set for some tasks like *texture* classification.

When pixels in an image has a sufficiently strong central tendency, i.e. a tendency to cluster around some particular value, then it may be useful in characterizing the image by a few numbers that are related to its *moments* [15] — the sums of integer powers of the pixel values.

Besides lower-order histogram features —Mean and Standard Deviation, two higher-order momentum features — Skewness and Kurtosis, are often extracted for hard tasks. In out application of basic GP approach, several combinations of these features are assigned to GP terminals for more difficult tasks (*coins* and *texture*).

4.2.1 Mean as the 1st Order Moment

Mean is to estimate the value around which central clustering occurs, its formula is:

$$\bar{X} = \frac{\sum_{i=1}^{N} X_i}{N}$$

The pixels distribution of an image with larger mean will be on the right to the distribution with smaller mean, as shown in Figure 4.1 (a).

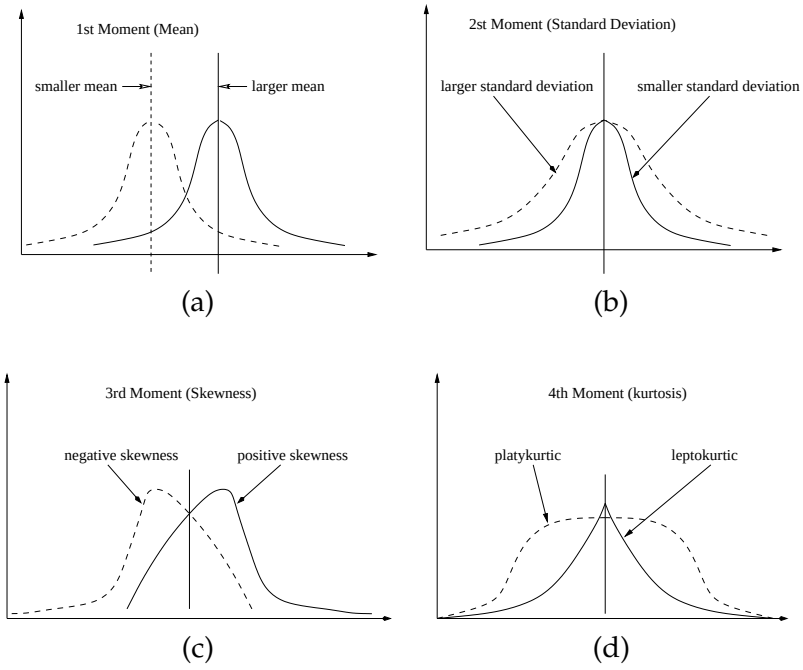


Figure 4.1: Pixels distribution of images: (a) larger mean vs smaller mean; (b) larger standard deviation vs smaller standard deviation; (c) positive skewness vs negative skewness; (d) platykurtic vs leptokurtic.

4.2.2 Standard Deviation as 2nd Order Moment

Having characterized a distribution's central value by 1st moment, one conventionally next characterizes its "width" or "variability" around that value. Standard Deviation is such a feature used to describe the dispersion of the pixels in an image, its formula is:

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{N} \left(X_i - \bar{X}\right)^2}{N}}$$

If two distributions have same Mean value, the one with larger Standard Deviation value will have a wider distribution body than the one with smaller Standard Deviation value, as shown in Figure 4.1 (b).

4.2.3 Skewness as 3rd Order Moment

Skewness is to characterize the degree of asymmetry of a pixel distribution around its Mean. Unlike Mean and Standard Deviation which have the same units as raw pixels — "grey

degree", the skewness is defined in such a way as to make it has different unit from row pixels. It is a pure number that characterizes only the shape of pixel distribution. Its formula is:

$$Skewness(X) = \frac{\sum_{i=1}^{N} (X_i - \bar{X})^3}{N\sigma^3}$$

If skewness is negative, the pixels are spread out more to the left of Mean than to the right, and vice versa, as shown in Figure 4.1 (c). The Skewness of a normal distribution is zero.

4.2.4 Kurtosis as 4th Order Moment

Kurtosis measures how outlier-prone the pixel distribution is, that is, measures the relative peakedness or flatness of the distribution. Its formula is:

$$Kurtosis(X) = \frac{\sum_{i=1}^{N} (X_i - \bar{X})^4}{N\sigma^4}$$

The kurtosis of a normal distribution is 3. Distributions that are more outlier-prone than the normal distribution have kurtosis greater than 3, and have a sharp peak around Mean. It is termed leptokurtic [15]. Distributions that are less outlier-prone have kurtosis less than 3, and have a flat around Mean. It is termed platykurtic [15], as shown in Figure 4.1 (d).

4.3 Function Set

In this project, we use the combination of normal arithmetic operators { +, -, *, % (protected division) }, Boolean statement {IF...THEN...ELSE}to compose in function set. Thus F = { +, -, *, %, IF...THEN...ELSE }. The meaning of "protect" can be referred to Section 2.2.1 Page 8. In our application, we let protected division returns 0 when it takes 0 as divisor. All the normal arithmetic operators form 2nd order nodes (i.e. take two arguments). Like the normal conditional branch, IF...THEN...ELSE statement form 3rd order nodes and takes three arguments: the first one is used as IF condition, the other two are used to be returned according to the evaluation of IF condition.

4.4 Parameters Selected and Termination Criteria

The fitness cases for each of the three sets are uniformly split into 3 portions to form training set, validation set and test set respectively. As the evolution proceeds, fitness drop, which indicates the over-fitting point, is checked every 10 generations, for an early stopping, although the best program and its fitness on validation set are recorded in the every generation base.

Both in initialization of 0 generation and mutation of programs, we use ramped half-and-half method to keep a certain degree of grogram tree balance and also a certain degree of randomness.

4.4.1 Parameters

The parameters are optimized based on many experiments with different parameter values. The parameters are summarized as below. For convenience of quick reference, we also summarize them in Table 4.1 for each task.

- population size: 300 for Shape, 500 for both Coin and Texture
- initial-max-depth: 5
- initial-min-depth: 3
- max-depth: 5 for Shape, 6 for Coin and 8 for Texture
- min-depth: 3
- maximal number of generations: 50
- crossover rate: 50%
- mutation rate: 30%
- reproduction rate: 20%

Table 4.1: Main parameter values for applying basic GP approach on *shape*, *coin* and *texture* datasets

Dataset	Parameter	Value
Shape	Pop. Size	300
	Initial Max. Depth	5
	Initial Min. Depth	3
	Max. Depth	5
	Min. Depth	3
	Max. Num. of Generations	50
	Crossover rate:	50%
	Mutation rate:	30%
	Reproduction rate	20%
Coin	Pop. Size	500
	Initial Max. Depth	5
	Initial Min. Depth	3
	Max. Depth	6
	Min. Depth	3
	Max. Num. of Generations	50
	Crossover rate:	50%
	Mutation rate:	30%
	Reproduction rate	20%
Texture	Pop. Size	500
	Initial Max. Depth	5
	Initial Min. Depth	3
	Max. Depth	8
	Min. Depth	3
	Max. Num. of Generations	50
	Crossover rate:	50%
	Mutation rate:	30%
	Reproduction rate	20%

4.4.2 Termination Criteria

We terminate a GP run when any of followings is satisfied:

- when the classification accuracy on the training set is 100%, OR
- when the maximal generation number (50) is reached, OR
- when over-fitting point is reached

4.5 Fitness Function

In this project, we calculate the correct classification percentage for a program over a set of fitness cases as the program's fitness. That is to say, a program's fitness is its classification accuracy.

The program in a population with the highest fitness is regarded as the best program. Its fitness on validation set is considered as the whole population's fitness when dealing with the termination criteria concerning overfitting.

4.6 Experimental Results

The final performance measure is in term of effectiveness and efficiency. The effectiveness for classification refers to the classification accuracy, which is calculated fitness of best program against test set. The efficiency is evaluated on both the training time and the index of the generation at which the final solution is found. The results of each experiment on each task are the average values based on 80 runs. The results are summarized in the Table 4.2.

Table 4.2: Results comparison of different feature combinations on the *shape*, *coin* and *texture* datasets using basic GP approach.

Dataset	No of Exp.	Feat Dscptn	Gen.	Time(s)	Acc(%)
Shape	Exp. 1	2 feats (Mean, SD)	7.51	0.07	97.62 %
	Exp. 2	4 feats (+ CircMean, CircSD)	8.59	0.09	96.16 %
	Exp. 3	8 feats (+ More CircMean, CircSD)	9.28	0.10	93.81 %
Coin	Exp. 4	2 feats (Mean, SD)	28.62	1.85	91.16 %
	Exp. 5	4 feats (+ CircMean, CircSD)	28.64	1.78	90.37 %
	Exp. 6	10 feats (+ More CircMean, CircSD)	32.70	2.09	93.13 %
	Exp. 7	12 feats (+ Skew, Kurt)	31.51	2.01	94.67 %
Texture	Exp. 8	2 feats (Mean, SD)	18.32	1.10	58.19 %
	Exp. 9	4 feats (+ CircMean, CircSD)	21.57	1.23	62.17 %
	Exp. 10	6 feats (+ More CircMean, CircSD)	25.73	1.57	59.23%
	Exp. 11	4 feats (Mean, SD, Skew and Kurt)	29.99	1.83	71.45 %
	Exp. 12	6 feats (+ CircMean, CircSD)	26.76	1.59	71.35 %

4.6.1 Abbreviations

The abbreviations in Table 4.2 are explained as follows:

- Mean: Mean over all the pixels of a sample image cutout, referred as overall Mean in the text.

- SD: Standard Deviation over all the pixels of a sample image cutout, referred as overall Standard Deviation in the text.
- CircMean: Mean on a selected circle on an image cutout, referred as circle Mean in the text.
- CircSD: Standard Deviation on a selected circle on an image cutout, referred as circle Standard Deviation in the text.
- Skew: Skewness of the pixel distribution of an image cutout.
- Kurt: Kurtosis of the pixel distribution of an image cutout.
- "+": The symbol "+" means that beside the features used in the last experiment, more features which follows "+" are added.
- Exp: Experiment.

4.6.2 Feature Combinations and Experimental Arrangement

We only extract Mean / Standard Deviation for *shape* dataset, because it is a relatively simple task. The first experiment on *shape* dataset (Exp. 1) is based on overall Mean / Standard Deviation. The second experiment (Exp. 2) is based on 4 features: besides overall Mean / Standard Deviation, one pair of circle Mean / Standard Deviation is added. In the third experiment (Exp. 3), we two more pairs of circle Mean / Standard Deviation on two different circles on a cutout are added. Thus, we make Exp. 3 to be based on 8 features.

For *coin* dataset and *texture* dataset, we extract features in the similar way, except that we add the Skewness and Kurtosis for some experiments.

For *coin* dataset, the first experiment (Exp. 4) is based on 2 features: overall Mean Standard Deviation. The second experiment (Exp. 5) is based on 4 features: besides overall Mean / Standard Deviation, a pair of circle Mean / Standard Deviation are added. The third experiment (Exp. 6) uses 10 features: overall Mean / Standard Deviation plus 4 pairs of circle Mean / Standard Deviation. In the last experiment (Exp. 7), we use previous 10 features in Exp. 6, plus the Skewness and Kurtosis, i.e. we get 12 features in total.

For *texture* set, the feature selection for the first two experiments (Exp. 8 and Exp. 9) is same as coin; In the third experiment (Exp. 10), we add one more pair of circle Mean / Standard Deviation, forming 6 features. In the forth experiment (Exp. 11), we use 4 features: overall Mean, overall Standard Deviation, Skewness and Kurtosis. The last experiment (Exp. 12) is based on 6 features: 4 features are same as last experiment, plus a pair of circle Mean / Standard Deviation.

Such a feature selection and experiments arrangement are mainly for two purposes:

- try to reveal whether the better performance can be obtained with more features used.
- try to reveal whether the features which characterize different aspects of an image cutout help in improving performance

4.7 Analysis and Discussion

A general rule revealed by all results is the more the features used for the terminal set, the longer the time to converge. This is because adding more features to the system enlarges the search space.

4.7.1 Analysis for *Shape* Task

In general, the results on *shape* show that Mean / Standard Deviation can lead to satisfactory results for easy image recognition problem.

Comparing Exp. 1, 2 and 3, we can see along with adding more circle Means / Standard Deviations to the terminal set, the performance (both effectiveness and efficiency) is getting worse. This is not a surprising result. Because the noise in each pixel of one class of shape is generated randomly around its Mean and all pixels is uniformly spread over the whole shape, the Circle Mean / Standard Deviation characterize the same aspects as the overall Mean / Standard Deviation. Adding more circle Means and Standard Deviations is just to increase the search space without adding valuable candidate solution. Consequently, it makes the efficiency worse.

Furthermore, circle Mean / Standard Deviation are calculated on less pixels than overall Mean / Standard Deviation, its "quality" is not as good as overall Mean / Standard Deviation. This adds more noise to features that belongs to Mean / Standard Deviation Category. In machine learning term, it increases the search space by adding worse candidate solution to it. Therefore, it decreases the effectiveness as well.

4.7.2 Analysis for *Coin* Task

In general, the results on *coin* dataset show that overall Mean / Standard Deviation can obtained the reasonable performance on tasks with middle-level difficulty like em coin (Exp. 4), but adding different features can improve the classification accuracy (Exp. 5, Exp. 6 and Exp. 7). It should be noticed that although overall Mean / Standard Deviation and circle Mean / Standard Deviation are still in same category, but they capture different characteristics of the object appearance this time, because*coins* do vary their appearances radically.

The interesting fact (refer to Exp. 4, Exp. 5 and Exp. 6) is that adding circle Means / Standard Deviations sometimes makes effectiveness worse, and sometimes better. For *coin* task, *exactly* selecting the circle center and radius becomes the key. If the selection of circle center and radius happens to make the circle Mean / Standard Deviation capturing some characteristics of a coins appearance, this set of features will help in improving the effectiveness. Otherwise, it adds erroneous values to terminal set and consequently decreases the classification effectiveness. The circle center and radius are manually selected during object cutouts preparation.

The two higher-order momentum features are originally expected for improving classification accuracy for *texture* set, fortunately, they also help in *coin* set. Adding Skewness and Kurtosis to terminal set (Exp. 7) gets the highest effectiveness than just using Mean / Standard Deviation (Exp. 4, 5 and 6) for this task. This reveals that the higher-order moment-based features do reflect some aspect of image appearance such as fineness.

4.7.3 Analysis for *Texture* Task

The experiments on *texture* dataset clearly reveal that Skewness and Kurtosis add "power" to features. It makes the classification accuracy improved obviously. In detail, comparing Exp. 9 with Exp. 11 as well as Exp. 10 with Exp. 12, we find that Exp. 9 and Exp. 11, Exp. 10 and Exp. 12, use same number of features, but the classification accuracy of Exp. 11 and Exp. 12 are much better than Exp. 9 and Exp. 10. At the same time, they do not decrease much on the classification efficiency. One should conclude that adding different "type" of features, which capture different aspects over the object cutout, will greatly increase the classification effectiveness. The feature selection should mainly focus on quality of features rather than quantity of features.

It is same that adding circle Means / Standard Deviations sometimes makes effectiveness worse, and sometimes better, for *texture* task, too. But we argue that the reason is absolutely different. Although *texture* does vary its pixel value in radius direction, this variation is occurs only in a small location. In larger scale, there is no radical variation.

However, there is no satisfactory classification accuracy reached only by adding higher-order momentum features. More "powerful" features (such as "energy") or more "powerful" feature extraction methods (such as Wavelet transform) are needed for a good result, but this is out of the scale of this project.

We also need to find other ways around GP itself, rather than feature selection and extraction, to improve GP performance on image recognition.

4.8 Chapter Summary

For a specific task, adding features which capture the different characters of the object helps in improving classification effectiveness, at the same time, does not lower down much the GP process efficiency. Adding more but the same "type" of features does not help in improving effectiveness, and sometimes even decreases the effectiveness, but the effect is minor. Generally, if the features newly added to the terminal set reflect the same aspect of the objects that has already captured by old one, it will enlarges the search space and make efficiency worse.

For relatively hard task, performance improvement by only feature selection and extraction is not big enough. GP should find its own way for difficult image recognition problem.

Overall, the basic GP approach did not achieve satisfactory results on relatively difficult problems. However, the application of the basic GP approach provides a basic line of this project as a starting point. In later chapters, we will extend the crossover in some smarter ways and examine whether the extended crossover operators can improve the performance.

In this project, we mainly focus on a comparison between basic GP approach and other newly developed GP crossover operators, and a comparison among the three tasks. For convenience and comparability, we will use 4 features for each task in the experiments for later chapters, they are Exp. 2 for *shape*, Exp. 5 for *coin* and Exp. 11 for *texture* in Table 4.2.

Chapter 5

Headless Chicken Crossover

The basic GP approach can not get acceptable result for hard tasks, as shown in Chapter 4. We have doubted that the hill climbing crossover *greatly* outperforms the basic GP approach for difficult tasks in Section 2.3 of Chapter 2 (Page 14). In this chapter, we will investigation this crossover operator on our three tasks so as to reason our doubtfulness.

5.1 Chapter Goals

The reason that crossover is the predominant operator lies in: the individuals in a population can communicate each other by crossover rather than mutation or reproduction. However, a lot of empirical evidence shows that the standard GP crossover acts primarily as a macromutation operator. That is why there is a lot of empirical evidence showing that mutation-base system works well or slightly better than GP system.

Although headless chicken crossover combines GP crossover with hill climbing search, it is still a sort of macromutation. The big performance improvement by headless chicken crossover on some tasks only prove that the bias of hill climbing algorithm is particularly well for some specific problem, such as Boolean 3-multiplexier problem [11], which has no worry about local minima.

In this chapter, we will investigate headless chick crossover operator on our three tasks. In particular, we are trying to achieve the following goals:

- to add more weights to the crossover-is-macromutation theory.
- to justify that headless chicken crossover is still in scale of macromutation.
- to verify that headless chicken crossover fails in elimination the disruptive effect of GP crossover by the observed the stagnation of GP runs — the result of GP ”bloat”.
- to give more confidence to the direction on improving crossover by preservation of good building block as well as homology.

5.2 Revisit of Headless Chicken Crossover

As shown in Figure 5.1, only one parent A is selected and an entirely new individual $B1$ is created randomly. The selected parent A is then crossed over with the new and randomly

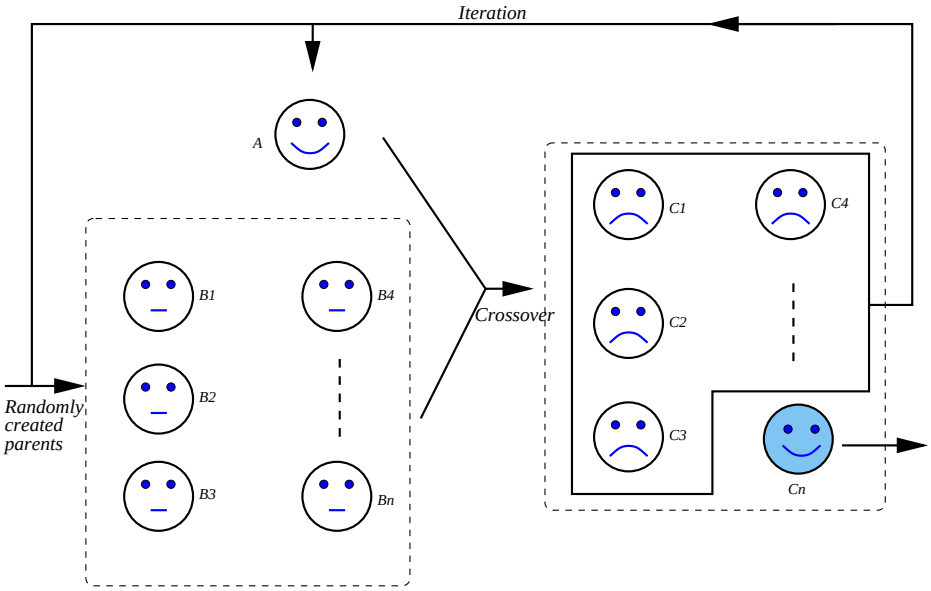


Figure 5.1: The illustration of headless chicken crossover

created individual *B1*. The offspring *C1* is kept if it is better than or equal to the parent *A* in fitness. Otherwise, it is discarded and entirely new individual *B2* is created randomly, a new crossover occur between *A* and *B2*. The iteration goes on until the offspring *Cn* is better than or equal to the parent *A* in fitness. Cn is kept and put in the next generation. Thus, headless chicken crossover is a form of hill climbing.

5.3 Experimental Results and Analysis

To be consistent with previous chapters, the results on each task are the average values based on 80 runs with different random seeds. Noticed that evaluation of children will be time-consuming, we only use 1/5 training set to save time, because it is only important that the child selected to put in the next generation is approximately better than the parent.

During doing the experiments, we found, for some selected parent which has a very high fitness, and it takes hours to produce a good child by headless chicken crossover, this extremely slow down the evolutionary process. So we limit the number of crossover iterations for one parent to be 10000, when the operation exceeds this number, the iteration stop and the candidate parent is put into the next generation directly without any change, i.e. crossover turns to reproduction for this parent, though this situation is quite rare.

The results is shown in Figure 5.1 with result of basic GP approach for comparison.

The results in Table 5.1 shows that the performance improvement on hard tasks like *text* is still unsatisfactory by headless chicken crossover.

As we expected, headless chicken crossover slightly outperforms standard GP crossover from classification accuracy point of view. But the significant improvement is on the number of generations needed to find solution, i.e. the convergence speed. This proves that the performance improvement is mainly the result from the effect of employing the hill climbing strategy rather than crossover operation itself.

However, although we have already made effort to save time, such as using small training set to evaluate children, this approach is still extremely time-consuming. This is because

Table 5.1: Results of headless chicken crossover and basic GP approach on *shape*, *coin* and *texture* datasets

Dataset	Approach	Generation	Time(s)	Accuracy(%)
Shape	Basic GP crossover	8.59	0.09	96.16%
	Headless Chicken Crossover	2.29	0.49	98.03%
Coin	Basic GP crossover	28.64	1.78	90.37%
	Headless Chicken Crossover	20.14	42.16	92.37%
Texture	Basic GP crossover	29.99	1.83	72.45%
	Headless Chicken Crossover	19.23	71.40	74.78%

it makes too much effort on hill climbing, and this search is "headless". The randomness of creation of another parent and randomness of selection of crossover point are the source of this headlessness.

Headless chicken crossover, in fact, turns off the primary crossover and turns on sort of hill climbing mutation, because it cuts off the communication among the existing members of a population. Thus, headless chicken crossover also cuts off the possibility to preserve homology and simulate biologically homologous crossover, which should be the central issue in redefining the crossover operation. Headless chicken crossover is still in the scale of macromutation.

The fact that headless chicken crossover turns off the standard GP crossover proves that macromutation techniques with hill climbing techniques may perform as well as and sometimes slightly better than basic GP crossover. This, from another perspective, also proves that the traditional GP crossover acts primarily as macromutation.

Additionally, we found the evolution process converge relatively faster at the beginning and become very slow at the end of a run. We have also monitored the tree structure changes during a GP run. We found that programs are fairly small and unbalance in early stage of the evolution, but they are getting bigger very fast and tend to become more balanced. These two observed facts are highly consistence with the stagnation of GP run — the result of GP "bloat". The disruptive effect of crossover is the most suspicious source for GP bloating effect. This, from another perspective again, proves that performance improvement by headless chicken crossover is mainly from hill climbing rather than eliminating the disruptive effect of crossover.

5.4 Chapter Summary

Focussing on the combining genetic operator with hill climbing search strategy, headless chicken crossover slightly outperforms the traditional GP crossover. However, it is an extremely time-consuming approach and a trade-off between effectiveness and efficiency need to be carefully considered before adoption of the approach in practice. Headless chicken crossover cuts off the possibility of homologous crossover, leave very litter room for further improvement. Headless chicken crossover is still in the scale of macromutation and the fact it is slightly better than basic GP crossover, somehow, prove that basic GP crossover is macromutation. The fact that headless chicken crossover also has the GP bloating effect shows that it does not preserve good building block very much.

Because preserving good building block as well as homology becomes the center of storm in GP crossover research, we do not regard headless chicken crossover as the direction for further development.

Chapter 6

Looseness-controlled Crossover

6.1 Introduction

The basic GP approach and headless chicken crossover can not get acceptable result for hard task as shown in Chapter 4 and 5. As we discussed in Chapter 1 and 2, crossover operator tends to break good building block at later stage of an evolution process, this is due to the randomness of selection for the crossover points. In this chapter, we describe a new crossover operator we have developed — Looseness Controlled Crossover (LCC).

6.2 Chapter Goals

LCC approach is based on selected crossover points to improve GP performance, we will introduce our development concerning this approach in this chapter and its extension in next chapter. In this chapter, we try to achieve the following goals:

- to propose a way to identify a good building block from other part of a tree.
- to build up our methodology in controlling selection of crossover points, which targets at eliminating the disruptive effect of standard GP crossover.
- to construct the fundamental elements for further improvement in this direction.

6.3 Basic Idea

To avoid randomly selecting the crossover points, we give each link between two nodes in a program tree a value - looseness, and let this value reflect how "sticky" the two nodes should be with each other. Ideally, the looseness values of links within a good building block should be smaller than other parts of the tree, i.e. the nodes within this block should be more sticky one another. The smaller the looseness value, the better the block. On the other hand, when we select the point for crossover, only the link with the biggest looseness value in the tree are selected. If more than one links have the biggest looseness value, the selection among these links are random.

According to the definition of looseness above, we state our hypothesis on LCC:

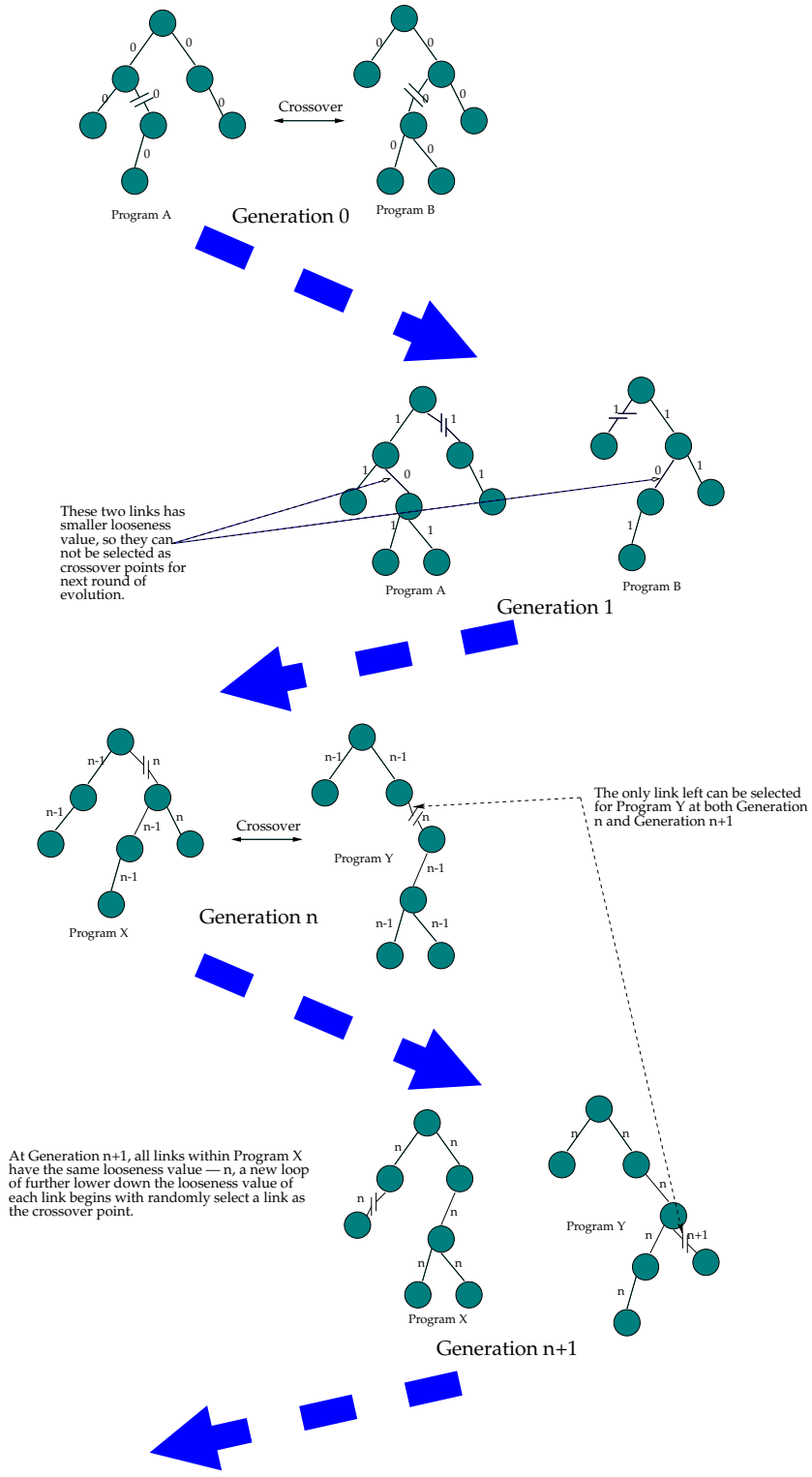


Figure 6.1: The illustration of looseness-controlled crossover

Hypothesis: If we assign looseness value to each link at start point and update them during the evolutionary proceeds, then according to the value, a good building block is more sticky than other non-building-block parts which have higher looseness value. By selecting the crossover point only on those loosest links, the disruptive effect of crossover should be eliminated.

How building block grows or in which way it grows is still an issue that is far more

from being discovered. Thus, according to the range in which the links are made sticky after crossover, we propose three versions of this new crossover operator. The three versions will be addressed and experimented individually in the rest of this chapter.

6.4 Methodology

6.4.1 Version 1

As shown in Figure 6.1, in generation 0, all links in all grogram trees are assigned 0 as the looseness value. This is reasonable, because no good building block has been recognized at generation 0 (even though there might exist), and we treat all links equally. Thus, the selection of the crossover point are random at the first round evolution. Once crossover happened between two nodes, supposing that the new link are better than the old one, we give the crossover-ed point a smaller looseness value, so that the two nodes connected by this link are more sticky than other parts of the tree. This is done by keeping the looseness value of this link unchanged and increasing values of all other links of the tree by 1, i.e. the looseness value of these other links will have same number as the index of generation. In next round of evolution, this link will not be selected as the crossover point, the two nodes are considered as a small good building block. With the evolution process going on, more and more nodes are getting sticky and the good building blocks grow and connect with each other, when all links have a looseness value of gen-1 where gen is the index of current generation, a new loop of further reducing the looseness down to gen-2 starts. We expect that the final solution program is itself a good building block as a whole and have a fairly small looseness value.

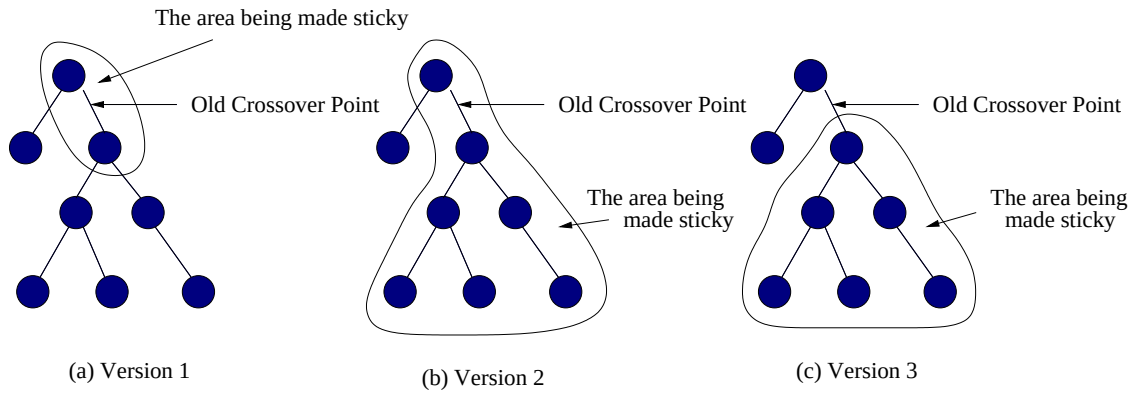


Figure 6.2: The illustration of 3 versions for basic looseness-controlled crossover: (a) Version 1 (b) Version 2 (c) Version 3

6.4.2 Version 2 and Version 3

The above version focuses on making only the link selected as the crossover point more sticky than the other parts of the tree as shown in Figure 6.2 (a). However, building block can grow in other ways rather than the way shown in Figure 6.2. So after the crossover operation, we can select other range of links to make sticky. Thus, we get another two versions: supposing that the whole new subtree including the crossover point is better than the old one, version 2 makes the subtree crossovered more sticky, including the crossover link itself, as shown in Figure 6.2 (b). On the other hand, supposing that only the new

subtree but excluding the crossover point are better than the old one, version 3 makes only the crossovered subtree more sticky, excluding the crossover link itself, as shown in Figure 6.2 (c).

6.5 Experimental Results

To be consistent with previous chapters, the results on each task are the average values based on 80 runs with different random seeds. The three versions of LCC are examined individually and the results are extracted in the Figure 6.1 together with the basic GP approach results for the convenience of comparison.

Table 6.1: Results of three versions of LCC and basic approach on the *shape*, *coin* and *texture* datasets

Dataset	Approach	Generation	Time(s)	Accuracy(%)
Shape	Basic	8.59	0.09	96.16%
	LCC(version1)	8.68	0.13	97.53%
	LCC(version2)	9.03	0.14	97.17%
	LCC(version3)	8.47	0.13	96.73%
Coin	Basic	28.64	1.78	90.37%
	LCC(version1)	28.80	3.36	91.02%
	LCC(version2)	29.69	3.54	90.35%
	LCC(version3)	29.62	3.55	90.85%
Texture	Basic	29.99	1.83	72.45%
	LCC(version1)	26.31	2.05	70.07%
	LCC(version2)	26.39	2.07	71.81%
	LCC(version3)	25.76	1.81	69.68%

From the table above, the three versions of LCC, in general, slightly improve the classification accuracy for the first two tasks, but surprisingly make little worse for the texture task. It seems the new crossover operator does not help in improving the effectiveness. In term of efficiency, the situation is just reversed, the more difficult the task, and the quicker the convergence of the evolution process.

Amongst the three versions, the first two are definitely better than the third one from effectiveness point of view. We noticed that the first two versions are both involved the crossover links itself: i.e. the first version makes only the links themselves sticky; the second version makes the whole subtree sticky, including the links. Whereas, the third version makes the whole subtree sticky, but excluding the links. On the other hand, there is no big difference amongst all three versions from efficiency point of view, although the average execution time of LCC increases a bit due to a high calculation complexity.

LCC seems failed, because it almost does not help in improving the performance in general.

After a detail analysis, we found that our hypothesis is not very accurate. Our hypothesis is based on, for all three of the versions, a precondition: after the crossover, the new link, the new subtree (including the crossover point) or the new subtree (excluding the crossover point) are better than those old ones in the new context. However, a normal crossover operation does not guarantee this. On one hand, after the crossover, the new link or subtree

might be worse than the old one in the new context. On the other hand, even if the new program's performance gets better after the crossover, we can not say it must be resulted from the crossovered link (as a small building block) or crossovered subtree, it might be resulted by another part of the program tree in the new tree structure, or some lager-scale subtree which contain the crossovered subtree and link.

During the experiments, we observed that a good building block in a program can lead to a very poor performance if put it in the wrong place of another program, and a bad building block in a program can lead to a very good performance if put it in the "right context" of another program. We also observed experimental fact that two programmes can produce very bad offspring by crossover operation, even though the two crossovered subtrees are both good building block in the original parent program, and vice versa.

6.6 Chapter Summary

The idea of the LCC is launched to eliminate the randomness of crossover point selection. We expect the links within a building block have less looseness (i.e. are more sticky) than other parts of the program tree. But it is hard to identify where the good building block is in a program, because one building block can be "good" only in this program's context, and it might result in a very bad performance while putting it in another program and coping with other parts of the program.

After this very first try on this direction, we need further exploration on this approach. We have to put more efforts on making the precondition of our hypothesis become true. We will make an extension on LCC in next Chapter.

Chapter 7

Extended Looseness-controlled Crossover

7.1 Introduction

In last chapter, we have described LCC approach in which we introduce the idea of looseness. We use looseness value to reflect how "sticky" to adjacent nodes are and distinguish a good build block from other parts in a program tree. We also use it to minimize the randomness of selecting crossover points so as to reduce the destructive effect of normal crossover operation. However, the manipulating and updating the looseness values of links is based on the assumption that new programs after crossover have better building blocks around crossover points than their parents, but this is not always true.

Borrowing some ideas from headless chicken crossover[11], which, in term of machine learning, combine hill climbing search on individual crossover operation and beam search on whole generation, we try to extend the basic LCC approach by enforcing the assumption above to be true in this chapter. That is our extended Looseness Controlled Crossover (eLCC).

7.2 Chapter Goals

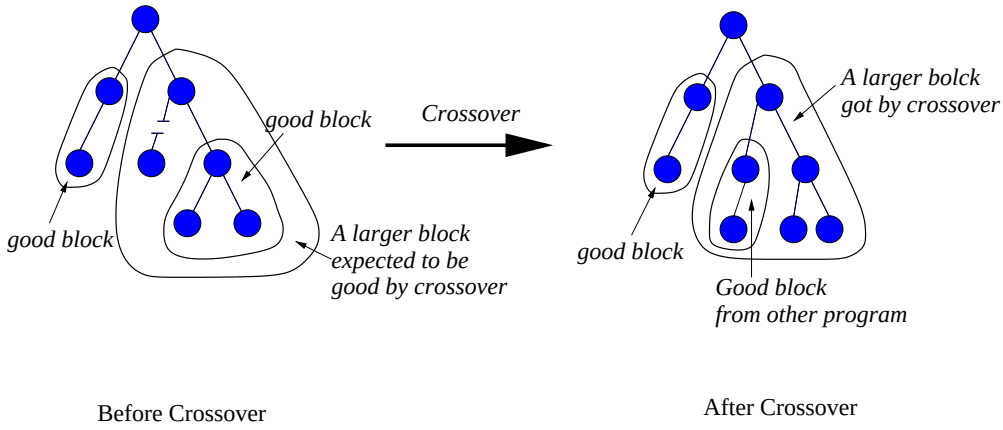
Based on basic LCC approach, eLCC is purely a methodological extension, by which we try to make the pre-condition of our hypothesis in Section 6.4 of last chapter guaranteed. In detail, we try to achieve the following goals in this chapter:

- Verify the hypothesis given for LCC
- Extend the basic methodology of LCC in the way of enforcing that the precondition of our hypothesis to be true
- Predict the future works on this approach

7.3 Theory Behind

Our hypothesis of LCC is assuming that new programs after crossover have better building blocks around crossover points than their parents. But this assumption needs to be pre-satisfied to make the hypothesis accurate.

In last chapter, we argued that a good building block in one solution can become the source of bad performance of another solution. The figure 7.1 shows one case of this possibility. Before the crossover, the program has two good building blocks. It is expected to grow one of them by crossover to make a larger good building block. After the crossover, although it replace the unsatisfied part of the larger block with a good building block from another program, it is not guaranteed to form a larger good building block. To check if the larger area is a good grown building block, we need to evaluate the new program.



The larger block is better or worse can be evaluated by the whole program's fitness: if the The fitness is better than before, we regard the block as a good grown building block, otherwise there is no building block has been grown.

Figure 7.1: The illustration of building block growing

To make an offspring program contain better building block around the crossover point than its parent, we evaluate it by comparing its fitness value with its parent's one. If the program has better fitness than its parent, we consider this crossover as a successful one, and a larger good building block has been formed. Otherwise, we treat this crossover operation as a failed one. The failed crossover operation is undone and a new crossover operation is attempted. This progress continues till the offspring has better fitness than its parent. In this way, we suppose that the building block around the crossover point has been grown or has been getting better.

7.4 Methodology Extended

We assign looseness values in generation 0, update and manipulate these values in other generations in the same way as described in last chapter. We keep the same way to select crossover points, too. The extension focuses on how we make the offspring programs has better building block than the parents.

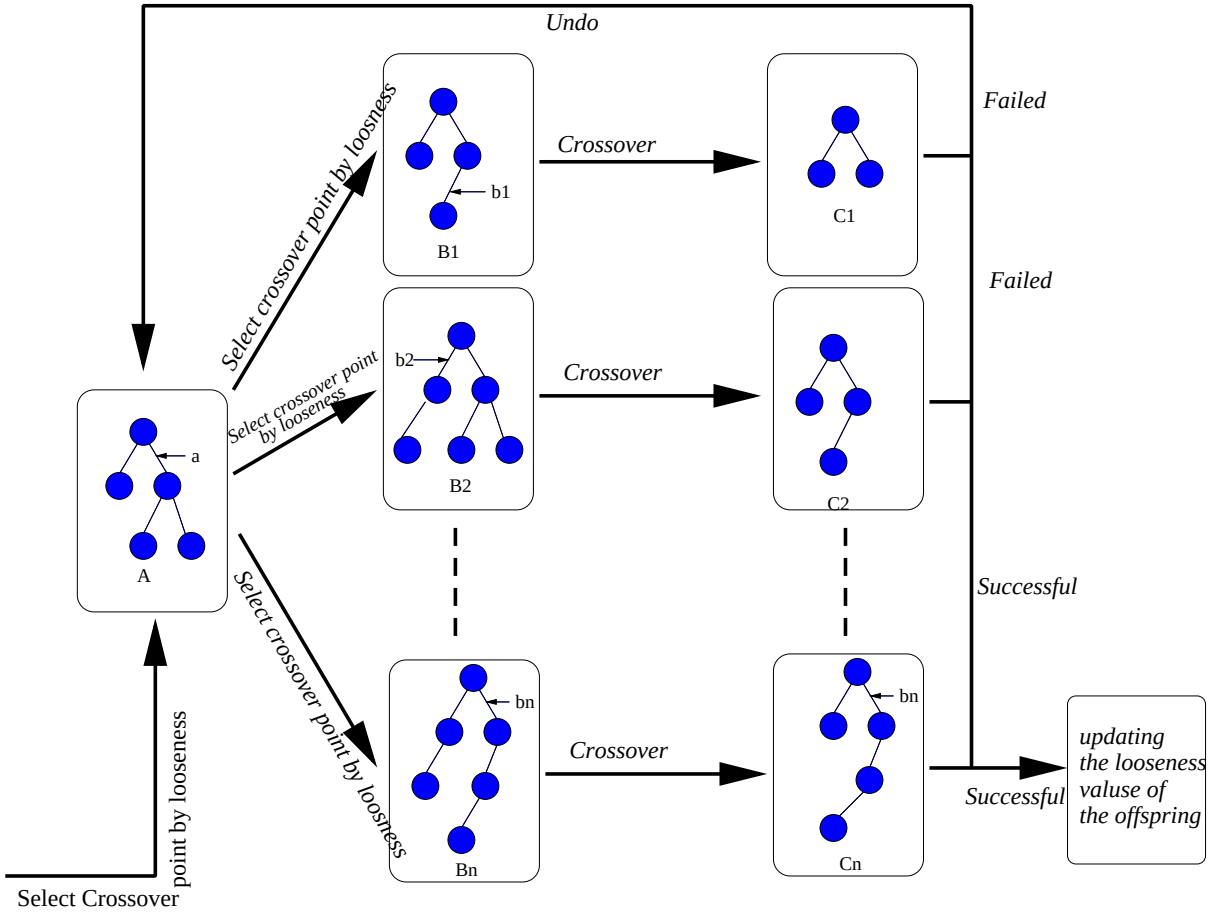


Figure 7.2: The illustration of extended looseness-controlled crossover

As shown in Figure 7.2, originally, two parent programs *A* and *B1* are selected and their crossover points *a* and *b1* are chosen according to the looseness value on each link. Then crossover operation occurs by replacing the subtree at *a* with the subtree at *b1*, we get a child *C1*. We then evaluate it by compare its fitness with its parent *A*'s. If it is getting better, we update the looseness value; otherwise, we undo the crossover operation and find another parent *B2*, and try a new crossover operation, this process continues in the form of loop till we find a child with better fitness, such as *Cn*.

In deed, the number of the iteration n can be any of the natural number from 1 to ∞, because there is the case that a successful crossover can never be found, due to the reason that the candidate parent as a whole is already a quite good building block and no other suitable program in the current population can make it better by crossover. In practice, we limit this number to 1000, i.e. when the iteration has been executed beyond 1000 times and still no better child has been found, we terminate the iteration and elite the parent and put the copy in the next generation.

Again, we implementation three version for eLCC, according to the range that the links are made "sticky" after the successful eLCC crossover, the same way as described in LCC.

7.5 Experimental Results and Analysis

To be consistent with previous chapters, the results on each task are the average values based on 80 runs with different random seeds. The three versions of eLCC are examined individually and the results are extracted in the Figure 7.1 together with the basic GP approach results for the convenience of comparison.

Table 7.1: Results of three versions of eLCC and basic approach on *shape*, *coin* and *texture* datasets

Dataset	Approach	Generation	Time(s)	Accuracy(%)
Shape	Basic	8.59	0.09	96.16%
	eLCC(version1)	4.21	0.31	97.88%
	eLCC(version2)	4.33	0.32	97.75%
	eLCC(version3)	4.53	0.31	97.84%
Coin	Basic	28.64	1.78	90.37%
	eLCC(version1)	19.96	9.23	92.74%
	eLCC(version2)	20.09	8.77	92.56%
	eLCC(version3)	20.64	9.74	92.21%
Texture	Basic	29.99	1.83	72.45%
	eLCC(version1)	22.50	12.84	77.18%
	eLCC(version2)	21.86	11.22	76.74%
	eLCC(version3)	21.77	10.20	76.15%

The above figure shows that all three versions of eLCC, in general, have improved classification accuracy and used less number of generations than basic GP approach for all three tasks. Furthermore, with the tasks' difficulty increasing, the performance improvement is getting more impressive. But in term of execution time, eLCC approach is extremely time-consuming due to the effort that is made when the each individual crossover operation tries to make the hill climbing search. A further investigation is needed to shorten the execution time. For example, when we evaluation the child program, if we can check its fitness only on small subset of validation set rather than the whole validation set.

Amongst the three versions, the first two versions are slightly but definitely better than the third one. This is consistent with the assumption that the range to be make "sticky" after crossover should cover the crossover points themselves rather then only the crossover-ed subtree, so that these points are not to selected as the crossover points in the next generation.

Between the first two versions, the first one is slightly better than the second, which suggests another hypothesis that the building blocks grows in a way of "link-by-link" rather than "block-by-block", as shown in figure 7.3, and it needs to be examined further, but this is out of the range of this project.

7.6 Chapter Summary

Combining the hill climbing search theory for individual program and the selective crossover point, eLCC outperforms basic GP approach as well as LCC described in last chapter in both classification accuracy and the speed of convergence. With the increment of difficulty

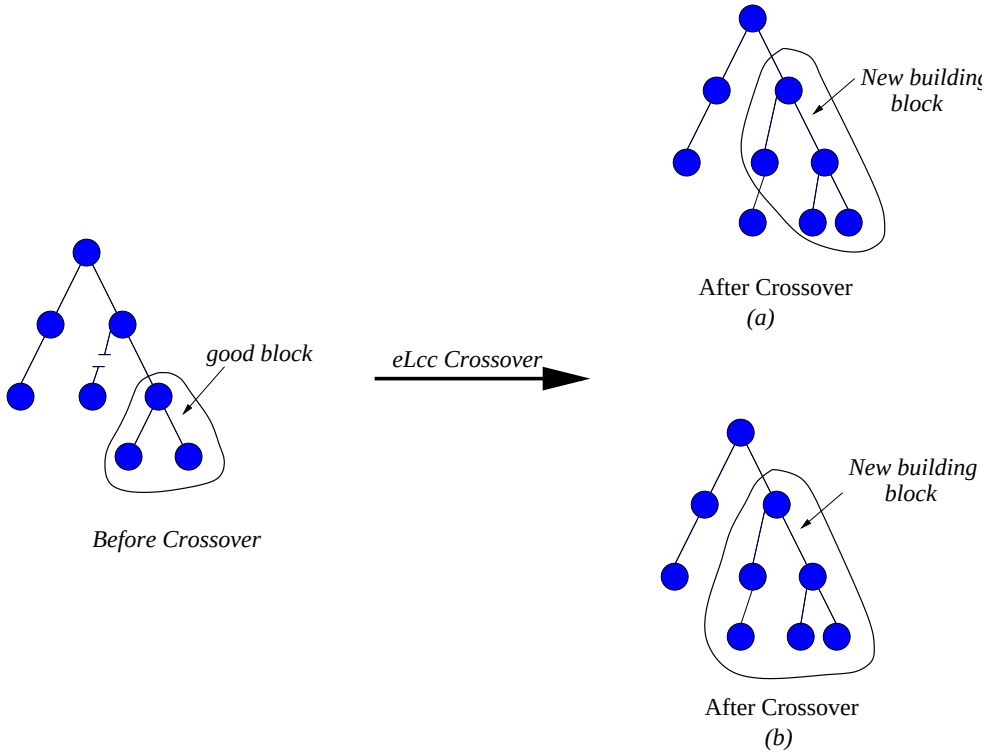


Figure 7.3: The illustration of the way in that building blocks grow: (a) link-by-link (b) subtree-by-subtree

of tasks, the improvement is getting more. From time consumption point of view, eLCC is also an acceptable approach. Comparing to headless chicken approaches, it saves huge amount of running time by limiting the crossover points with highest looseness value in each program tree.

However, eLCC mainly focus on preserving the good building block rather than homology. We can still extend it further to have the homology issues considered or combined.

Chapter 8

Brood Recombination Crossover

8.1 Introduction

The ordinary crossover operator constructs good building blocks at the early stages and tears them apart at the later stages during the GP evolution process, as discussed in previous chapters. We have developed eLCC approach to minimize the destructive effect of crossover operation in Chapter 6 and Chapter 7. In this chapter we examine another approach — brood recombination crossover, which focuses on giving GP the ability to look ahead when adjusting the beam/population and eliminating the disruptive force of normal GP crossover.

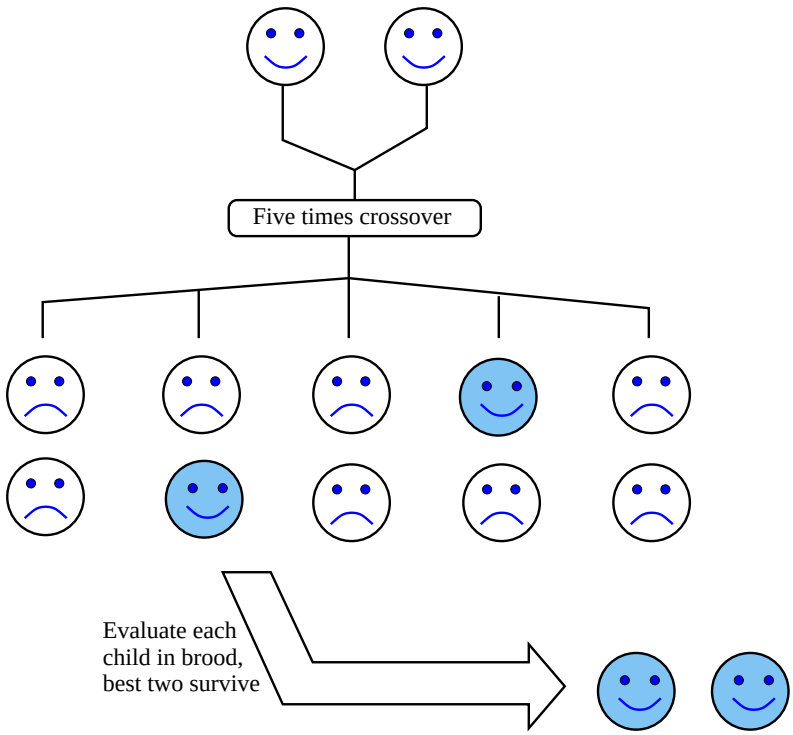


Figure 8.1: The illustration of brood recombination crossover

Firstly drawn on work by Altenberg [1], Tackett developed the new crossover operator — brood recombination in 1994[20]. In the real world, many animal species produce far

more offspring than needed to survive, and to keep the population size of one specie not changed much, only small amount individuals out of the "brood" live, others die. Tackett has modelled this observed fact in brood recombination crossover: when doing each crossover, this operation will happen more than once on same pair of parent programs, and a brood is produced, as shown on Figure 8.1. The size of the brood should be natural number multiply 2, i.e. 2N. Then all children in the brood is evaluated by its fitness, only the best two survive, others all discarded.

However, there seems to be a shortage with this approach. GP is usually slow in performing evaluation. Instead of having only two children to evaluate in basic GP, GP with brood recombination crossover will have a whole brood to be evaluated. The large amount of evaluations on those broods in this approach will make GP extremely slow. To solve this problem, Altenberg and Tackett evaluate the broods only on a small portion of the training set. They argues that it is only important that the selection on the brood selects children that are "in the ballpark" — not that they are certainly the best of the brood [4]. We will follow this idea to do the experiments in this Chapter. We only use 1/5 portion of training set for the evaluation of each brood.

Additionally, as the building blocks got larger, the probability of destructive crossover would increase. This suggests that a dynamic form of brood recombination, i.e. the brood size should grow as evolution proceeds to protect larger building block from disruption.

The key parameter in brood recombination is the brood size 2N. In Figure 8.1, the brood size is 10, i.e. N=5.

8.2 Chapter Goals

Based on the existing ideas of brood recombination crossover, we will further investigate :

- Whether increasing the brood size 2N will help in improving performance, and if so, how big the improvement will be.
- whether a variable brood size 2N in one GP run will help in improving performance
- some suitable functions N(gen) which can be used to adjust N during evolution process, where gen is the index of generation, and compare these functions

The new ideas will be examined and compared with basic GP approach on the same three classification tasks described earlier.

8.3 Experimental Results and Analysis on fixed Brood Size

To be consistent with previous chapters, the results on each task are the average values based on 80 runs with different random seeds.

We firstly examine the relationship between the performance and the brood size 2N. To do this, we have fixed the N with 4, 8, 12, 16, and 20 respectively when we run GP with brood recombination crossover for each tasks. A collection of the result is give in Table 8.1 together with the basic GP approach results for the convenience of comparison.

Table 8.1: Results of brood recombination crossover with fixed different brood size and basic approach on the *shape*, *coin* and *texture* datasets

Dataset	Approach	Generation	Time(s)	Accuracy(%)
Shape	Basic (N=1)	8.59	0.09	96.16%
	Brood Recombination (N=4)	5.26	0.10	98.25%
	Brood Recombination (N=8)	3.48	0.10	98.25%
	Brood Recombination (N=12)	3.01	0.11	98.12%
	Brood Recombination (N=16)	2.66	0.12	98.44%
	Brood Recombination (N=20)	2.30	0.13	98.03%
Coin	Basic (N=1)	28.64	1.78	90.37%
	Brood Recombination (N=4)	21.88	2.50	92.42%
	Brood Recombination (N=8)	19.70	3.07	93.08%
	Brood Recombination (N=12)	17.59	3.53	92.82%
	Brood Recombination (N=16)	15.85	3.89	93.08%
	Brood Recombination (N=20)	15.94	4.49	92.80%
Texture	Basic (N=1)	29.99	1.83	72.45%
	Brood Recombination (N=4)	26.01	3.31	76.68%
	Brood Recombination (N=8)	21.82	4.23	76.46%
	Brood Recombination (N=12)	23.69	6.09	79.82%
	Brood Recombination (N=16)	20.00	6.12	80.71%
	Brood Recombination (N=20)	17.80	6.50	78.13%

The table above shows that GP with brood recombination greatly improves the performance in term of classification accuracy and number of generations needed for the solution for all three tasks, comparing with the basic GP approach. The more difficult the task, the more obvious the improvement.

However, along with the brood size increasing, the classification accuracy is not improved in a linear mode, and neither the convergence speed.

8.3.1 Curves: Accuracy vs N and Generations vs N

If we treat the basic GP approach as a special situation of brood recombination, where N=1 and brood size is 2, then we can abstract two sorts of curves as shown in Figure 8.2 to show the relationship between brood size 2N and the performance. In Figure 8.2 (a), the horizontal axis is the value of N, the vertical axis is the classification accuracy. In Figure8.2 (b), the horizontal axis is still the value of N, the vertical axis is the number of generations needed to find the solution. Thus, we have drawn two series of curves to show how the performance is effected by N: Figure8.2 (a) shows how the classification accuracy is effected by N, Figure 8.2 (b) shows how number of generations needed for finding the solution is effected by N, i.e. how convergence speed of GP evolution process is effected by N. Each curve is for a task. In both (a) and (b), we have shifted the curves vertically so that they all have the same starting point. There are two reasons to do this shifting: firstly, these curves are abstracted to reveal not only the relationship between performance and N, but also to show how big the performance improvement is among the three tasks; secondly, this shifting can save large room on the figure.

Figure 8.2 (a) shows before N is increased up to around 6, the classification accuracy is improved fast and stably for all three tasks. After the accuracy reach the peak where N

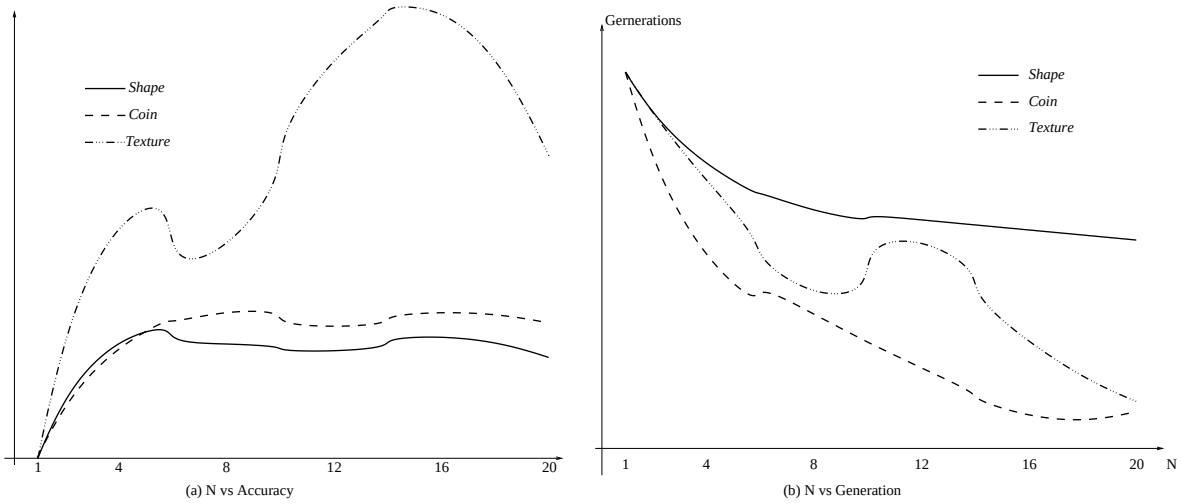


Figure 8.2: Improvement by brood recombination crossover for the three tasks (a) Accuracy (b) Generation

has the value 6, and along with N increasing, different task has different performance: for *Shape* and *Coin*, the accuracy goes to almost unchanged; for *Texture*, the accuracy goes to unstable with a generally improving trend, it goes to a second peak, where N has the value around 15. Figure 8.2 (b) shows a similar situation: while N is increased up to around 8, the number of generations needed to find the solution are decreased fast and stably for all three tasks, and the GP convergence gets faster. After this point, the curve for *Shape* goes down continuously but relatively slowly, the curve for *Coin* go down but relatively fast, the curve for *Texture* goes unstable with a generally going-down trend.

8.3.2 Brood-diversity Point

The situation shown by Figure 8.2 are interesting. Unlike the real animal world, where two parents can produce a large brood, in which there is almost no same individuals due to big size of the parents' chromosome. The big size of chromosome results huge number of crossover points. Additionally, the biological crossover allows multiple crossover points in one chromosome in one crossover operation. But the candidate programs in GP have very small size, and in one crossover operation only one crossover point is selected, i.e. only single part of the program tree (rather than multiple parts) is crossed over. Furthermore, in our GP system, we limit each programs by programs size criteria, such as maximal tree depth, minimal tree depth, etc, we thus have very even fewer number of crossover points for each candidate parent, because once a subtree is selected in a parent, the range of subtrees in another parent is restricted due to the reason that the selected subtree has to fit into the size limitation criteria after crossover, and vice versa. So, when N exceeds a level in GP system, there must be the same children in a brood. We call this point "brood-diversity" point. Once N exceed this number, no matter how big N is, GP with brood recombination crossover will not help much, because two parents can produce only limit different children. That is to say, if the brood size exceeds this limitation, all other children are just the duplication of those different ones.

8.3.3 Effect of Brood-diversity Point

This brood-diversity point depends on different tasks and the different parameters selected while running GP system. In our experiments, we have limited the program size for the three tasks with maximal depth of 5 (for *Shape*, 6 for*Coin*) and 8 (for *Texture*) individually, the results in Table 8.1 and curves in Figure 8.2 show that GP with brood recombination crossover for tasks *Shape* gets very satisfactory classification accuracy while N reach 6, after that the accuracy has no improvement, although its convergence speed continuously get improved but slowly; GP with brood recombination crossover for tasks *Coin* gets very satisfactory classification accuracy while N is up to 8 - 10, after that the accuracy has almost no improvement, although its convergence speed still get improved; for task *Texture*, GP with brood recombination crossover gets the best accuracy and also satisfactory convergence speed while N is around 15, after that both the accuracy and convergence speed get worse. This reveals that the value of brood-diversity point is related to program size. In general, the bigger the program size, the higher the brood-diversity point.

We argue that the brood-diversity point must not be the maximal number of different children that two program can produce by crossover operation. Instead, it should be less than that. For example, two parents can produce maximally 12 different children in theory, but in practice, they might already have duplicated children in a brood of size 8 due to randomness of selection of crossover point. An extreme situation is two parents have same pair of children with brood size 4. So the brood-diversity point for each crossover operation also random and should shift in a range. The bigger the program size, the bigger this randomness, the bigger the range of the brood-diversity point. This could be the reason for task *Texture* (which is run with maximal depth of 8) why GP with brood recombination crossover gets most unstable performance improvement, whereas it gets most stable performance improvement for task *Shape* (which is run with maximal depth of 5).

Furthermore, in our GP system, it allows one candidate program to be selected more than once to make the crossover operation in one generation, thus GP with brood recombination crossover has a potential shortage: reduce the generations' diversity. When one program crossovers with other programs, the possibility to produce the same best child in different crossover operation is getting increased, along with N getting bigger. This will also affect the performance and make the operator highly task specific (e.g. depending on the number of feature), because once the diversity is decreased, although the convergence of evolution process is getting fast, the effectiveness is getting no change. This can explain why, for some tasks (e.g. *Shape* and *Coin*), while their N exceed the brood-diversity point, the convergence speed continuously improved, but the classification accuracy does not further improved.

8.4 Experimental Results and Analysis on Varied Brood Size

To be consistent with previous chapters and the section above, the results on each task are still the average values based on 80 runs with different random seeds in this section.

8.4.1 Four Functions We Used

As proposed in Section 8.1, when the building blocks got larger, the probability of destructive crossover would increase, this suggests the dynamic form of brood recombination crossover— the brood size should grows as evolution proceeds to protect larger building

block from disruption. We examine this idea by a second series of experiments in this section.

We propose two dynamic ways to grow brood size — linearly and non-linearly. We formed the following three relations, in which the first one is linear relation, and the other two are non-linear ones:

$$N(gen) = A1 * gen + B1$$

$$N(gen) = A2 * gen^2 + B2$$

and

$$N(gen) = A3 * \sqrt{gen} + B3$$

where gen is the index of the generation in evolution process

In last section, we have examined the fixed N with different values. For convenience of comparison, we let B1, B2 and B3 be 4, thus we have N equal to 4 at generation 0 for all dynamic ways above so that we can compare them with fixed N with value 4. We assign A1 with 0.4 and 1.0 respectively. When A1 = 0.4, we have the function:

$$N(gen) = 0.4 * gen + 4 \tag{8.1}$$

where N starts with 4 at generation 0 and ends with 25 at generation 50;

When A1 = 1.0, we have the function:

$$N(gen) = 1.0 * gen + 4 \tag{8.2}$$

where N starts with 4 at generation 0 and ends with 54 at generation 50.

To be comparable, we assign A2 with 0.0084 and we get function:

$$N(gen) = 0.0084 * gen^2 + 4 \tag{8.3}$$

where N starts with 4 at generation 0 and ends with 25 at generation 50, which is consistent with the linear function 8.1.

To be comparable, we assign A3 with 2.97 and we get the function:

$$N(gen) = 2.97 * \sqrt{gen} + 4.0 \tag{8.4}$$

where N starts with 4 at generation 0 and ends with 25 at generation 50, which is consistent with the first linear function and the square function 8.1.

These relations between N and gen we examined are summarised in Figure 8.3.

8.4.2 Results and Analysis

The experimental results are collected with the results of fixed N with 4 and basic GP approach together in Figure 8.2 for the convenience of comparison.

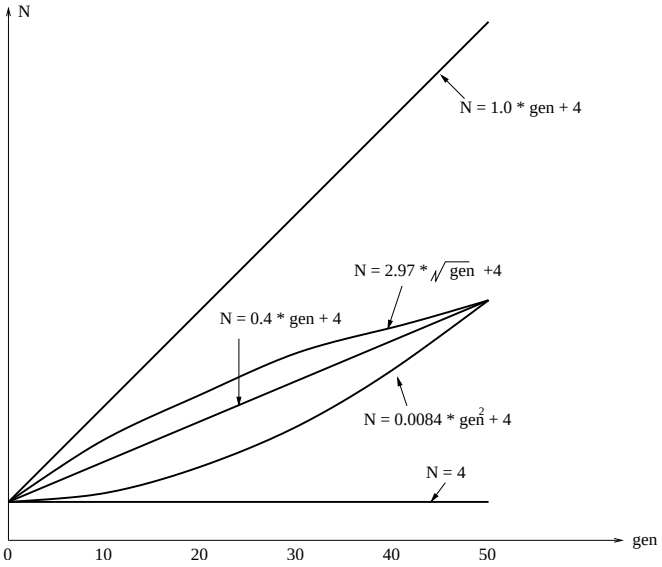


Figure 8.3: Dynamic N vs fixed N

Table 8.2: Results comparison : basic GP vs brood recombination crossover with fixed N (N=4) vs brood recombination crossover with dynamic N on the *Shape*, *Coin* and *Texture* datasets

Dataset	Approach	Generation	Time(s)	Accuracy(%)
Shape	Basic (N=1)	8.59	0.09	96.16%
	BR (fixed N=4)	5.26	0.10	98.25%
	function 8.1: N=0.4*gen+4	4.40	0.09	98.62%
	function 8.2: N=1.0*gen+4	4.35	0.10	98.97%
	function 8.3: $N = 2.97 * \sqrt{gen} + 4$	4.54	0.12	97.38%
	function 8.4: $N = 0.0084 * gen^2 + 4$	4.99	0.09	98.00%
Coin	Basic (N=1)	28.64	1.78	90.37%
	BR (fixed N=4)	21.88	2.50	92.42%
	function 8.1: N=0.4*gen+4	20.59	3.44	92.91%
	function 8.2: N=1.0*gen+4	19.26	5.25	93.17%
	function 8.3: $N = 2.97 * \sqrt{gen} + 4$	20.59	4.71	93.23%
	function 8.4: $N = 0.0084 * gen^2 + 4$	23.26	3.36	92.87%
Texture	Basic (N=1)	29.99	1.83	72.45%
	BR (fixed N=4)	26.01	3.31	76.68%
	function 8.1: N=0.4*gen+4	21.93	4.63	80.50%
	function 8.2: N=1.0*gen+4	20.27	6.67	80.56%
	function 8.3: $N = 2.97 * \sqrt{gen} + 4$	21.31	6.11	77.18%
	function 8.4: $N = 0.0084 * gen^2 + 4$	23.88	4.07	76.32%

The Table 8.2 shows that brood recombination with dynamically varied N also outperforms basic GP approach in term of both classification accuracy and number of generations needed to find the best solution. And with the increasing difficulty of the tasks, the improvement is getting more obvious.

Linear Functions with Different Slope

The two linear functions have different slope. function 8.2 with slope of 1.0 is slightly better than function 8.1 with slope of 0.4. This is reasonable, because function 8.2 intends to increase N more quickly than function e:linear1 as evolution proceeds, so it gives more guarantee to preserve the building blocks when they are growing.

Varied N vs Fixed N

Comparing the dynamically varied N with fixed N, depending on different function we used, the improvement is different. In general, the linear functions 8.1 and 8.2 get slightly better performance then fixed N approach for both *Shape* and *Text* dataset in term of both classification accuracy and number of generations needed to find the solution, and clearly get large improvement for task *Texture*. Whereas, non-linear function 8.3 and 8.4 have not got obvious improvement in all cases.

Functions 8.1, 8.3 and 8.4 all have the same starting N at generation 0 and ending N at generation 50. Function 8.1 increases N in a steady speed with the evolutionary process; function 8.3 increases N faster at early stage but slower at later stage; function 8.3, just the opposite, increases N slower at early stage but faster at later stage. Surprisingly, the linear function 8.1 gets the best result in most cases. Particularly, for the most difficult task *Texture*, the difference on classification accuracy is obvious. This is consistent with the implication that building block grows in a way of link-by-link rather than block-by-block, which is proposed at the end of Chapter 7.

However, because we only examine two sorts of functions for non-linear relationship on N(gen): square and square-root, more formulas on N(gen) in the future should be explored to draw a complete conclusion here.

8.5 Chapter Summary

Focussing on giving GP the ability to look ahead when adjusting the beam and the preservation of building blocks, GP with brood recombination crossover outperforms the basic GP. In general, the more difficult the task is, the more the performance improvement gets.

Increasing the brood size (2N) can improve the performance before N reaches the brood-diversity point. When N exceeds the brood-diversity point, the performance improvement is not guaranteed and sometimes gets unstable.

The improvement by brood recombination is fairly task specific, usually depending on experimental parameters, such as number of features selected, program size limitation, etc.

Brood recombination with dynamic varied N slightly outperforms brood recombination with fixed N. Linearly increasing N along with generation is a good way to vary N.

More formulas on N(gen) need to be investigated in the future to reveal how building block grows as the evolution proceeds. The relation between the number of fitness cases used for evaluation broods and the performance improvement is also left to be examined in the future.

Chapter 9

Conclusions

In this project, we have investigated the various possibilities for improving GP performance on the three tasks of increasing difficulty. In applying the basic GP approach, we mainly focus on the selection and extraction of "powerful" features. In other attempts, we made our efforts on improving the basic GP crossover operators. We have added our voice to the crossover-is-macromutation theory by the investigation of headless chicken crossover on our tasks, developed our new crossover operator — eLCC and examined the effect on the performance by brood size for brood recombination crossover.

9.1 Comprehensive Comparison

We collect the experiments which we have done when we applied the basic GP approach to our three object classification of tasks and are the base for investigation of other improved approaches. We put their results in the Table 9.1 for a comprehensive comparison.

Table 9.1: Best experimental results of each approach on the *shape*, *coin* and *texture* datasets

Dataset	Approach	Generation	Time(s)	Accuracy(%)
Shape	Basic GP approach	8.59	0.09	96.16%
	GP with headless chicken crossover	2.29	0.49	98.03%
	GP with eLCC	4.21	0.31	97.88%
	GP with brood recombination crossover	4.35	0.10	98.97%
Coin	Basic GP approach	28.64	1.78	90.37%
	GP with headless chicken crossover	20.14	42.16	92.37%
	GP with eLCC	19.96	9.23	92.74%
	GP with brood recombination crossover	19.26	5.25	93.17%
Texture	Basic GP approach	29.99	1.83	72.45%
	GP with headless chicken crossover	19.23	71.40	74.78%
	GP with eLCC	22.50	12.84	77.18%
	GP with brood recombination crossover	20.27	6.67	80.56%

Clearly, three advanced approaches outperform the basic GP approach, particularly for hard tasks.

Amongst the three advanced approaches, the GP with brood recombination crossover gets best performance. GP with brood recombination crossover simulates the observed fact

in the real world that many animal species breed far more offspring than needed to survive. It therefore mimics the biological homologous crossover in a hidden way. In another word, brood recombination crossover, preserves the homology implicitly. Brood recombination crossover, in a certain sense, eliminates the disruptive effect of crossover as well, because the child with highest fitness is more likely made from constructing a building block rather than tearing a building block apart.

Our newly developed crossover operator — eLCC also gets an impressive performance. eLCC focuses on selecting crossover points by their looseness values, and thus mainly makes effort on preventing the good building block from being torn apart. Our first try of this controlled selection of crossover points has not considered much about biological homology. This leaves a big room for further improvement in this direction. For example, combine this approach with other proposed homologous crossover.

Headless chicken crossover slightly improves the performance. GP with headless chicken crossover combines the beam search between generations and the hill climbing search inside of a population. The improvement of performance is considered as being from the hill climbing effort rather than preserve the building block and homology, because it turns off the crossover between the existing members in a generation. Additionally, it is an extremely time-consuming methodology because it climbs the hill in a "blind" or "headless" way.

9.2 Conclusions

For image recognition problems, the basic GP approach is usually good for relatively easy tasks but not satisfactory for hard tasks. Adding features to GP terminal set will help in improving GP performance, if the features can capture different aspects or characteristics of the objects classified. When we select and extract more "suitable" features for GP terminal set, the performance can be improved, but it is not enough for some tough tasks.

Headless chicken crossover turns off the primary GP crossover and turns on hill climbing sort of mutation. It is still in the scale of macromutation. The fact it works as well as or slightly better than basic GP approach, from another perspective, proves the traditional GP crossover primarily acts as macromutation. It cuts off the possibility of communication between existing individuals in a population, thus it is not a homologous crossover. The GP bloating effect still exists as the evolution proceeds. This is a clue that the performance improvement comes from hill climbing effort rather than eliminating the disruptive effect of crossover.

Looseness value on a link is used for presenting how "sticky" two nodes at each end of the link should be. It provides a way to identify a good building block from other parts in a tree program. eLCC combines hill climbing search strategy for manipulation of looseness values as evolution proceeds and the techniques of selective crossover points using looseness value. eLCC focuses directly on eliminating the disruptive effect of traditional GP crossover and preserving building block. It is a new approach for improving the basic GP crossover, with a big room for further improvement in this direction.

Brood recombination crossover implicitly preserves the building block and homology. The brood size is a key parameter which effects the performance improvement. We find that there is a brood-diversity point in GP with brood recombination crossover. However, this point seems task specific. It is also related to experimental parameters, such as program limitation criteria and number of features (size of terminal set). The performance can greatly be improved before brood size increases up to the brood-diversity point. When

brood size exceeds the brood-diversity point, the classification can not be improved or get unstable. The brood recombination crossover implemented in *our* GP system has the shortage of decreasing the population diversity, which could be one reason for the observed fact that convergence is still getting fast but classification accuracy stops to improve after brood size reaches the brood-diversity point. The dynamic varied brood size is aiming to preserve building blocks as much as possible when they grow. The dynamic varied brood size does improve the performance. The linear mode, in which the brood size increases along with evolution proceeds, is a best one observed by us so far.

9.3 Future Work

In this project, we have tried to improve the performance of GP system mainly by means of feature selection/extraction and modifying the traditional crossover operator. This left many works worthy to attempt in the future.

During applying the basic GP approach to relatively hard task, such as texture, we only tried part of advanced features. It is almost sure that there are still a large bunch of other features that can improve GP performance for that task, such as the features that capture energy information of an image appearance. Also, various transformations which transform the features from spatial domain to spectral domain are worthy to try for performance improvement for *texture* task.

In eLCC approach, we use hill climbing search to enforce that a better building block has been constructed or grown around the crossover point. But this is still not an ideal way, although promising. In fact, a child program with a better fitness than its parent can be resulted from other reasons, such as a bad building block is put into a very hospitable context. We are still on the way to find more guaranteed way to manipulating and updating the looseness values. We need to try assigning and updating the looseness values not only according to the whole program's fitness but also other factors, such as evaluation on subtrees or evaluation of how similar the two crossovered parts are. These evaluation methods also need to be explored.

When we investigate GP with brood recombination crossover, we only tried very simple non-linear functions by which the dynamic varied brood size increases as evolution proceeds. That is not enough to support any conclusion about the way in which building blocks grow or what is the function that describes the relation between brood size and index of generation best, more sophisticated functions might be examined, such as exponent or logarithm functions. Additionally, whether the size of training set used in evaluation children in broods affect the performance and how big the effect will be is another work in the future.

It is obvious that preserving building block will help in preserving homology, and vice versa. They are related but still two independent aspects in improving crossover. Preservation of good building block is more syntax-like or structural problem, whereas preservation of homology is more semantics-like or contextual problem. Finding the balance between them or finding which one is predominant issue, is another big research topic left for GP community.

Bibliography

[1] ALTENBERG, L. Emergent phenomena in genetic programming. In Evolutionary Programming — Proceedings of the Third Annual Conference (1994), A. V. Sebald and L. J. Fogel, Eds., World Scientific, Singapore. 1994.

[2] BAMZJAF, W. Genetic programming for pedestrians. Tech. rep., MERL Technical REport 93-03, Mitsubishi Electric Research Labs, Ca, 1993.

[3] BANZHAF, W., FRANCONE, F. D., AND NORDIN, P. The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation* (Berlin, Germany, 22-26 Sep. 1996), H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds., vol. 1141 of *LNCS*, Springer Verlag, Heidelberg Germany, pp. 300–309.

[4] BANZHAF, W., NORDIN, P., KELLER, R. E., AND FRANCONE, F. D. *Genetic Programming - An Introduction*. Morgan Kaufmann Publishers, Inc. and dpunkt - Verlag fur digitale Technologie GmbH, Printed in the United States of America, 1998.

[5] FRIEDBERG, R. A learning machine, part I. *IBM J. Research and Development 2* (1958), 2–13.

[6] FUCHS, M., FUCHS, D., AND FUCHS, M. Solving problems of combinatory logic with genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference* (Stanford University, CA, USA, 13-16 Jul. 1997), J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Eds., Morgan Kaufmann, San Francisco, CA, USA, pp. 102–110.

[7] Webpage: `http://www.genetic-programming.org/`.
accessed on 25 July, 2004.

[8] Webpage:
`http://www.genetic-programming.com/gpanimatedtutorial.html`. by Genetic-Programming Inc.
accessed on 25 July, 2004.

[9] IBA, H., AND DE GARIS, H. Extending genetic programming with recombinative guidance. In *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. MIT Press, Cambridge, MA, USA, 1996, ch. 4, pp. 69–88.

[10] KOZA, J. R. Genetic programming: On the programming of computers by natural selection. *MIT Press, Cambridge, MA.* (1992).

[11] LANG, K. J. Hill climbing beats genetic search on a boolean circuit synthesis of koza's. In Procedings of the Twelfth International Conference on Machine Learning (1995). Tahoe City, CA. Morgan kaufmann, San Francisco, CA. 1995.

[12] MITCHELL, T. *Machine Learning*. McGrawHill, New York, 1996.

[13] NORDIN, P., AND BANZHAF, W. Complexity compression and evolution. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)* (Pittsburgh, PA, USA, 15-19 Jul 1995), L. Eshelman, Ed., Morgan Kaufmann, San Francisco, CA, USA, pp. 310–317.

[14] NORDIN, P., FRANCONE, F., AND BANZHAF, W. Explicitly defined introns and destructive crossover in genetic programming. In *Advances in Genetic Programming 2*, A. P. J. and J. K. E., Eds. MIT Press, 1996.

[15] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical Recipes in C — The Art of Scientific Computing*, second ed. Cambridge University Press, 1992.

[16] RUSSELL, S. J., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, second ed. Prentice Hall, Inc. Pearson Education, Inc., Upper Saddle River, New Jersey 07458, 2003.

[17] SMART, W., AND ZHANG, M. Applying online gradient descent search to genetic programming for object recognition. Tech. rep., CS-TR-03/13, Victoria University of Wellington, 2003.

[18] SMART, W., AND ZHANG, M. Multiclass object classification using cenetic programming. Tech. rep., CS-TR-04/2, Victoria University of Wellington, 2004.

[19] SMART, W., AND ZHANG, M. Probability based genetic programming for multiclass object classification. Tech. rep., CS-TR-04/7, Victoria University of Wellington, 2004.

[20] TACKETT, W. A. *Recombination, Selection and the Genetic Construction of Computer Programs*. PhD thesis, University of Souithern California, Department of Electrical Engineering Systems, 1994.

[21] Webpage:
`http://sipi.usc.edu/services/database/database.cgi?volume=textures`. by Signal & Image Processing Institute of University of Southern California.
accessed on 22 July, 2004.

[22] ZHANG, M. Lecture note comp422: Data mining, neural nets, and genetic programming. Victoria University of Wellington, New Zealand, Jun.- Sep. 2004.

[23] ZHANG, M., ANDREAE, P., AND BHOWAN, U. A two phase genetic programming approach to object detection. Tech. rep., CS-TR-04/6, Victoria University of Wellington, 2004.

[24] ZHANG, M., AND BHOWAN, U. Pixel statistics and program size in genetic programming for object detection. Tech. rep., CS-TR-04/3, Victoria University of Wellington, 2004.

[25] ZHANG, M., AND SMART, W. Genetic programming with gradient descent search for multiclass object classification. Tech. rep., CS-TR-04/4, Victoria University of Wellington, 2004.

[26] ZHANG, M., AND VICTOR, C. Genetic programming for multiple class object detection. In *12th Australian Joint Conference on Artificial Intelligence —AI'99* (1999), LNAI 1747, Department of Comuter Science, Royal Melbourne Institute of Technology, Springer-Verlag Berlin Heidelerg, pp. pp.180–192.