# Experiments on Brood Size in GP with Brood Recombination Crossover for Object Recognition

Mengjie Zhang, Xiaoying Gao, Weijun lou

Technical Report CS-TR-06/6
February 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

# Experiments on Brood Size in GP with Brood Recombination Crossover for Object Recognition

Mengjie Zhang, Xiaoying Gao, Weijun lou

Technical Report CS-TR-06/6
February 2006

**Abstract**

To improve the standard crossover operator in genetic programming (GP), Tackett [19] introduced the brood recombination crossover method. The key parameter in this method is the brood size. This paper investigates a number of new developments of brood size in the brood recombination crossover method in GP. We first investigate the effect of different fixed brood sizes, then construct three dynamic models for setting the brood size. These developments are examined and compared with the standard crossover operator on three object classification problems of increasing difficulty. The results suggest that the brood recombination methods with all the new developments outperforms the standard crossover operator for all the problems. As the brood size increases, the system effective performance can be improved. When it exceeds a certain point, however, the effective performance will not be improved and the system will become less efficient. Investigation of three dynamic models for the brood size reveals that a variable brood size which is dynamically set with the number of generations can further improve the system performance over the fixed brood size.

**Keywords** Brood recombination crossover, building blocks, brood size, genetic programming.

## 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 [1, 2, 3, 4]. In many cases, people (possibly highly trained experts) are able to perform the classification task well, but either there is 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 (e.g. an object) 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 [5, 6], Genetic programming (GP) is a relatively recent and fast developing approach to automatic programming [7, 8, 9]. 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 GP techniques to a range of object recognition problems such as shape classification, face identification, and medical diagnosis [10, 11, 12, 13, 14, 15, 16, 17]. While showing promise, current GP techniques are limited and frequently do not give satisfactory results on difficult classification tasks. While there are many problems in the current GP techniques that often lead to unacceptable programs in a reasonable time frame, one main problem is that the crossover operator is not sufficiently powerful to generate good solutions.

The crossover genetic operator has been considered a centre storm of GP [18]. 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. Furthermore, this often destroys the good “building blocks” (sub-programs which are good for that task) in evolved programs.

To improve the standard crossover operator and preserve potential good building blocks, Tackett [19] 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. In this way, better child programs will survive and be put into the next generation while the population size will remain constant. One disadvantage of this method is that the total number of evaluations of genetic programs was increased by $N$ times. To deal with this situation, Tackett used only a small portion of the training fitness cases to replace the whole training set.

### 1.1 Goals

While a brood size of four is commonly used in the method as in [19], a larger size might result in better performance. This size might be task dependent and is also related to the evolutionary process parameters such as the number of generations. To further study the effectiveness of the method, these issues clearly need to be investigated. In addition, the evolutionary training efficiency should also be considered.

The goal of this paper is to further analyse the brood size for the brood recombination crossover method in terms of effect of a fixed brood size and a variable size. To do this, the brood recombination crossover with different brood sizes will be compared with the standard crossover operator in GP on three object classification problems of increasing difficulty to measure the effectiveness and efficiency. Specifically, we are interested in the following issues:

- whether increasing the brood size can result in better performance,
- whether there exists a diversity point for the brood size in the situation that a larger size does not improve the system performance, and
- whether a variable size is better than a fixed size, and
- to find a effective heuristic for setting the brood size.

The remainder of this paper is organised as follows. Section 2 gives a brief overview of the brood recombination crossover method and describes the experiment configurations. Section 3 investigates the use and effect of fixed brood sizes. Section 4 describes the analysis of a variable brood size compared with the fixed size and the basic GP approach. Finally, we draws conclusions in section 5.

## 2 Brood Recombination and Experiment Configuration

### 2.1 Brood Recombination Overview

The main objective of the brood recombination crossover method is to reduce the destructive effect of crossover and preserve good potential building blocks. The rationale behind this method is that many animal species produce far more offspring than are expected to live. Although there are many different mechanisms, the excess offspring die. So should GP crossover. A simple illustration of the method is shown in figure 1.

In this method, a "brood" $N$ was created for each crossover operation. The standard crossover operation was repeated $N$ times on the two same parent programs selected from the population and $2N$ child programs are generated. These child programs are then evaluated and their fitness ranked. The two programs with the best fitness are considered the "real" children of the parents and retained, but other children are discarded.

From the effectiveness point of view, this method improves the standard crossover from two-fold. Firstly, it reduces the effect of disrupting potential building blocks of the standard crossover operator through multiple trials of searching for good crossover points. Secondly, it actually adds a kind of hill-climbing search into the genetic beam search in GP.

Clearly, the brood size is a key parameter in this approach. However, Tackett [19] did not make further investigation and analysis on this parameter, which is the main focus of this paper.

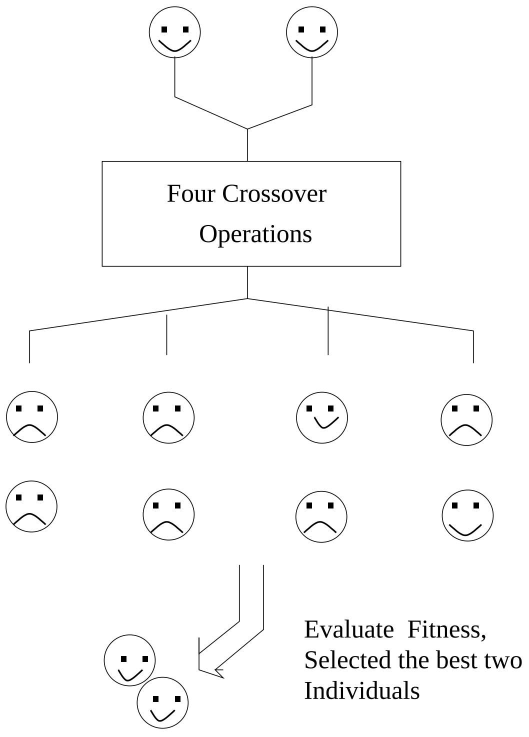
Four Crossover Operations
Evaluate Fitness, Selected the best two Individuals

Figure 1: The brood recombination crossover method.

## 2.2 Experiment Configuration

### 2.2.1 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 shown in figure 2.

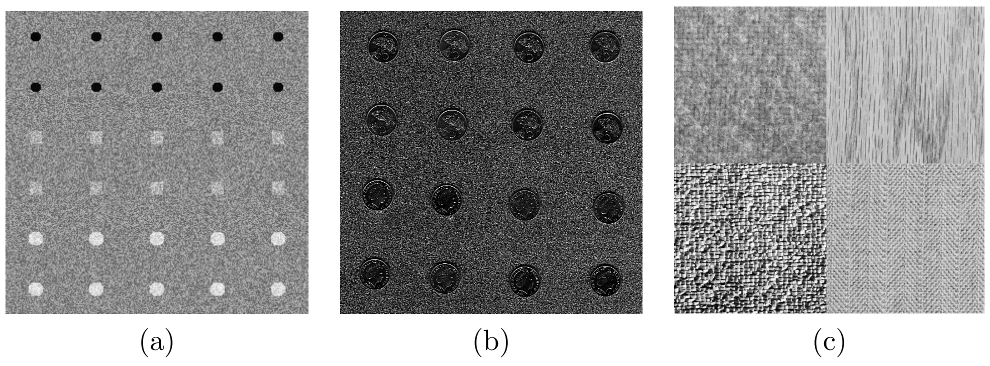

Figure 2: Sample images in datasets: (a) Shape; (b) Coins; (c) Texture.

The first data set (*shape*, figure 2a) 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.*

The second set of images (*coin*, figure 2b) 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 objects are still regular, the problem is quite hard due to the noisy background and the low resolution.

The third set of images (figure 2c) 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 [20]. 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.

### 2.2.2 Terminal Set and Function 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 as features. Since the ranges of these four feature terminal values are quite different, we scale them into the range [-1, 1] based on all object image examples to be classified. While these features are not the best for these particular problems, our goal is to investigate the brood size 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. To be consistent with the feature terminals, we also set the range of these constant terminals to [-1, 1].

The function set consists of the four standard arithmetic and a conditional operations:

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

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 the third argument.

### 2.2.3 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 GP system is a floating point number. In this approach, we used a variant version of the *program classification map* [17] to translate the single output value of a genetic program into a set of class labels.

### 2.2.4 Parameters and Termination Criteria.

The parameter values used in this approach are shown in table 1. These values are determined based on empirical search. The evolutionary process is terminated when the number of generations reaches the pre-defined number, *max_generations*, or when the classification problem has been solved on the training set or the accuracy on the validation set starts falling down, in which case the evolution was terminated earlier.

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

| Parameter | Shape | Coin | Texture | Parameter | Shape | Coin | Texture |
|---|---|---|---|---|---|---|---|
| Pop. Size | 300 | 500 | 500 | Crossover rate: | 50% | 50% | 50% |
| Initial Max. Depth | 5 | 5 | 5 | Mutation rate: | 30% | 30% | 30% |
| Max. Depth | 5 | 6 | 8 | Reproduction rate | 20% | 20% | 20% |
| Max_generations | 50 | 50 | 50 | Brood size | see later | | |

In the approach, we used the tree-structure to represent genetic programs [8]. The ramped half-and-half method was used for generating programs in the initial population and for the mutation operator [7]. The proportional selection mechanism and the reproduction, crossover and mutation operators [9] were used in the learning and evolutionary process.

To compare the results with different brood sizes and the standard crossover operator, 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 following sections.

## 3 Investigation of Fixed Brood Sizes

To investigate the effect of the brood size in the brood recombination crossover method, we did experiments on the three data sets using different fixed brood sizes with 2, 4, 6, 8, and 10 respectively. The average results on the *test set* of the GP system with these brood sizes together with the standard crossover operator (N = 1) are presented in table 2. The first line of the table shows that for the shape data set on the 80 runs, the GP system with the standard crossover operator (N = 1) used 8.59 generations and 0.09 second for the evolutionary process on average and achieved an average classification accuracy of 96.16% on the test set.

As shown in table 2, for all brood sizes investigated here, the brood recombination crossover method achieved better classification accuracy than the standard crossover operator for all the data sets. Although the number of generations used in the evolutionary process for the brood recombination method was smaller than the standard crossover operator, the actual training time was increased. This is mainly because the number of real evaluations in each generation in the brood recombination method was increased. These results further confirmed Tackett's conclusions [19].

The results also show that different brood sizes resulted in different results. For the object classification problems investigated here, it seems that a brood size of 4–8 could be a good starting point.

### 3.0.5 Further Analysis.

Further inspection of the results reveals that as the brood size increases at a certain number (4 or 8 for different data sets), the classification accuracy is increased. When the brood size

Table 2: Results of brood recombination crossover with fixed different brood sizes.

| Dataset | Brood size $N$ | Generation | Time(s) | Accuracy(%) |
|---|---|---|---|---|
| Shape | 1 | 8.59 | 0.09 | 96.16% |
| | 2 | 5.26 | 0.10 | 98.25% |
| | 3 | 3.48 | 0.10 | 98.25% |
| | 6 | 3.01 | 0.11 | 98.12% |
| | 8 | 2.66 | 0.12 | 98.44% |
| | 10 | 2.30 | 0.13 | 98.03% |
| Coin | 1 | 28.64 | 1.78 | 90.37% |
| | 2 | 21.88 | 2.50 | 92.42% |
| | 4 | 19.70 | 3.07 | 93.08% |
| | 6 | 17.59 | 3.53 | 92.82% |
| | 8 | 15.85 | 3.89 | 93.08% |
| | 10 | 15.94 | 4.49 | 92.80% |
| Texture | 1 | 29.99 | 1.83 | 72.45% |
| | 2 | 26.01 | 3.31 | 76.68% |
| | 4 | 21.82 | 4.23 | 76.46% |
| | 6 | 23.69 | 6.09 | 79.82% |
| | 8 | 20.00 | 6.12 | 80.71% |
| | 10 | 17.80 | 6.50 | 78.13% |

exceeds this number, the accuracy achieved starts falling down. These results suggests that there exists such a brood size that could lead to the best performance for a particular task. We refer to this number as *the brood-diversity point*.

In the biological world, the chromosomes for a particular species are usually quite long and the crossover can occur in multiple genes in different positions. Accordingly, a huge number of crossover points can be provided, which allows a large size of brood to produce *distinguished* child chromosomes.

In most GP systems, however, the program size is limited to the parameter, *maximum program size*. In addition, the GP crossover only choose a single point and swap the subtrees in the parent programs. Accordingly, when the brood size increases to a certain number, the probability of the crossover operation on the same two parent programs to produce redundant programs will be extremely high. In other words, when the brood size exceeds the brood-diversity point, the brood recombination crossover operator will not only be unable to produce *distinguished* child programs, but also have to take longer time for more evaluations. This will result in a longer training time with non-improved even slightly worse performance in effectiveness due to possibility of pre-mature convergence.

Clearly, the brood-diversity point is related to the maximum program size. In general, the larger the program size, the bigger the brood size effectively allowed. When setting the brood size, one should consider the *threshold* number of crossover points that the two parent programs with the maximum program size can provide. We expect that this heuristic can help users to choose the brood size when using the brood recombination method in GP.

## 4 Investigation of Variable Brood Sizes

Since the genetic programs can have a different size and the program size varies with the number of generations in the evolutionary process, this section investigate a different approach from the last section — a variable brood size with respect to the number of generations.

According to the GP building block analysis [7], the GP crossover operator will generally preserve small building blocks and construct larger building blocks in the first stage of evolution, but is very likely to tear the larger building blocks apart in the later stage. Accordingly, we can allow the brood size to grow as evolution proceeds in order to protect the larger building blocks.

To meet these requirements, we proposed three ways to dynamically grow the brood size, as shown in equations 2, 3 and 4, respectively, where $N$ is the brood size and $gen$ is the number of generations.

$$N_1(gen) = round(0.14 \times gen) + 1 \tag{2}$$

$$N_2(gen) = round(0.0028 \times gen^2) + 1 \tag{3}$$

$$N_3(gen) = round(0.98 \times \sqrt{gen}) + 1 \tag{4}$$

The main consideration for the three formulas is as follows. In the previous experiment, the results suggest that the maximum brood-diversity point for the three data set is 8. Since the maximum number of generations in this approach is 50, we allow the brood size gradually increases to this number as evolution proceeds closely to generation 50. Notice that the growing speed of the three methods for the brood size is different.

To investigate whether dynamic variable brood size is better than the fixed size and the standard crossover approach, we examined them on the three data sets. The average results are shown in table 3.

Table 3: Results of brood recombination with variable brood sizes and the basic approach. (A1: the basic approach with standard crossover; A2: the linear variable model — equation 2; A3: the squared variable model — equation 3; A4: the squared root model — equation 4)

| Data set | Shape | | | | Coin | | | | Texture | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Approach | A1 | A2 | A3 | A4 | A1 | A2 | A3 | A4 | A1 | A2 | A3 | A4 |
| Generation | 8.59 | 4.40 | 4.54 | 4.99 | 28.64 | 20.59 | 20.59 | 15.85 | 29.99 | 21.93 | 21.31 | 23.24 |
| Time (s) | 0.09 | 0.10 | 0.11 | 0.10 | 0.09 | 2.50 | 3.14 | 3.24 | 1.83 | 3.31 | 4.25 | 4.12 |
| Accuracy(%) | 96.16 | 98.62 | 99.03 | 98.63 | 90.37 | 93.08 | 93.54 | 93.21 | 72.45 | 80.05 | 81.34 | 80.16 |

According to tables 3, the classification accuracies achieved by brood recombination method with the three variable models for the brood size were better than the standard crossover. Inspection of table 2 reveals that these results were also better than that with almost all of the fixed brood sizes. In addition, the evolutionary training times for the variable brood sizes were also shorter than those for the fixed sizes. These results suggest that in the brood recombination method, a variable brood size which is dynamically set with the increase of the number of generations can further improve the system performance.

Inspection of the three variable models for the brood size reveals that the squared variable model (equation 3) achieved the best results for all the three data sets. The other two models achieved similar results but it is not clear which one is better than the other on these data sets. This might be because the squared variable model is closer to the building block disruption trend in evolution, but further investigation is needed in the future.

## 5 Conclusions

The goal of this paper was to investigate the effect of brood size in the brood recombination crossover method in GP for object classification problems. The goal was successfully achieved by testing five different fixed brood sizes and designing three variable models for the brood size. The brood recombination methods with these new developments were examined and compared with the standard crossover operator on three object classification problems of increasing difficulty. The experiment results suggest that the brood recombination method with all these new developments achieved better classification performance than the standard crossover operator while the evolutionary training time was increased.

The results also suggest that different brood sizes usually result in different performances. As the brood size increases by the *brood-diversity point*, the system effective performance can be improved. When the brood size exceeds this point, however, the effective performance will not be improved and the system will become more inefficient. Our research suggests that the brood size is related to the number of generations, the program size and specific tasks.

Investigation of three dynamic models for setting the brood size reveals that a variable brood size which is dynamically set with the number of generations can further improve the system performance over the use of fixed brood sizes. In particular, the squared variable model achieved the best performance for the three data sets examined here.

Although developed for object classification problems, this approach could be expected to be applied to even more general problems.

This work reveals that the brood size is closely related to the maximum program size parameter. We will investigate the relationship between the brood size and the program size together with the number of generations in the future.

## Acknowledgement

We would like to thank Prof Dongqing Qian at Artificial Intelligence Research Centre, Agricultural University of Hebei for his providing work environment and a number of useful discussions.

This work was supported in part by the national Marsden Fund of New Zealand (Grand No. 05-VUW-017) and the University Research Fund at Victoria University of Wellington (Grant No. URF 6/9).

## References

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

[2] Howard, D., Roberts, S.C., Ryan, C.: The boru data crawler for object detection tasks in machine vision. In: Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002. Kinsale, Ireland, Springer (2002) 220–230

[3] Valentin, D., Abdi, H., 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) (1994) 413–429

[4] Poli, R.: Genetic programming for image analysis. In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press (1996) 363–368

[5] Holland, J.H.: 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)

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

[7] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications. Morgan Kaufmann Publishers (1998)

[8] Koza, J.R.: Genetic programming : on the programming of computers by means of natural selection. Cambridge, Mass. : MIT Press, London, England (1992)

[9] Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. Cambridge, Mass. : MIT Press, London, England (1994)

[10] Howard, D., Roberts, S.C., Brankin, R.: Target detection in SAR imagery by genetic programming. Advances in Engineering Software **30** (1999) 303–311

[11] Loveard, T., Ciesielski, V.: Representing classification problems in genetic programming. In: Proceedings of the Congress on Evolutionary Computation. Volume 2., Seoul, Korea, IEEE Press (2001) 1070–1077

[12] Song, A., Ciesielski, V., Williams, H.: Texture classifiers generated by genetic programming. In: Proceedings of the 2002 Congress on Evolutionary Computation CEC2002, IEEE Press (2002) 243–248

[13] Tackett, W.A.: Genetic programming for feature discovery and image discrimination. In: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann (1993) 303–309

[14] Winkeler, J.F., Manjunath, B.S.: Genetic programming for object detection. In: Proceedings of the Second Annual Conference, Morgan Kaufmann (1997) 330–335

[15] Zhang, M., Ciesielski, V.: Genetic programming for multiple class object detection. In: Proceedings of the 12th Australian Joint Conference on Artificial Intelligence, Sydney, Australia, Springer(1999) 180–192

[16] Zhang, M., Andreae, P., Pritchard, M.: Pixel statistics and false alarm area in genetic programming for object detection. In: Applications of Evolutionary Computing, Lecture Notes in Computer Science, Vol. 2611, Springer (2003) 455–466

[17] Zhang, M., Ciesielski, V., Andreae, P.: A domain independent window-approach to multiclass object detection using genetic programming. EURASIP Journal on Signal Processing, **2003**(8) (2003) 841–859

[18] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming - An Introduction. Morgan Kaufmann Publishers (1998)

[19] Tackett, W.A.: Recombination, Selection and the Genetic Construction of Computer Programs. PhD thesis, University of Southern California, Department of Electrical Engineering Systems (1994)

[20] 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*)

[21] Gray, H.F., Maxwell, R.J., Martinez-Perez, I., Arus, C., Cerdan, S.: Genetic programming for classification of brain tumours from nuclear magnetic resonance biopsy spectra. In Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press (1996) 424

[22] Andre, D.: Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In Kinnear, K.E., ed.: Advances in Genetic Programming, MIT Press (1994) 477–494