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

School of Mathematics, Statistics and Computer Science
Computer Science

A New Crossover Operator in GP for Object Classification

Mengjie Zhang, Xiaoying Gao, Weijun Lou

Technical Report CS-TR-06/2
January 2006

School of Mathematics, Statistics and Computer Science
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

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

School of Mathematics, Statistics and Computer Science
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

A New Crossover Operator in GP for Object Classification

Mengjie Zhang, Xiaoying Gao, Weijun Lou

Technical Report CS-TR-06/2
January 2006

Abstract

This paper describes a new crossover operator in genetic programming for object recognition particularly object classification problems. In this approach, instead of randomly choosing the crossover points as in the standard crossover operator, we use a measure called *looseness* to guide the selection of crossover points. Rather than using the genetic beam search only, this approach uses a hybrid beam-hill climbing search scheme in the evolutionary process. This approach is examined and compared with the standard crossover operator and the headless chicken crossover method on a sequence of object classification problems. The results suggest that this approach outperforms both the headless chicken crossover and the standard crossover on all of these problems.

Keywords Crossover points, looseness controlled crossover, genetic programming, hybrid search

Author Information

All authors are academic staff members and postgraduate students in computer science in the School of Mathematics, Statistics and Computer Science, Victoria University of Wellington, New Zealand.

1 Introduction

Classification tasks arise in a very wide range of applications, such as detecting faces from video images, recognising words in streams of speech, diagnosing medical conditions from the output of medical tests, and detecting fraudulent credit card fraud transactions [4, 9, 6, 23, 18]. In many cases, people (possibly highly trained experts) are able to perform the classification task well, but there is either a shortage of such experts, or the cost of people is too high. Given the amount of data that needs to be classified, automatic computer based classification programs/systems are of immense social and economic value.

A classification program must correctly map an input vector describing an instance (such as an object image) to one of a small set of class labels. Writing classification programs that have sufficient accuracy and reliability is usually very difficult and often infeasible: human programmers often cannot identify all the subtle conditions needed to distinguish between all instances of different classes.

Derived from genetic algorithms and evolutionary programming [7, 15], Genetic programming (GP) is a relatively recent and fast developing approach to automatic programming [3, 11, 12]. In GP, solutions to a problem can be represented in different forms but are usually interpreted as computer programs. Darwinian principles of natural selection and recombination are used to evolve a population of programs towards an effective solution to specific problems. The flexibility and expressiveness of computer program representation, combined with the powerful capabilities of evolutionary search, make GP an exciting new method to solve a great variety of problems. A strength of this approach is that evolved programs can be much more flexible than the highly constrained, parameterised models used in other techniques such as neural networks and support vector machines.

Since the early 1990s, there has been a number of reports on applying genetic programming techniques to a range of object recognition problems such as shape classification, face identification, and medical diagnosis [1, 8, 14, 19, 21, 24, 27, 26, 28]. While showing promise, current GP techniques are limited and frequently do not give satisfactory results on difficult classification tasks. There are many aspects of the design of a GP approach to a specific problem that could be addressed and lead to improvement — in this paper we choose to explore the crossover operator.

The crossover genetic operator has been considered a centre storm of genetic programming [2]. In the current crossover operator, two sub-programs (crossover points) are randomly chosen from two parent programs, and two new programs are generated by simply swapping them. However, the totally random choice is clearly unable to guarantee the best choice. More importantly, this random choice often destroys the good "building blocks" (sub-programs which are good for that task) in evolved programs, particularly in the later stage of evolution [2]. A better way of performing crossover needs to be investigated.

1.1 Related Work

Crossover is the predominant search operator in most GP systems [29, 25, 2]. In Genetic Algorithms, the building block hypothesis [5] shows that there are good building blocks which can improve the fitness of individuals. In GP, however, past research has shown this is not always true.

The famous Gedanken experiment [3] empirically investigated the relationship between the GP crossover operator and the building blocks in a genetic program. The results suggest that there might be two different phases in a GP run. In the first phase, the building blocks are relatively small and the probability of the building blocks being disrupted by crossover is also small, where the evolution completely depends on fitness. In the second phase, the

building blocks become large and the probability of building blocks being disrupted also becomes large, where the evolution depends on both fitness and the structure of individual programs. Accordingly, the crossover operator mainly preserves small building blocks and combines them into larger blocks at the early stage of evolution, then starts tearing the larger building blocks apart at the later stage of evolution.

Nordin and Banzhaf [16, 17] investigated the effect of crossover on the relative fitness of parent programs to their offspring on tree bases GP systems. The results show that the average fitness of the child programs is less than half the fitness of the parents for about 75% of all crossover events. It seems that the standard crossover operator has an overwhelmingly negative effect.

Lang [13] launched 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 on the problems he studied. While the results were only based on the Boolean 3-multiplexer problem where no local minima exist and the above conclusion was not necessarily true for other problems, the experiments clearly showed that the standard crossover operator was not sufficiently effective.

To improve the standard crossover operator, Tackett [20] introduced the "brood recombination" method. In this method, a "brood" N was created for each crossover operation and the operation was repeated N times to produce $2N$ child programs, but only the best two children were kept and all others were killed. The rationale behind the method is an analogy of the observed fact that many animal species produce offspring far more than those expected to live. This approach focused on making better crossover by trying more times, but it did not consider heuristics or intelligent selection of the crossover points.

Iba and Garis [10] added some heuristics to the standard crossover operator to make the operator "smart". This crossover evaluates performance for subtrees. This approach assumes that a good building block can be built up, but after that, the good block might be moved into a new individual that does not treat it as a good building block.

Banzhaf, et al. [3] proposed a homologous crossover operator. This crossover operator measures the structural similarity and functional similarity of each subtrees by some distances between two programs. The two measurements are interpreted as probabilities and are then used to select crossover points. The computational cost of this approach was quite high.

1.2 Goals

The goal of this paper is to investigate a different approach to improving the crossover operator in tree based GP for object classification problems. This approach will be based on the selection of good crossover points to preserve potential building blocks and use a hybrid beam-hill climbing search to improve the system performance. This approach will be examined and compared with the standard crossover operator and the headless chicken crossover operator on three object classification problems of increasing difficulty. Specifically, we are interested in investigating the following issues.

- how to use a kind of "looseness" on the links between every two nodes in a program tree to help select good crossover points,
- whether this approach can do a good enough job on these object classification problems,
- whether this approach outperforms the standard crossover operator on the same sequence of problems, and

- whether this approach outperforms the headless chicken crossover operator on these problems.

1.3 Organisation

The remainder of the paper is organised as follows. Section 2 briefly overviews the standard crossover operator and the headless chicken crossover operator. Section 3 describes the new looseness control crossover operator and the hybrid beam-hill climbing search in the operator. Section 4 presents the experiment design and configurations. Section 5 presents and analyses the experiment results. Section 6 draws conclusions and gives future work directions.

2 Overview of Standard GP Crossover and the Headless Chicken Crossover

2.1 The Standard GP Crossover Operator

To perform the standard crossover operator in tree based genetic programming, two genetic programs are selected from the population and are put into the mating pool. A crossover point is then *randomly* chosen in each of the two parent programs, then the subtrees below the crossover points are swapped and two new (child) programs are created. The two child programs are then put into the population in the next generation.

This operator aims to have a chance of combining aspects of two programs into one. From biological point of view, it is expected to mimic the biological sexual mating behaviour to take the advantages of different programs and integrate the useful attributes in order to form better programs for a particular task. However, the totally random selection of crossover points causes a number of problems. A major problem is that this random selection can result in good "genes" or "building blocks" to be destroyed, which could deteriorate the performance and even lead to "illegal" solutions for some particular tasks.

2.2 The Headless Chicken Crossover Operator

Headless chicken crossover [13] is a kind of macromutation. As shown in figure 1, only *one* parent A is selected and an entirely new individual B_1 is *randomly* generated. The selected parent A is then crossed over with the new and randomly created individual B_1. The offspring C_1 is kept if it is better than the parent A measured in fitness. Otherwise, it is discarded and another entirely new individual B_2 is *randomly* generated, and a new crossover occurs between A and B_2. The iteration goes on until the offspring C_n is better than the parent A in fitness. C_n is then kept and put in the next generation. Thus, headless chicken crossover uses a form of hill climbing search.

3 Looseness Control Crossover

To improve the effectiveness of the GP crossover operator, in this approach, we introduce a weight called *looseness* to control the choice of the crossover points and combine the hill climbing search scheme into the genetic beam search in GP. The main idea and the method are presented in the rest of this section.

3.1 Main Idea

To avoid randomly selecting the crossover points, we give each link between two adjacent nodes in a program tree a value, *looseness*, to reflect how "sticky" the two nodes should be

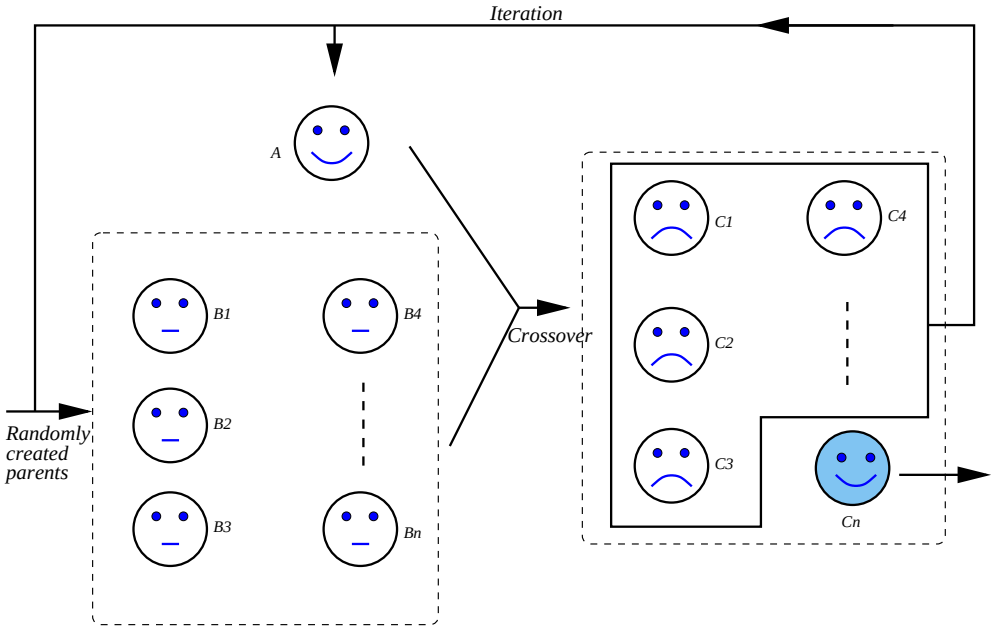


Figure 1: The main idea of headless chicken crossover.

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 to each other. The smaller the looseness value, the better the block. Accordingly, when we select the point for crossover, only the link with the biggest looseness value in the tree will be selected. If two or more links have the same biggest looseness value, we randomly choose one from them.

In this approach, we assign a looseness value to each link in the program tree at the initial generation before evolution. This looseness value will be updated during the evolutionary process based on the results of the crossover operator — a good building block is more sticky than other parts of the program tree which have higher looseness values. By selecting the crossover points only on those loosest links, we expect that the disruptive effect of crossover can be eliminated or at least the disruption possibility can be reduced.

3.2 The Methodology

As shown in Figure 2, at generation 0, since no good building blocks have been recognised (even though there might exist some), we treat all links equally: all links in the program trees are assigned 0 as the looseness value. Thus, the selection of the crossover point are random at the first round evolution. Once crossover happens between two nodes, supposing that the new link is better than the old one based on the child fitness evaluation, we will keep the link associated with the crossover 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 the other links will have the same number as the index of generation, as shown in figure 3. In the next round of evolution, this link will not be selected as the crossover point, and the two nodes with the link are considered a small good building block. As the evolutionary process goes on, more and more nodes are getting sticky and the good building blocks grow and connect with each other. In case all links have an equal looseness value of $gen-1$ where gen is the index of current generation, a new loop of further

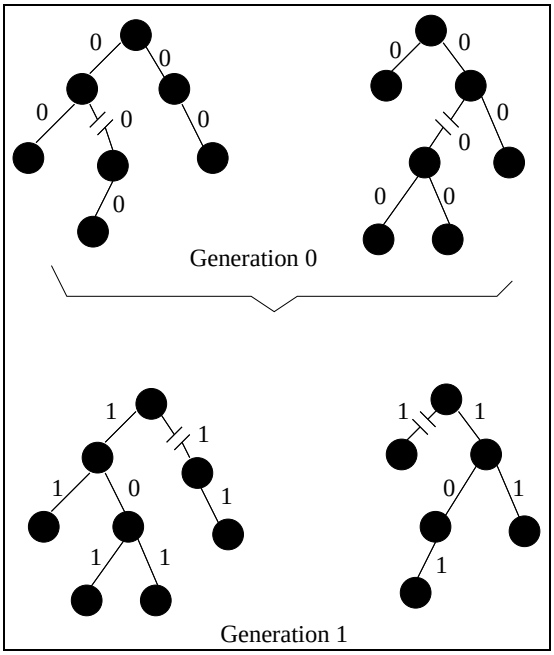


Figure 2: The main idea of looseness-controlled crossover.

reducing the looseness down starts. We expect that the final solution program is itself a good building block as a whole and have a fairly small looseness value and that this approach can mimic the biological gene crossover point selection and improve the effectiveness of the crossover operator.

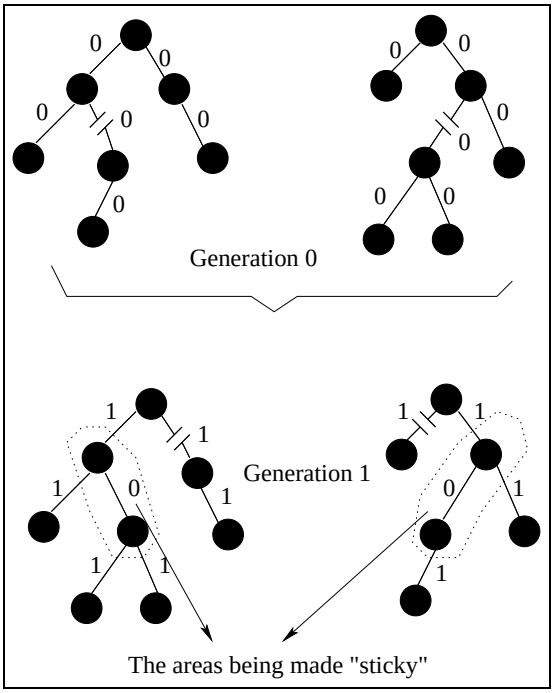


Figure 3: Sample “sticky” areas in the looseness-controlled crossover.

3.3 Further Improvement — Hybrid Search

The above method is based on the assumption that the new programs after crossover have better building blocks around crossover points than their parents. In the actual situation, however, this is often not true. For example, a good building block in one solution can become the source of bad performance of another solution. Figure 4 shows one case of this possibility. Before the crossover happens, the program has two good building blocks. It is expected one of them to grow by crossover to make a larger good building block. After the crossover, although it replaces 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. So we need to check whether the larger area is a good grown building block or not. To guarantee the newly generated programs by crossover are better than the old programs, we introduce a hill climbing search into the genetic beam search. The main idea is presented as follows.

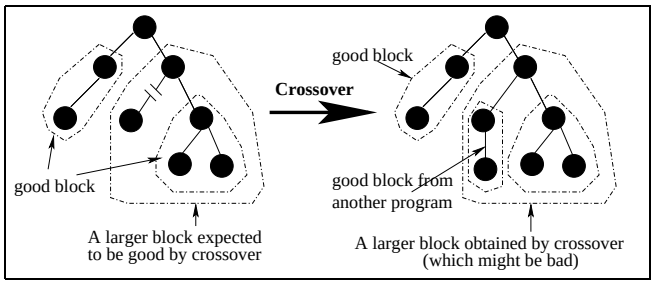


Figure 4: Illustration of building block growing and a sample large building block which might not be good.

To make sure that a child program contain a better building block around the crossover point than its parents, we evaluate these programs and compare their fitness values. If the child program has better fitness than its parents, we treat this crossover operation as a *successful* one, and a larger good building block is treated as formed. Otherwise, we treat this crossover operation as *failed*. The failed crossover operation is undone and a new crossover operation is attempted. This process continues until the offspring has better fitness than its parent. In this way, we expect that the building block around the crossover point has been growing or has been getting better. For presentation convenience, we call this approach *Looseness Controlled Crossover*, LCC for short.

3.4 An Example

As shown in Figure 5, originally, two parent programs A and B_1 are selected and their crossover points a and b_1 are chosen according to the looseness value on each link. Then crossover operation occurs by replacing the subtree at a with the subtree at b_1, and we get a child C_1. We then evaluate C_1 by compare its fitness with its parent A's. If the program C_1 is better than A, then we update the looseness value of program C_1; otherwise, we "undo" the crossover operation by deleting the new program C_1. Then we find another parent program B_2 from the population, and try a new crossover operation with the first parent program A and check whether the newly generated program C_2 is better than program A or not. This process will continue in the form of loop until we find a child program with better fitness, such as C_n, and we update the looseness values of program C_n. Notice that the crossover operator will produce two child programs and we check both of them and update the looseness values accordingly.

In theory, the number of the iteration n can be very large even infinite, particularly for

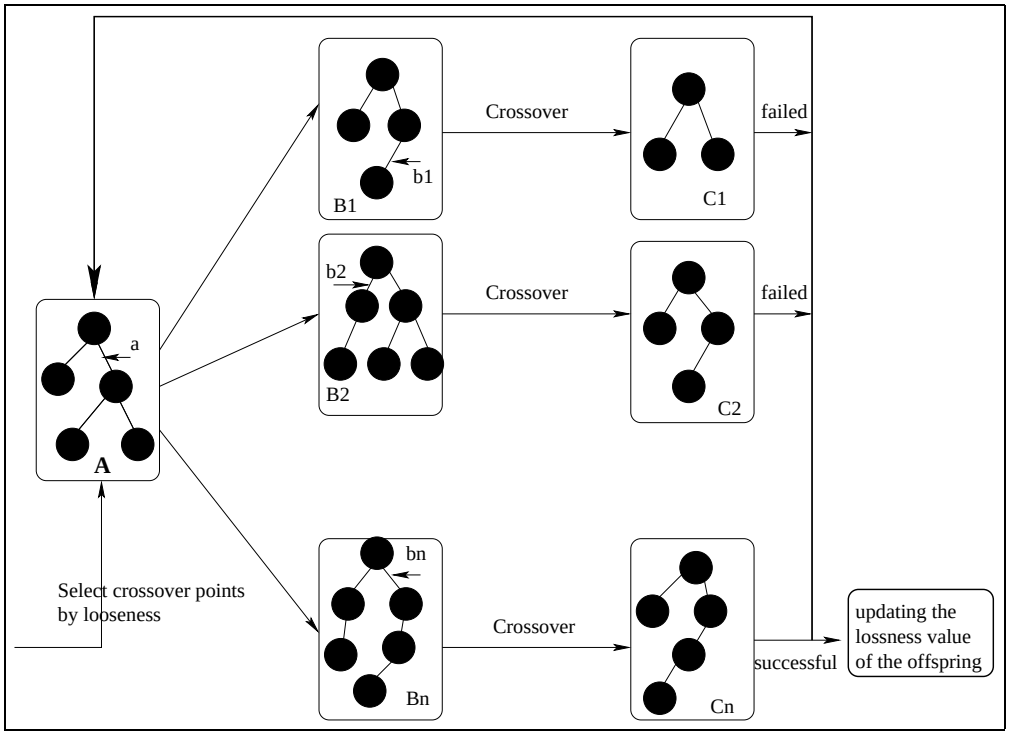


Figure 5: Illustration of looseness-controlled crossover.

the case that a successful crossover can never be found due to the reason that the candidate program (such as A in figure 5) as a whole is already a very good building block and no other suitable program in the current population can make it better by crossover. To avoid such infinite loops and for efficiency reasons, we limit this number to 1000, i.e. when the iteration has been executed 1000 times and still no better child program can be found, we terminate the iteration and elite the parent program and put a copy into the next generation.

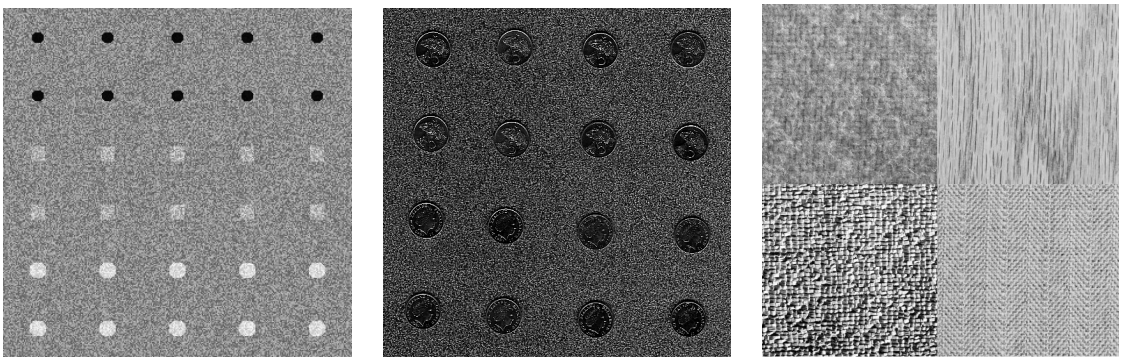
4 Experiment Configurations

4.1 Image Data Sets

Experiments were conducted on three different image data sets providing object classification problems of increasing difficulty. Sample images for each data set are show in figure 6.

The first data set (figure 6a) was generated to give well defined objects against a reasonably noisy background. The pixels of the objects were produced using a Gaussian generator with different means and variances for each class. Four classes of 600 small objects (150 for each class) were cut out from the images and used to form the classification data set. The four classes are: *dark circles, grey squares*, *light circles* and *noisy background*. For presentation convenience, this data set is referred to as *shape*.

The second set of images (*coin*, figure 6b) contains scanned 10 cent New Zealand coins. The coins were located in different places with different orientations and appeared in different sides (head and tail). The background was also cluttered. Three classes of 500 objects were cut out from the large images to form the data set. The three classes are: *head*, *tail* and *background*. Among the 500 cutouts, there are 160 cutouts for *head*, 160 cutouts for *tail* and 180 cutouts for *background* respectively. Compared with the *shape* data set, the classification problem in this data set is harder. Although these are still regular, man-made objects, the problem is quite hard due to the noisy background and the low resolution.



(a) (b) (c)

Figure 6: Sample images in datasets: (a) Shape; (b) Coins; (c) Texture.

The third set of images (figure 6c) contains four different kinds of texture images, which are taken by a camera under the natural light. The images are taken from a web-based image database held by SIPI of USC [22]. The four texture classes are named *woollen cloth*, *wood grain*, *raffia* and *herringbone weave* respectively. Because they are quite similar in many aspects, this classification task is expected to be more difficult than that in the *coin* data set. There are 900 sample cutouts from four large images, and each class has 225 samples. This dataset is referred to as *texture*.

For all the three data sets, the objects were equally split into three separate data sets: one third for the training set used directly for learning the genetic program classifiers, one third for the validation set for controlling overfitting, and one third for the test set for measuring the performance of the learned program classifiers.

4.2 Terminal Set

For image recognition tasks, terminals correspond to image features. In this approach, we use four simple pixel statistics extracted from each data set as terminals. Given an object cutout image, the four pixel statistics, *mean, standard deviation, skewness*, and *kurtosis*, are calculated by the following formulas respectively. Since the ranges of these four feature terminal values are quite different, we linearly scaled them into the range [-1, 1] based on all object image examples to be classified.

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

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

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

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

where X_i is the brightness of the ith pixel in the given object (cutout) image, $\bar{X}$ is the mean value, and σ_X is the standard deviation of all pixels of the object image. While these features are not the best for these particular problems, our goal is to investigate the crossover operators, rather than finding good features for a particular task, which is beyond the scope of this paper.

In addition to the feature terminals, we also used a constant terminal for all the three tasks.

4.3 Function Set

In the function set, the four standard arithmetic and a conditional operation were used to form the function set:

$$FuncSet = \{+, -, *, /, if\} \tag{1}$$

The $+$, $-$, and $*$ operators have their usual meanings — addition, subtraction and multiplication, while / represents "protected" division which is the usual division operator except that a divide by zero gives a result of zero. Each of these functions takes two arguments. The *if* function takes three arguments. The first argument, which can be any expression, constitutes the condition. If the first argument is negative, the *if* function returns its second argument; otherwise, it returns its third argument. The *if* function allows a program to contain a different expression in different regions of the feature space, and allows discontinuous programs, rather than insisting on smooth functions.

4.4 Fitness Function

We used classification accuracy on the training set of object images as the fitness function. The classification accuracy of a genetic program classifier refers to the number of object images that are correctly classified by the genetic program classifier as a proportion of the total number of object images in the training set. According to this design, the best fitness is 100%, meaning that all object images have been correctly recognised.

The output of a genetic program in the standard GP system is a floating point number. In this approach, we used a variant version of the *program classification map* [28] to translate the single output value of a genetic program into a set of class labels. This variation situates class regions sequentially on the floating point number line. The object image will be classified to the class of the region that the program output with the object image input falls into. Class region boundaries start at some negative number, and end at the same positive number. Boundaries between the starting point and the end point of each class (except for the first and last class) are allocated with an identical interval of 1.0. For example, a four class problem would have the following classification map:

$$\textbf{class} = \begin{cases} \text{Class 1}, & r < -1.0 \\ \text{Class 2}, & -1.0 \leq r < 0 \\ \text{Class 3}, & 0 \leq r < 1.0 \\ \text{Class 4}, & 1.0 \leq r \end{cases} \tag{2}$$

where r is the program output.

4.5 Parameters and Termination Criteria

The parameter values used in this approach are shown in table 1. These values are determined based on empirical search.

In this approach, the learning/evolutionary process is terminated when one of the following conditions is met:

- The classification problem has been solved on the training set, that is, all objects of interest in the training set have been correctly classified without any missing objects or false alarms for any class.

Table 1: Main parameter values for the three data sets.

Parameter	Shape	Coin	Texture
Population Size	300	500	500
Initial Max. Depth	5	5	5
Initial Min. Depth	3	3	3
Max. Depth	5	6	8
Min. Depth	3	3	3
Max_generations	50	50	50
Crossover rate:	50%	50%	50%
Mutation rate:	30%	30%	30%
Reproduction rate	20%	20%	20%

- The accuracy on the validation set starts falling down.
- The number of generations reaches the pre-defined number, *max_generations*.

4.6 Experiment Configuration

In the approach, we used the tree-structure to represent genetic programs [11]. The ramped half-and-half method was used for generating programs in the initial population and for the mutation operator [3]. The proportional selection mechanism and the reproduction, crossover and mutation operators [12] were used in the learning and evolutionary process.

To compare the new approach and the approaches with the standard crossover and headless chicken crossover, we use the classification accuracy, training time and the number of generations to measure the performances of these methods. For each experiment, we run 80 times and the average results are presented in the next section.

5 Results and Discussion

Table 2 shows the average results of the genetic programming systems with the three different crossover operators in terms of the classification accuracy, evolutionary training time, and the number of generations used for the evolution.

Table 2: Results of the three crossover operators. (HCC: Headless Chiken Crossover; LCC: Looseness Controlled Crossover)

Data Sets	Method	Accuracy (%)	Time(s)	Generations
Shape	Standand	96.16±1.35	0.09	8.59
	HCC	98.03±1.53	0.49	2.29
	LCC	97.88±0.76	0.31	4.21
Coin	Standard	90.37±2.51	1.78	28.64
	HCC	92.37±2.83	42.16	20.14
	LCC	92.74±1.34	9.23	19.96
Texture	Standard	72.45±3.23	1.83	29.99
	HCC	74.78±4.65	71.40	19.23
	LCC	77.18±2.84	12.84	22.50

As can be seen from table 2, for the shape data set, both the looseness controlled crossover operator developed in this approach and the headless chicken crossover achieved an improve-

ment in classification accuracy, but spent a longer time than the standard crossover operator. In particular, the headless chicken crossover operator used about five to six times longer time than the standard crossover operator. The number of generations spent on the evolutionary process, however, is the smallest one for the headless chicken crossover, which suggests that the evaluation cost per generation for the headless chicken crossover is very high.

For the relatively difficult object classification tasks in the coin and the texture data sets, the new looseness controlled crossover operator resulted in the best classification accuracy among all the three approaches. The improvement of the LCC approach over the other two methods is significant for the most difficult texture data set. While the headless chicken crossover approach also achieved a better accuracy than the standard crossover, its performance was worse than the looseness controlled crossover. In particular, the headless chicken crossover spent a huge amount of time in the evolutionary process, suggesting that it is quite inefficient.

Overall, headless chicken crossover performed slightly better than the standard crossover in terms of the classification accuracy but it is very inefficient. Headless chicken crossover mainly uses a kind of hill climbing search and is kind of macromutation. This is consistent with previous research suggesting that the standard GP crossover operator is only a kind macromutation and could be further improved.

The looseness controlled crossover operator developed in this approach added some sort of "intelligence" and the crossover points selection is more "clever" than in the standard crossover operator. The looseness controlled crossover also used a hybrid beam-hill climbing search scheme, and it seemed that it used the advantages from both the standard crossover and the headless chicken crossover. Accordingly, this method achieved a markedly improved accuracy particularly for the more difficult tasks than the standard crossover operator, and is much more efficient than the headless chicken crossover operator.

6 Conclusions

The goal of this paper was to develop a more intelligent crossover operator than the standard crossover operator in genetic programming for object recognition problems. The goal was successfully achieved by introducing a value called "looseness" on the link between every adjacent nodes in the program tree, using this looseness to guide the selection of crossover points and using a hybrid beam-hill climbing search scheme to guarantee good building blocks to be achieved by crossover. This approach was examined and compared with the standard GP crossover operator on three object classification problems of increasing difficulty. The results suggest that this new approach outperformed the standard GP crossover operator in terms of the classification accuracy. While it spent a bit longer time, this is a small price to pay for many object recognition problems.

This approach was also examined and compared with the headless chicken crossover operator on the same data sets. The results suggest that the new looseness controlled crossover operator can achieve better classification accuracy on most data sets particularly the relatively difficult problems than the headless chicken crossover operator, and it is also much more efficient.

Although developed for object classification problems, we expect this approach can be applied to other general classification problems.

While this approach was much more efficient than the headless chicken crossover operator, it spent a longer time than the standard GP crossover operator on these tasks. This is mainly due to the use of the hill climbing search by evaluating more individual programs within a particular generation. We will investigate more efficient ways such as evaluation on some subprograms. It is also interesting to investigate more clever ways for updating the looseness of

links between every two adjacent nodes in the individual program trees in the future.

Acknowledgment

This work was supported in part by the Marsden Fund of New Zealand under grant no. 05-VUW-017 and the University Research Fund 06/9 at Victoria University of Wellington.

References

[1] David Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, pages 477–494. MIT Press, 1994.

[2] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic Programming - An Introduction*. Morgan Kaufmann Publishers, Inc. and dpunkt - Verlag fur digitale Technologie GmbH, Printed in the United States of America, 1998.

[3] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications*. San Francisco, Calif. : Morgan Kaufmann Publishers; Heidelburg : Dpunkt-verlag, 1998. Subject: Genetic programming (Computer science); ISBN: 1-55860-510-X.

[4] J. Eggermont, A. E. Eiben, and J. I. van Hemert. A comparison of genetic programming variants for data classification. In *Proceedings of the Third Symposium on Intelligent Data Analysis (IDA-99), LNCS 1642*. Dpringer-Verlag, 1999.

[5] D.E. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine learning*. Addison Wesley, Reading, Ma, 1989.

[6] H. F. Gray, R. J. Maxwell, I. Martinez-Perez, C. Arus, and S. Cerdan. Genetic programming for classification of brain tumours from nuclear magnetic resonance biopsy spectra. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, page 424, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[7] John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor : University of Michigan Press; Cambridge, Mass. : MIT Press, 1975.

[8] Daniel Howard, Simon C. Roberts, and Richard Brankin. Target detection in SAR imagery by genetic programming. *Advances in Engineering Software*, 30:303–311, 1999.

[9] Daniel Howard, Simon C. Roberts, and Conor Ryan. The boru data crawler for object detection tasks in machine vision. In Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and G"unther Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, volume 2279 of *LNCS*, pages 220–230, Kinsale, Ireland, 3-4 April 2002. Springer-Verlag.

[10] Hitoshi Iba and Hugo de Garis. Extending genetic programming with recombinative guidance. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 4, pages 69–88. MIT Press, Cambridge, MA, USA, 1996.

[11] John R. Koza. *Genetic programming : on the programming of computers by means of natural selection*. Cambridge, Mass. : MIT Press, London, England, 1992.

[12] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, Mass. : MIT Press, London, England, 1994.

[13] K J Lang. 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.

[14] Thomas Loveard and Victor Ciesielski. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1070–1077, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[15] Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK, 1996.

[16] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 Jul 1995. Morgan Kaufmann, San Francisco, CA, USA.

[17] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Angeline Peter J. and Jr. K. E., editors, *Advances in Genetic Programming 2*. MIT Press, 1996.

[18] Riccardo Poli. Genetic programming for image analysis. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 363–368, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[19] Andy Song, Vic Ciesielski, and Hugh Williams. Texture classifiers generated by genetic programming. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 243–248. IEEE Press, 2002.

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

[21] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303–309, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[22] 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.

[23] D. Valentin, H. Abdi, and O'Toole. Categorization and identification of human face images by neural networks: A review of linear auto-associator and principal component approaches. *Journal of Biological Systems*, 2(3):413–429, 1994.

[24] Jay F. Winkeler and B. S. Manjunath. Genetic programming for object detection. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 330–335, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

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

[26] Mengjie Zhang, Peter Andreae, and Mark Pritchard. Pixel statistics and false alarm area in genetic programming for object detection. In Stefano Cagnoni, editor, *Applications of Evolutionary Computing, Lecture Notes in Computer Science, LNCS Vol. 2611*, pages 455–466. Springer-Verlag, 2003.

[27] Mengjie Zhang and Victor Ciesielski. Genetic programming for multiple class object detection. In Norman Foo, editor, *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99)*, pages 180–192, Sydney, Australia, December 1999. Springer-Verlag Berlin Heidelberg. Lecture Notes in Artificial Intelligence (LNAI Volume 1747).

[28] Mengjie Zhang, Victor Ciesielski, and Peter Andreae. A domain independent window-approach to multiclass object detection using genetic programming. *EURASIP Journal on Signal Processing, Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis*, 2003(8):841–859, 2003.

[29] Mengjie Zhang and will Smart. Genetic programming with gradient descent search for multiclass object classification. Technical report, CS-TR-04/4, Victoria University of Wellington, 2004.