Applying Online Gradient Descent Search to Genetic Programming for Object Recognition

Will Smart **Mengjie Zhang**

School of Mathematical and Computer Science
Victoria University of Wellington,
P. O. Box 600, Wellington, New Zealand
Email: {smartwill,mengjie}@mcs.vuw.ac.nz

Abstract

This paper describes an approach to the use of gradient descent search in genetic programming (GP) for object classification problems. In this approach, pixel statistics are used to form the feature terminals and a random generator produces numeric terminals. The four arithmetic operators and a conditional operator form the function set and the classification accuracy is used as the fitness function. In particular, gradient descent search is introduced to the GP mechanism and is embedded into the genetic beam search, which allows the evolutionary learning process to globally follow the beam search and locally follow the gradient descent search. This method is compared with the basic GP method on four image data sets with object classification problems of increasing difficulty. The results show that the new method outperformed the basic GP method on all cases in both classification accuracy and training time, suggesting that the GP method with the gradient descent search is more effective and more efficient than without on object classification problems.

Keywords: Genetic Programming, Machine Learning, Data Mining, Object Classification.

1 Introduction

Genetic programming (GP) is a relatively recent and fast developing approach to automatic programming (Banzhaf, Nordin, Keller & Francone 1998, Koza 1992). In genetic programming, solutions to a problem are represented 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, makes GP an exciting new method to solve a great variety of problems.

Since the early 1990s, there has been only a small amount of work on applying genetic programming techniques to object recognition problems, such as (Andre 1994, Howard, Roberts & Brankin 1999, Loveard & Ciesielski 2001, Song, Ciesielski & Williams 2002, Tackett 1993, Winkeler & Manjunath 1997, Zhang & Ciesielski 1999). Typically, these GP systems used either high level or low level image features to form the terminal set, arithmetic and conditional operators to form the function set, and classification accuracy, error rate or similar measures as the fitness function. During the evolutionary process, selection, crossover and mutation operators were applied to the genetic beam search to find good solutions. While most of these GP systems achieved reasonable even good results, they usually spent a long time for training/learning good programs for a particular task. In addition, because of the long training times, the evolutionary process was often stopped when a maximum number of generations was reached, rather than an ideal solution was found.

Gradient descent is a long term established search/learning technique and commonly used to train multilayer feed forward neural networks (Rumelhart, Hinton & Williams 1986). This algorithm can guarantee to find a local minima for a particular task. While the local minima is not the best solution, it often meets the request of that task.

In this paper, we apply the gradient descent search to genetic programming, so that a hybrid beam-gradient descent search scheme can be formed. During the evolutionary process, the beam search is still the basic global search mechanism, but the gradient descent search is locally applied to individual programs in the population inside a particular generation.

The goal of this research is to develop such a search scheme in genetic programming for object classification problems, and to investigate whether this approach can perform better than the basic GP approach in terms of training efficiency and classification performance.

The rest of the paper is organised as follows. Section 2 describes our basic GP approach to object classification, including terminals, functions, fitness function and genetic operations. Section 3 specifies the gradient descent search algorithm in our approach. Section 4 gives the image data sets used in the experiments. Section 5 presents experimental results and section 6 draws the conclusions and gives future work.

2 GP Applied to Object Classification

In this approach, we used the tree-structure to represent genetic programs (Koza 1992). The ramped half-and-half method was used for generating the programs in the initial population and for the mutation operator (Banzhaf et al. 1998). The proportional selection mechanism and the reproduction (Zhang, Ciesielski & Andreae 2003), crossover and mutation operators (Koza 1994) were used in the learning and evolutionary process.

In the remainder of this section, we address the other aspects of our GP learning/evolutionary system: (1) Determination of the terminal set; (2) Determination of the function set; (3) Construction of the fitness measure; and (4) Selection of the input parameters and determination of the termination strategy. The key part, gradient descent search applied to genetic programming, will be descried in section 3.

2.1 Terminals

In this approach, we used two kinds of terminals: *feature terminals* and *numeric terminals*.

2.1.1 Feature Terminals

Feature terminals form the inputs from the environment and usually correspond to image features in object recognition and image analysis. To achieve domain independent object classification, our system used *pixel statistics*, domain independent low level image features, as the feature terminals. While raw image pixels can also achieve this objective, such systems would have to face the problem of a large number of terminals and a long time of evolutionary computation. Since our goal is to introduce the gradient descent search to genetic programming rather than to investigate the effectiveness of raw pixels, we chose a small number of pixel statistics as feature terminals rather than raw pixels.

The pixel statistics considered in this approach are the means and variances of certain regions in the object cutout images. Two such regions are used, the entire object cutout image (A-B-C-D) and the central square region (E-F-G-H), as shown in figure 1. This makes four feature terminals. Since the ranges of these four features are quite different, we linearly scaled these feature values into the range [-1, 1] based on all object image examples to be classified.

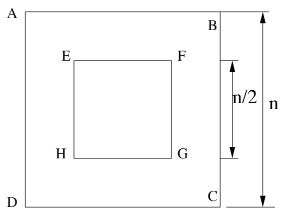

Figure 1: Region features as terminals.

The values of the feature terminals are the values of the pixel statistics for certain object examples. Similarly to neural network inputs which are not changed during network training, these values would remain unchanged in the evolutionary process, although different objects usually have different feature values.

2.1.2 Numeric Terminals

Numeric terminals are floating point numbers randomly generated using a uniform distribution at the beginning of evolution. To be consistent with the feature terminals, we also set the range of these parameters to [-1.0, 1.0].

Unlike the feature terminals, the values of this kind of terminals are the same for all object images. Similarly to feature terminals, they would remain unchange during the evolutionary recombination (crossover and reproduction). However, they will be changed and updated with a continuous parameter entity when the gradient-descent algorithm (see section 3) is applied, which are similar to the weights and biases in neural networks.

Thus, our gradient descent search algorithm will be applied to this type of terminals rather than the feature terminals.

2.2 Functions

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.

All functions can take the result of any other functions or terminals as arguments.

2.3 Classification Strategy

The output of a genetic program in the standard GP system is a floating point number. Generally genetic programs can perform two class object classification tasks quite well where the division between positive and negative numbers of a genetic program output corresponds to the separation of the two object classes. However, for multiple class object classification problems described here, where more than two classes of objects are involved, the standard genetic programming classification strategy mentioned above cannot be applied.

In this approach, we used a different strategy — a variant version of the *program classification map* (Zhang et al. 2003). This variation situates class regions sequentially on the floating point number line. An 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 are allocated with an identical interval of 1.0. For example, a five class problem would have the following classification map:

$$\textbf{class} = \begin{cases} \text{Class 1}, & r < -1.5 \\ \text{Class 2}, & -1.5 \le r < -0.5 \\ \text{Class 3}, & -0.5 \le r < 0.5 \\ \text{Class 4}, & 0.5 \le r < 1.5 \\ \text{Class 5}, & 1.5 \le r \end{cases} \tag{2}$$

where r is the program output.

2.4 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.

2.5 Parameters and Termination Criteria

The parameter values used in this approach are shown in table 1.

Table 1: Parameters used for GP training for the four datasets.

Parameter Kinds	Parameter Names	Shape	coin1	coin2	coin3
Search Parameters	population-size	300	300	500	500
	initial-max-depth	3	3	3	3
	max-depth	5	5	6	6
	max-generations	51	51	51	51
	object-size	16×16	70×70	70×70	70×70
Genetic Parameters	reproduction-rate	20%	20%	20%	20%
	cross-rate	50%	50%	50%	50%
	mutation-rate	30%	30%	30%	30%
	cross-term	15%	15%	15%	15%
	cross-func	85%	85%	85%	85%

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.
- The accuracy on the validation set starts falling down.
- The number of generations reaches the predefined number, *max-generations*.

3 Gradient Descent Applied to Genetic Programming

In this section, we describe how to apply the gradient descent search to genetic programming. In this approach, the online learning (the stochastic procedure) scheme is used, that is, the numeric terminals are updated for each object in the training set.

3.1 Overview of the Algorithm

We assume the successful degree to which the task has been performed can be measured on some real scale and that the performance is scalar and continuous. We also assume that the *cost function* C is used as the measure.

A continuous cost surface can be formed for a given task based on a set of parameters. The lower the point in the cost surface, the better the performance of the system. To improve the system performance, the gradient descent search is applied to take steps "downhill" on C from the current parameter θ. For presentation convenience, the algorithm that applies the gradient descent search to GP is referred to as the *gradient descent algorithm* in this paper.

The gradient of C is found as the vector of partial derivatives with respect to the parameter values. This gradient vector points along the surface, in the direction of maximum-slope at the point used in the derivation. Changing the parameters proportionally to this vector (negatively, as it points "uphill") will move the system down the surface C. The distance moved should therefore be the length of the vector times a factor α.

$$\Delta\theta_i = -\alpha \cdot \frac{\partial C}{\partial \theta_i} \tag{3}$$

where θ_i is the i'th parameter, α is a factor.

It is important to note that the gradient-descent algorithm does not replace *any* of the normal genetic operators that produce populations from generation to generation. Instead, the gradient-descent algorithm augments the existing GP system, by locally applying gradient-descent search to each program in the current population in a particular generation.

The parameters to be changed are the numeric terminals in each program. The feature terminals cannot be changed, as they are set based on actual object examples. The gradient of the cost surface with respect to the values of the numeric terminals can be found through the gradient descent algorithm, which is similar to the back propagation algorithm used in training neural networks.

3.2 The Cost Function

In this approach, we used half of the squared difference between the program actual output and the desired output for a particular object input as the cost function, as shown in equation 4.

$$C_\theta = \frac{(Y - y_\theta)^2}{2} \tag{4}$$

where C_θ is the value of the cost surface at the current parameters θ, y_θ is the actual output of program with the current object as input. Y is the corresponding desired output and is calculated by equation 5.

$$Y = \textbf{class} - \frac{\textbf{numclass} + 1}{2} \tag{5}$$

where **class** is the class label of the object and **numclass** is the total number of classes. For example, for a five class problem as described in equation 2, the desired outputs are $-2, -1, 0, 1$, and 2 for object classes 1, 2, 3, 4, and 5, respectively. In other words, we take the centres of the class regions as desired outputs except the beginning and the end classes.

Based on equations 4 and 5, we will obtain the derivative of the whole genetic program which contains one or more numeric terminals, as shown in equation 6.

$$\frac{\partial C}{\partial y} = \frac{\partial(\frac{(Y-y)^2}{2})}{\partial y} = y - Y \tag{6}$$

The partial derivatives for the numeric terminals can be obtained based on the *chained rule*, which will be described in the next sub section.

3.3 Chained Rules in Genetic Programs

In general, if f, g and h are functions where f depends on g and g depends on h, then the chained rule can be represented as several parts based on some partial derivatives:

$$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial g} \times \frac{\partial g}{\partial h} \tag{7}$$

Now we use the program tree shown in figure 2 as an example to describe the chained rule in our algorithm. Assume O_j is the evaluated result of node j, then $y = O_1$ is the final output of the program.

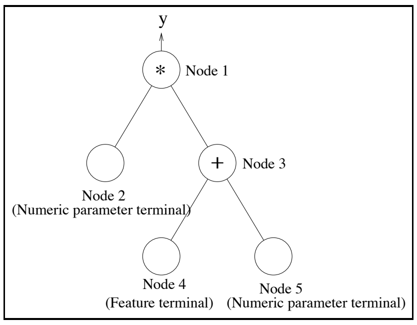

Figure 2: An example program.

Since nodes 2 and 5 are numeric terminals, the gradient vector should contain values from the partial derivatives from nodes 2 and 5. The partial derivatives of the cost function on node 2 and node 5 are:

$$\frac{\partial C}{\partial O_2} = \frac{\partial C}{\partial y} \cdot \frac{\partial y}{\partial O_2} = \frac{\partial C}{\partial y} \cdot \frac{\partial (O_2 O_3)}{\partial O_2} = (y - Y) \cdot O_3$$

$$\begin{aligned} \frac{\partial C}{\partial O_5} &= \frac{\partial C}{\partial y} \cdot \frac{\partial y}{\partial O_5} = \frac{\partial C}{\partial y} \cdot \frac{\partial (O_2 O_3)}{\partial O_5} \\ &= \frac{\partial C}{\partial y} \cdot \frac{\partial (O_2 O_3)}{\partial O_3} \cdot \frac{\partial O_3}{\partial O_5} \\ &= \frac{\partial C}{\partial y} \cdot \frac{\partial (O_2 O_3)}{\partial O_3} \cdot \frac{\partial (O_4 + O_5)}{\partial O_5} \\ &= (y - Y) \cdot O_2 \cdot 1 = (y - Y) \cdot O_2 \end{aligned}$$

Since y, Y, O_2, and O_3 can be obtained from evaluation of the whole program or a part of the program or calculated by equation 5, these gradients can be calculated accordingly. In other words, using the chained rule the gradients can be broken down to evaluated values and derived mathematical operators. The chained rule can be applied many times to accommodate programs of any depth.

The derivatives of the various functions used in this approach are listed in table 2.

Table 2: Function derivatives.

Function f	meanings	$\frac{\partial f}{\partial a_1}$	$\frac{\partial f}{\partial a_2}$	$\frac{\partial f}{\partial a_3}$
(+ a_1 a_2)	$a_1 + a_2$	1	1	n/a
(- a_1 a_2)	$a_1 - a_2$	1	-1	n/a
(* a_1 a_2)	$a_1 \times a_2$	a_2	a_1	n/a
(/ a_1 a_2)	$a_1 \div a_2$	a_2^{-1}	$-a_1 \times a_2^{-2}$	n/a
(if a_1 a_2 a_3)	if $a_1 < 0$ then	0	1 if $a_1 < 0$	0 if $a_1 < 0$
	a_2 else a_3	0	0 if $a_1 \geq 0$	1 if $a_1 \geq 0$

3.4 Calculation of the Factor α

In this approach, the factor α in equation 3 is proportional to the inversed sum of the square gradients on all numeric terminals along the cost surface, as shown in equation 8.

$$\alpha = \eta \cdot \frac{1}{\sum_i^N (\frac{\partial y}{\partial O_i})^2} \tag{8}$$

where N is the number of numeric terminals in the program, i indexes the numeric terminals, y is the output of the program, O_i is the output of the ith numeric terminal, η is a learning rate defined by the user.

Based on equation 3, the change of the value of the ith numeric terminal would be:

$$\Delta O_i = -\eta \cdot \frac{1}{\sum_i^N (\frac{\partial y}{\partial O_i})^2} \cdot \frac{\partial C}{\partial O_i} \tag{9}$$

Accordingly, the new value of the numeric terminal is:

$$(O_i)_{new} = O_i + \Delta O_i = O_i - \eta \cdot \frac{1}{\sum_i^N (\frac{\partial y}{\partial O_i})^2} \cdot \frac{\partial C}{\partial O_i} \tag{10}$$

Based on this definition, if the program only has linear operators performed on subtrees with numeric terminals, the changes to the numeric terminals on the program are independent, and the learning rate η is 1.0, then the program with the changes driven by this gradient descent algorithm would produce ideal result for this object input, that is, the error would be zero (the prove is omitted here due to the page limitation).

3.5 Summary of the Algorithm

FOR each program on each object, do the following:

- S1 Evaluate program, save the outputs of all nodes in the program.
- S2 Calculate the (partial) derivatives of the cost function at numeric terminals using the chained rule and table 2.
- S3 Calculate the change of the numeric terminals based on equation 9.
- S4 Update the numeric terminals according to equation 10.

4 Image Data Sets

We used four image databases in two groups in the experiments. Example images are shown in figure 3.

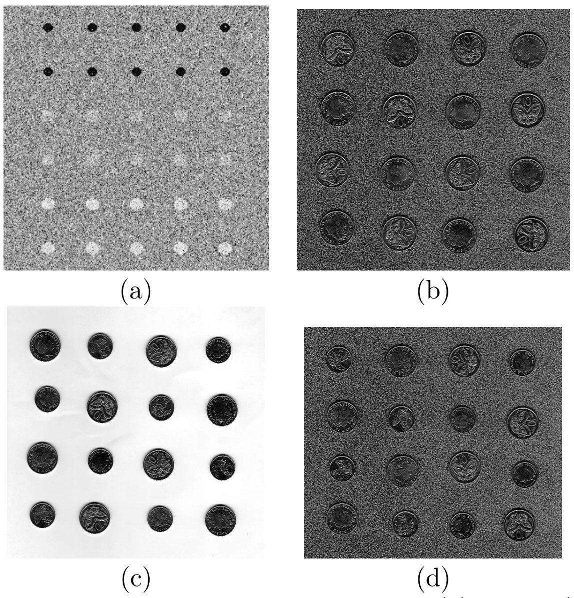

Figure 3: Example images from Shape (a), Coin1 (b), Coin2 (c) and Coin3 (d).

4.1 Computer Generated Shape Dataset

The first group of images (figure 3 (a)) was generated to give well defined objects against a noisy background. The pixels of the objects were produced using a Gaussian generator with different means and variances for each class. Four classes of 713 small objects were cut out from these images to form the classification data set *shape*. The four classes are: black circles, light grey squares, white circles, and grey noisy background. This set was considered to include an easy object classification problem.

4.2 NZ Coin Datasets

The second group of images has three NZ coin datasets. These datasets were intended to be harder than group 1 and consist of scanned 5 cent and/or 10 cent New Zealand coins. In this group, three data sets, *coin1*, *coin2* and *coin3*, were constructed to provide object classification problems of increasing difficulty. Example images for each of the three datasets are shown in figure 3 (b), (c), and (d), respectively. The first coin data set has 576 object cutout images of three classes: 10 cent heads, 10c tails, and a noisy background. The second coin data set consists of five classes of object cutouts: 5 cent heads, 5 cent tails, 10 cent heads and 10 cent tails, and a relatively uniform background. The third coin data set also consists of five classes of object cutouts, but the background is highly clustered, which makes the classification problems much harder.

The object cutout images in each of the these datasets 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 over-fitting, and one third for the test set for measuring the performance of the learned program classifiers.

5 Results and Discussion

This section presents the results of our GP approach with the online gradient descent algorithm on the four datasets and compare them with the results of the basic GP approach without the gradient descent search. For all cases, 10 runs were carried out and the average results on the test set are presented.

5.1 Shape Dataset

Table 3 shows the results on the shape data set using different learning rates. The first line shows that for the Shape data set, the basic GP approach without using the gradient descent algorithm (η is 0.0) achieved an average accuracy of 99.48% over 10 runs on the test set and the average number of generations of the 10 runs spent on the training process was 9.56.

Table 3: Results of the shape Dataset.

Dataset	η	**Generations**	**Accuracy (%)**
Shape	0.0	9.56	99.48
	0.2	1.00	100.00
	0.4	1.00	100.00
	0.7	1.00	100.00
	1.0	1.00	100.00
	1.4	1.00	99.95

For dataset *shape*, the GP approach with the gradient descent algorithm always outperformed the basic GP approach. In particular, using the gradient descent algorithm at learning rates of 0.2, 0.4, 0.7 and 1.0, ideal performances were achieved and only one generation[1] was required for the evolutionary learning process. This is considerably faster than the basic GP approach.

5.2 Coin Datasets

The results for the three coin datasets are shown in table 4. These results show a similar pattern to the *shape* dataset. For all the three datasets, the accuracy performances of the GP approach with the gradient descent algorithm were always superior to those without using the algorithm, and this was particularly true for difficult problems such as datasets *coin2* and *coin3*. These results suggests that a combination of the basic genetic beam search with the gradient descent search could always find better genetic program classifiers than the genetic beam search only for these object classification tasks.

Table 4: Results of the three *coin* Datasets.

Datasets	η	**Generations**	**Accuracy (%)**
Coin1	0.0	8.00	99.53
	0.2	1.00	99.95
	0.4	1.00	99.84
	0.7	1.00	99.79
	1.0	1.00	99.95
	1.4	1.00	99.95
Coin2	0.0	51.00	82.12
	0.2	20.40	98.94
	0.4	25.40	98.94
	0.7	23.30	98.81
	1.0	36.70	97.69
	1.4	34.00	97.94
Coin3	0.0	51.00	73.83
	0.2	51.00	83.50
	0.4	51.00	82.83
	0.7	50.20	85.17
	1.0	51.00	86.50
	1.4	51.00	80.83

In terms of the number of generations used in the training process, the GP approach with the gradient descent algorithm required much fewer generations to achieve good results for datasets *coin1* and *coin2*. For the *coin2* dataset, for example, the approach with the gradient descent algorithm at a learning rate of 0.2 only used 20.4 generations and achieved almost ideal performance (98.94%), while the basic GP approach used 51 generations but only obtained 82.12% accuracy. For dataset *coin3*, both the GP approaches used almost all the 51 generations (one of the stopping criteria). However, after examining the internal behaviour of the evolutionary learning process, we found that the method with the gradient descent algorithm converged at generations 10-20 in all the cases while the basic method without this algorithm got converged at generations 45-51 in almost all the cases. These results suggest that the GP approach with the gradient descent algorithm converged faster than without.

For dataset *coin3*, the accuracy performance could not be improved with even more generations. This is mainly because only four pixel statistics were used as features terminals, which were not sufficient for this difficult object classification task.

The results also show that different performances would be obtained if different learning rates were used. Similarly to the neural network method, this

[1]One generation in GP with the gradient descent algorithm usually requires a slightly longer time than that in the basic GP approach. However, this was considerably improved by only applying the gradient descent algorithm to the top 5% of the programs in the population. Thus, the actual times for a single generation in the two methods are still very similar.

learning parameter needs to be empirically searched through tuning certain experiments to obtain good results. However, if this can lead to considerably better results, this price is worth to pay. From our experiments, a learning rate between 0.2 to 1.0 is a good starting point for object classification problems.

As expected, the performances on all the four image datasets deteriorated as the degree of difficulty of the object classification problem was increased.

6 Conclusions

The goal of this paper is to develop an approach to integrating gradient descent search to genetic programming (GP) and to investigate whether this new approach is better than the basic GP approach for object classification problems. This goal was achieved by constructing a terminal set, a function set, a classification rule and a fitness function, developing a gradient descent algorithm, and applying both the new approach and the basic GP approach to four object classification problems of increasing difficulty.

The gradient descent search was introduced to the GP mechanism and was embedded into the genetic beam search, allowing the evolutionary learning process to globally follow the beam search and locally follow the gradient descent search to find good solutions.

On the relatively easy object classification problems such as *shape, coin1* and *coin2*, the new approach achieved almost ideal performance (100% accuracy). On the difficult dataset (*coin3*), the best result of 86.5% accuracy was achieved. On all the datasets investigated here, the method with the gradient descent algorithm always achieved better results than the method without in both classification accuracy and training generations.

These results show that the new method outperformed the basic GP method on all cases in both classification accuracy and training generations, suggesting that the GP method with the gradient descent search is more effective and more efficient than without for object classification problems.

Although developed for object classification problems, this new method is expected to be able to be applied to general classification and prediction tasks in data mining applications.

For future work, we will investigate whether the performance on the difficult coin data sets can be improved if more features are added to the feature terminal set. We will also investigate the power and reliability of the new method on even more difficult image classification problems such as face recognition problems and satellite image detection problems to find to what situations this method is best suited, and compare the performance with other long-term established methods such as decision trees, neural networks, and support vector machines.

References

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

Banzhaf, W., Nordin, P., Keller, R. E. & Francone, F. D. (1998), *Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications*, San Francisco, Calif. : Morgan Kaufmann Publishers; Heidelburg : Dpunkt-verlag. Subject: Genetic programming (Computer science); ISBN: 1-55860-510-X.

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

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

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

Loveard, T. & Ciesielski, V. (2001), Representing classification problems in genetic programming, *in* 'Proceedings of the Congress on Evolutionary Computation', Vol. 2, IEEE Press, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, pp. 1070–1077.
*http://goanna.cs.rmit.edu.au/ toml/cec2001.ps

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), Learning internal representations by error propagation, *in* D. E. Rumelhart, J. L. McClelland & the PDP research group, eds, 'Parallel distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations', The MIT Press, Cambridge, Massachusetts, London, England, chapter 8.

Song, A., Ciesielski, V. & Williams, H. (2002), Texture classifiers generated by genetic programming, *in* D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow & M. Shackleton, eds, 'Proceedings of the 2002 Congress on Evolutionary Computation CEC2002', IEEE Press, pp. 243–248.

Tackett, W. A. (1993), Genetic programming for feature discovery and image discrimination, *in* S. Forrest, ed., 'Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93', Morgan Kaufmann, University of Illinois at Urbana-Champaign, pp. 303–309.

Winkeler, J. F. & Manjunath, B. S. (1997), Genetic programming for object detection, *in* J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba & R. L. Riolo, eds, 'Genetic Programming 1997: Proceedings of the Second Annual Conference', Morgan Kaufmann, Stanford University, CA, USA, pp. 330–335.

Zhang, M. & Ciesielski, V. (1999), Genetic programming for multiple class object detection, *in* N. Foo, ed., 'Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99)', Springer-Verlag Berlin Heidelberg, Sydney, Australia, pp. 180–192. Lecture Notes in Artificial Intelligence (LNAI Volume 1747).

Zhang, M., Ciesielski, V. & Andreae, P. (2003), 'A domain independent window-approach to multiclass object detection using genetic programming', *EURIASP Journal on Signal Processing, Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis* **2003**(8), 841–859.