Genetic Programming for Object Detection: Improving Fitness Functions and Optimising Training Data

Mengjie Zhang, *Member, IEEE,* Malcolm Lett

***Abstract*— This paper describes an approach to the improvement of a fitness function and the optimisation of training data in genetic programming (GP) for object detection particularly object localisation problems. The fitness function uses the weighted F-measure of a genetic program and considers the localisation fitness values of the detected object locations. To investigate the training data with this fitness function, we categorise the training data into four types: *exact centre, close to centre, include centre,* and *background.* The approach is examined and compared with an existing fitness function on three object detection problems of increasing difficulty. The results suggest that the new fitness function outperforms the old one by producing far fewer false alarms and spending much less training time and that the first two types of the training examples contain most of the useful information for object detection. The results also suggest that the complete background type of data can be removed from the training set.**

***Index Terms*— Genetic programming, object detection, object localisation, object recognition, object classification, evolutionary computing, fitness function, training data.**

I. Introduction

OBJECT detection tasks arise in a very wide range of applications, such as detecting faces from video images, finding tumours in a database of x-ray images, and detecting cyclones in a database of satellite images [1], [2], [3], [4]. In many cases, people (possibly highly trained experts) are able to perform the detection task well, but there is either a shortage of such experts, or the cost of people is too high. Given the amount of image data containing objects of interest that need to be detected, computer based object detection systems are of immense social and economic value.

An object detection program must automatically and correctly determine whether an input vector describing a portion of a large image at a particular location in the large image contains an object of interest or not and what class the suspected object belongs to. Writing such programs is usually difficult and often infeasible: human programmers often cannot identify all the subtle conditions needed to distinguish between all objects and background instances of different classes.

Genetic programming (GP) is a relatively recent and fast developing approach to automatic programming [5], [6], [7]. 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. GP has been applied to a range of object detection and recognition tasks with some success [5], [8], [9], [10], [11], [12], [13].

Finding a good fitness function for a particular object detection problem is an important but difficult task in developing a GP system. Various fitness functions have been devised for object detection, with varying success [5], [9], [11], [14], [15]. These tend to combine many parameters using scaling factors which specify the relative importance of each parameter, with no obvious indication of what scaling factors are good for a given problem. Many of these fitness functions require clustering to be performed to group multiple localisations of single objects into a single point before the fitness is determined [16], [15], [14]. Other measures are then incorporated in order to include information about the pre-clustered results (such as how many points have been found for each object). While some of these systems achieved good detection rates, many of them resulted in a large number of false alarms. In addition, the clustering process during the evolutionary process made the training time very long.

Organising training data is critical to any learning approaches. The previous approaches in object detection tend to use all possible positions of the large image in training an object detector. However, this usually requires a very long training time due to the use of a large number of positions on the background.

This paper aims to investigate a new fitness function and a new way to optimise the training data in GP for object detection, in particular object localisation, with the goal of improving the detection performance and refining training examples. The approach will be examined and compared with an existing GP approach on a sequence of object detection problems of increasing difficulty.

The remainder of this paper is organised as follows. Section II gives some essential background on GP and object detection/recognition. Section III describes the GP approach to object detection, including the major components of the approach. Section IV focuses on the new fitness function and

Mengjie Zhang is with the School of Mathematics, Statistics and Computer Science, Victoria University of Wellington, P. O. Box 600, Wellington, New Zealand (phone: +64 4 463 5654; fax: +64 4 463 5045; email: mengjie@mcs.vuw.ac.nz).

Malcolm Lett is also with the School of Mathematics, Statistics and Computer Science, Victoria University of Wellington, P. O. Box 600, Wellington, New Zealand.

compares it with an existing clustering based fitness function. Section V investigates the training data. Finally, we draw conclusions in section VI. Some GP basics are given in the appendix.

II. Background

A. Genetic Programming and Main Characteristics

GP is an approach to automatic programming, in which a computer can construct and refine its own programs to solve specific tasks. First introduced by Koza [6] in the early 1990s, GP has become another main genetic paradigm in evolutionary computation (EC) in addition to the well known *genetic algorithms* (GAs).

Compared with GAs, GP has a number of characteristics. While the standard GAs use bit strings to represent solutions, the forms evolved by GP are generally trees or tree-like structures. The standard GA bit strings use a fixed length representation while the GP trees can vary in length. While the GAs use a binary alphabet to form the bit strings, the GP uses alphabets of various sizes and content depending on the problem domain. These trees are made up of internal nodes and leaf nodes, which have been drawn from a set of primitive elements that are relevant to the problem domain. Compared with a bit string to represent a given problem, the trees can be much more flexible.

The basic concepts, genetic operators, and the GP algorithm are described in the appendix.

B. Object Detection

The term *object detection* here refers to the detection of small objects in large images. This includes both *object classification* and *object localisation*. *Object classification* refers to the task of discriminating between images of different kinds of objects, where each image contains only one of the objects of interest. *Object localisation* refers to the task of identifying the positions of all objects of interest in a large image. The object detection problem is similar to the commonly used terms *automatic target recognition* and *automatic object recognition*.

Object detection performance is usually measured by *detection rate* and *false alarm rate*. The detection rate (DR) refers to the number of small objects correctly reported by a detection system as a percentage of the total number of actual objects in the image(s). The false alarm rate (FAR), also called false alarms per object [17], refers to the number of non-objects incorrectly reported as objects by a detection system as a percentage of the total number of actual objects in the image(s). Note that the detection rate is between 0 and 100%, while the false alarm rate may be greater than 100% for difficult object detection problems.

C. GP Related Work for Object Detection and Recognition

Since the early 1990s, there has been only a small amount of work on applying GP techniques to object classification, object detection and other image recognition problems. This in part reflects the fact that GP is a relatively young discipline compared with, say, neural networks and genetic algorithms.

In terms of the number of classes in object detection, there are two categories. The first is *one-class object detection problem*, where there are multiple objects in each image, however they belong to or are considered the same (single) class of interest. In nature, these problems contain a two-class (binary) classification problem: *object* versus *non-object*, also called *object* versus *background*. Examples are detecting small targets in thermal infrared images [17] and detecting a particular face in photograph images [18]. The problem is actually the same as *object localisation*, where the main goal is to find where the objects of interest are in the large images. The second is *multi-class object detection problem*, where there are multiple object classes of interest each of which has multiple objects in each image. Detection of handwritten digits in postal code images [19] is an example of this kind. While GP has been widely applied to the one-class object detection and binary classification problems [15], [8], [9], [20], it has also been applied to multi-class object detection and classification problems [21], [22], [23], [10], [24], [11].

In terms of the representation of genetic programs, different forms of genetic programs have been developed in GP systems for object classification and image recognition. The main program representation forms include tree or tree-like or numeric expression programs [5], [7], [21], [11], graph based programs [5], linear GP [25], linear-graph GP [26], and grammar based GP [27].

The use of GP in object detection and image recognition has also been investigated in a variety of application domains. These domains include military applications [9], [20], English letter recognition [28], face/eye detection and recognition [29], [22], [30], vehicle detection [15], [31] and other vision and image processing problems [32], [33], [6], [34], [35], [36].

III. The GP Approach to Object Detection

The process for object detection is shown in Figure 1. A raw image is taken and a trained localiser applied to it, producing a set of points found to be the positions of these objects. Single objects could have multiple positions ("localisations"), however ideally there would be exactly one localisation per object. Regions of the image are then "cut out" at each of the positions specified. Each of these cutouts are then classified using the trained classifier.

This method treats all objects of multiple classes as a single "object of interest" class for the purpose of localisation, and the classification stage handles attaching correct class labels. Compared with the single-stage approach [10], [11], this approach has the advantage that the training is easier for both stages as a specific goal is focused on the training of each of the two stages. The first is tailored to achieving results as close to the object centres as possible (to achieve high "positional accuracy"), while the second is tailored to making all classifications correct (high "classification accuracy").

The object localisation stage is performed by means of a window which sweeps over the whole image, and for each position extracts the features and passes them to the trained localiser. The localiser then determines whether each position is an object or not (i.e. background).

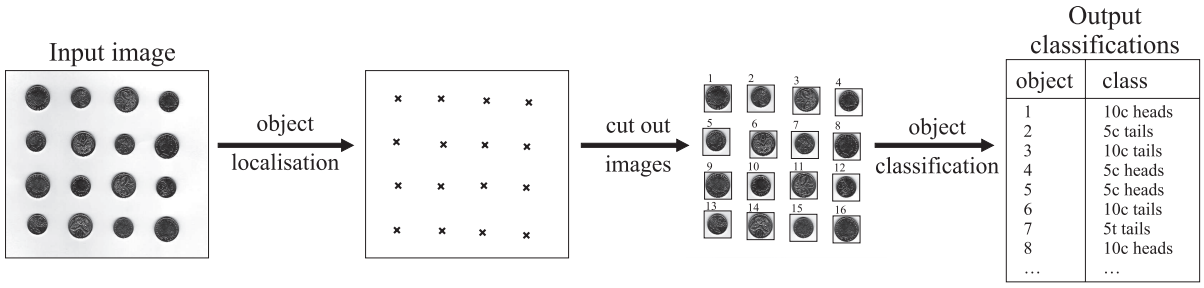


Fig. 1. An overview of the object detection process.

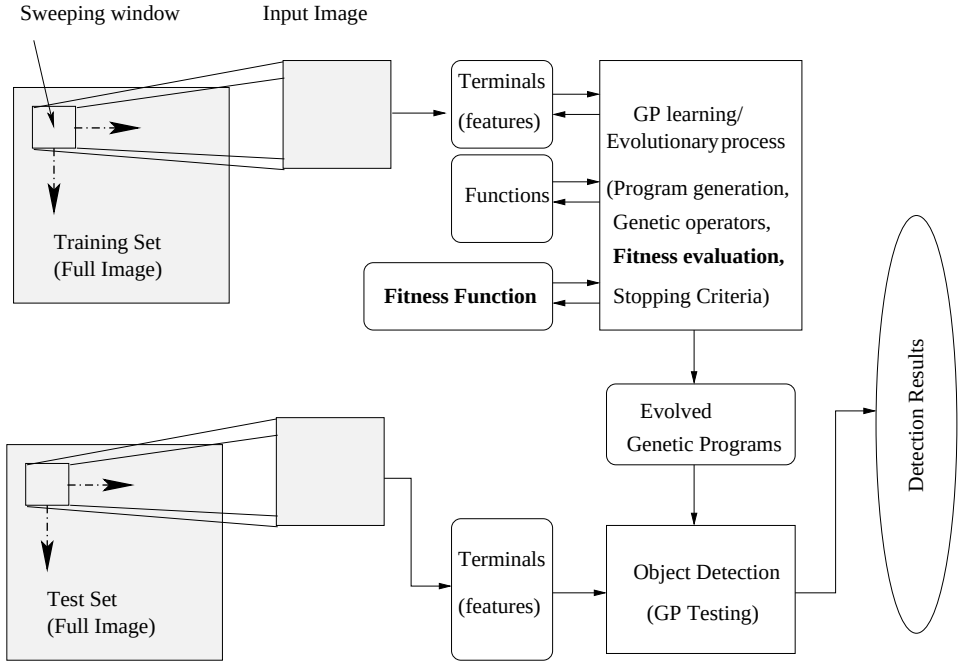


Fig. 2. GP approach to object detection.

Our work will focus on object localisation using genetic programming. Figure 2 shows an overview of this approach, which has a learning process and a testing procedure. In the learning/evolutionary process, the evolved genetic programs use a square input field which is large enough to contain each of the objects of interest. The programs are applied at many sampled positions within the images in the *training set* to detect the objects of interest. If the program localiser returns a value greater than or equal to zero, then this position is considered the centre of an object of interest; otherwise it is considered background. In the test procedure, the best evolved genetic program obtained in the learning process is then applied, in a moving window fashion, to the whole images in the *test set* to measure object detection performance.

This approach has five major components: (1) Determination of a terminal set; (2) Determination of a function set; (3) Construction of a new fitness function; (4) Determination of the major parameter values and the termination criteria; and (5) investigation of the training data. In addition, to examine the performance of this approach, we also need to choose the object detection example tasks.

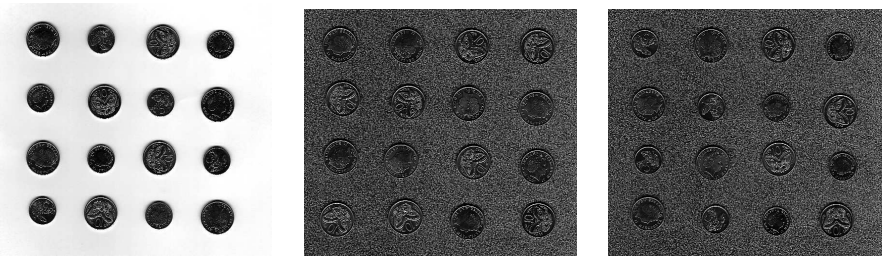
Construction of a new fitness function and investigation of the training data are the main focuses of this paper, which will be described in the next sections. In the rest of this section, we will describes all of the other components.

A. Terminal Set

For object detection problems, terminals generally correspond to image features. In this approach, the features are extracted by calculating the mean and standard deviation of pixel values within several circular regions. This set of features has the advantages of being rotationally invariance. In addition, we also used a constant terminal. Note that finding a good set of features is beyond the goal of this paper, and we will use this set of features to check the performance of both the existing and the new approaches for comparison purpose only.

B. Function Set

The function set contains the four standard arithmetic and a conditional operation: $FuncSet = \{+, -, *, /, if\}$. The $+$, $-$, and $*$ operators are usual addition, subtraction and multiplication, while / represents "protected" division. The *if* function returns its second argument if the first argument is positive or returns its third argument otherwise.



(a) (b) (c)

Fig. 3. Sample images in the three data sets. (a) Easy; (b) Medium difficulty; (c) Hard.

C. GP Structure, Parameters and Termination Criteria

In this system, we used tree structures to represent genetic programs [6]. The ramped half-and-half method [5] was used for generating programs in the initial population and for the mutation operator. The proportional selection mechanism and the reproduction, crossover and mutation operators were used in evolution.

We used a population of 500 genetic programs for evolution in each experiment run. The reproduction rate, crossover rate and mutation rate were 5%, 70% and 25%, respectively. The program size was initialised to 4 and it could increase to 8 during evolution.

The system run 50 generations unless it successfully found an ideal solution or the performance on the validation set fell down, in which cases the evolution was terminated early.

D. Data Sets

To investigate the performance of this approach, we chose three image data sets of New Zealand 5 and 10 cent coins in the experiments. Examples are shown in Figure 3. The data sets are intended to provide object localisation/detection problems of increasing difficulty. The first data set (*easy*) contains images of tails and heads of 5 and 10 cent coins against an almost uniform background. The second (*medium difficulty*) is of 10 cent coins against a noisy background, making the task harder. The third data set (*hard*) contains tails and heads of both 5 and 10 cent coins against a noisy background.

We used 24 images for each data set in our experiments and equally split them into three sets: a training set for learning good genetic programs, a validation set for monitoring the training process to avoid overfitting, and a test set to measure object detection performance.

In our experiments, a total number of 100 runs were performed on each data set and the average results are presented in the next two sections.

IV. Fitness Function

A. Design Considerations

During the evolutionary process for object detection, we expect that the evolved genetic programs only detect the objects when the sweeping window is centred over these objects. However, in the usual case, these evolved genetic programs will also detect some "objects" not only when the sweeping window is within a few pixels of the centre of the target objects, but also when the sweeping window is centred over a number of cluttered pieces of background. Clearly, these "objects" are not those we expected but false alarms.

Different evolved genetic programs typically result in different numbers of false alarms and such differences should be reflected when these programs are evaluated by the fitness function.

When designing a fitness function for object detection problems, a number of considerations need to be taken into account. At least the following requirements should be considered.

- R1. The fitness function should encourage a greater number of objects to be detected. In the ideal case, all the objects of interest in large images can be detected.
- R2. The fitness function should prefer a fewer number of false alarms on the background.
- R3. The fitness function should encourage genetic programs to produce detected object positions closer to the centres of the target objects.
- R4. For a single object to be detected, the fitness function should encourage programs to produce fewer detected "objects" (positions) within a few pixels from the target centre.
- R5. For two programs which produce the same number of detected "objects" for a single target object but the "objects" detected by the first program are closer to the target object centre than those detected by the second program, the fitness function should rank the first program better than the second.

Some typical examples of these requirements are shown in figure 4. In this figure, the circles are target objects and squares are large images or regions. A cross (x) represents a detected object. In each of the five cases, the program associated with the left figure should be considered better than that with the right.

B. An Existing Fitness Function

As the goal is to detect the target objects with no or a small number of false alarms, many GP systems uses a combination of detection rate and false alarm rate or recall and precision as the fitness function. For example, a previous GP system uses the following fitness function [10]:

$$fitness_{CBF} = A \cdot (1 - DR) + B \cdot FAR + C \cdot FAA \quad (1)$$

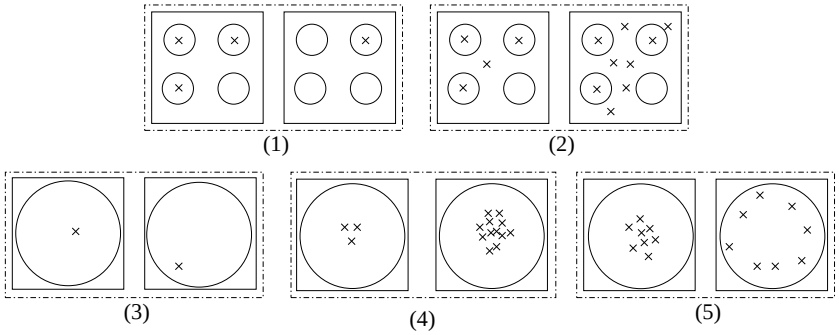


Fig. 4. Examples of the design considerations of the fitness function.

where DR, FAR, and FAA are detection rate (the number of small objects correctly reported by a detection system as a percentage of the total number of actual objects in the images), false alarm rate (also called *false alarms per object*, the number of non-objects incorrectly reported as objects by a detection system as a percentage of the total number of actual objects in the images), and false alarm area (the number of false alarm pixels which are not object centres but are incorrectly reported as object centres before clustering), respectively, and A, B, C are constant weights which reflect the relative importance of detection rate versus false alarm rate versus false alarm area.

Basically, this fitness function has considered requirement 1, and partially considered requirements 2 and 4, but does not take into accounts of requirements 3 and 5. Although this fitness function performed reasonably well on some problems, it still produced many false alarms and the evolutionary training time was still very long [10]. Since this method used clustering before calculating the fitness, we refer to it as *clustering based fitness*, or CBF for short.

C. A New Fitness Function — RLWF

To avoid a very large false alarm rate (greater than 100% for difficult problems) in the training process, we use precision and recall, both of which have the range between [0, 1], to construct the new fitness functions. *Precision* refers to the number of objects correctly localised/detected by a GP system as a percentage of the total number of object localised/detected by the system. *Recall* refers to the number of objects correctly localised by a system as a percentage of total number of target objects in a data set. Note that precision/recall and detection rate/false alarm rate have internal relationship, where the value of one pair for a problem can be calculated using the other for the same problem.

During the object localisation process, a genetic program might consider many pixel positions in an image as object centres and we call each object centre localised in an image by a genetic program a *localisation*.

Unlike the previous fitness function CBF, the new fitness function is based on a "Relative Localisation Weighted F-measure" (RLWF), which attempts to acknowledge the worth/goodness of individual localisations made by the genetic program. Instead of using either correct or incorrect to represent a localisation, each localisation is allocated a weight (referred to as the *localisation fitness*, *LF*) which represents its individual worth and counts towards the overall fitness.

Each weight is calculated based on its relative location, or the distance of the localisation from the centre of the closest object, as shown in Equation 2.

$$\mathrm{LF}(x,y) = \begin{cases} 1 - \frac{\sqrt{x^2+y^2}}{r} & , \text{ if } \sqrt{x^2+y^2} \le r \\ 0 & , \text{ otherwise} \end{cases} \quad (2)$$

where $\sqrt{x^2+y^2}$ is the distance of the localisation position (x, y) from target object centre, and *r* is called the "localisation fitness radius", defined by the user. In this system, r is set to a half of the square size of the input window, which is also the radius of the largest object.

In order to deal with all the situations in the five design requirements, we used the localisation fitness to construct our new fitness function, as shown in Equations 3 to 5. The precision and recall are calculated by taking the localisation fitness for all the localisations of each object and dividing this by the total number of localisations or total number of target objects respectively.

$$\mathrm{WP} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{L_i}\mathrm{LF}(x_{ij},y_{ij})}{\sum_{i=1}^{N} L_i} \quad (3)$$

$$\mathrm{WR} = \frac{\sum_{i=1}^{N}\frac{\sum_{j=1}^{L_i}\mathrm{LF}(x_{ij},y_{ij})}{L_i}}{N} \quad (4)$$

$$\mathrm{fitness}_{RLWF} = \frac{2 \times \mathrm{WP} \times \mathrm{WR}}{\mathrm{WP} + \mathrm{WR}} \quad (5)$$

where N is the total number of target objects, (x_{ij}, y_{ij}) is the position of the j-th localisation of object i, L_i is number of localisations made to object i, WP and WR are the weighted precision and recall, and *fitness*$_{RLWF}$ is the localisation fitness weighted F-measure, which is used as the new fitness function.

The new fitness function has a number of properties. Firstly, the main parameter in this fitness function is the *localisation fitness*, which can by easily determined in the way presented here. This has an advantage over the existing methods which have many parameters whose values usually need to be manually determined. Secondly, in the previous approaches, the multiple localisations of each object must be clustered into

TABLE I
RESULTS OF THE GP SYSTEMS WITH THE TWO FITNESS FUNCTIONS.

Dataset	Fitness function	Test Accuracy			Training Efficiency	
		LR (%)	LP (%)	ExtraLocs	Generations	time(sec)
Easy	CBF	99.99	98.26	324.09	13.69	178.99
	RLWF	99.99	99.36	98.35	36.44	111.33
Medium	CBF	99.60	83.19	804.88	36.90	431.94
	RLWF	99.90	94.42	95.69	34.35	105.56
Hard	CBF	98.22	75.54	1484.51	31.02	493.65
	RLWF	99.53	87.65	114.86	33.27	107.18

one group and its centre found. While this is not a too difficult task, it is very time consuming to do during training. This new fitness function does not require clustering before the fitness is calculated. We expect that the new fitness function can do a better job in terms of reducing false alarms and evolutionary training time.

D. Results

To give a fair comparison for the two fitness functions, the "localisation recall (LR) and precision (LP)" were used to measure the final object detection accuracy on the test set. LR is the number of objects with one or more correct localisations within the localisation fitness radius at the target object centres as a percentage of the total number of target objects, and LP is the number of correct localisations which fall within the localisation radius at the target object centres as a percentage of the total number of localisations made. In addition, we also check the "Extra Localisations" (ExtraLocs) for each system to measure how many extra localisations were made for each object. The training efficiency of the systems is measured with the number of training generations and the CPU (user) time in second.

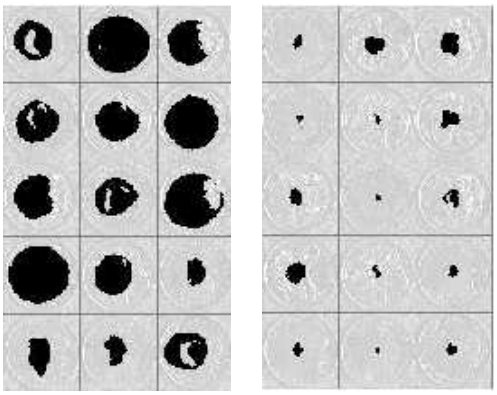
Table I shows the results of the GP systems with the two fitness functions. The results on the easy data set show that both the fitness functions achieved almost perfect test accuracy. Almost all the objects of interest in this data set were successfully localised with very few false alarms (both LR and LP are very close to 100%), reflecting the fact that the detection task in this data set is relatively easy. However, the extra locations and the training time resulted from the two approaches are quite different. The new fitness function (RLWF) produced a far fewer number of extra localisations per object than clustering based fitness function (CBF) and the gap between them is significant. Although the CBF approach used only 13.69 generations on average, which are considerably fewer than that of the new RLWF, it actually spent about 50% longer training time. This confirms our early hypothesis that the clustering process in the CBF approach is time consuming and the approach with the new fitness function is more efficient than that with CBF.

The results on the other two data sets show a similar pattern in terms of the number of extra localisations and training time. The systems with RLWF always produced a significantly fewer number of extra localisations and a much short training time than CBF. In addition, although almost all the objects of interest in the large images were successfully detected (LRs are almost 100%), the localisation precisions achieved by RLWF were significantly better than CBF, suggesting that the new fitness function outperforms the existing one in terms of reducing false alarms.

As expected, performance on the three data sets deteriorated as the degree of difficulty of the object detection problem was increased.

E. Detection Map Analysis

To give an intuitive view of detection performance of the two fitness functions, we checked the "detection maps" of some objects in the test set. Figures 5 (a) and (b) show the detection maps for the same 15 objects in the medium difficulty data set produced by the two approaches. The black pixels in these maps indicate the localisations of the 15 objects produced using the two fitness functions. The "background" means that no objects were found in those positions.



(a) (b)

Fig. 5. Sample object detection maps. (a) CBF; (b) RLWF.

As shown in the figure, the clustering based fitness function CBF resulted in a huge number of extra localisations for all the 15 objects detected. The new fitness function, however, only resulted in a small number of extra localisations. These maps confirm that the new fitness function was more effective than the clustering based fitness function on these problems.

V. OPTIMISING TRAINING DATA

We could train the detection system with a full set of cutouts taken from a window at all possible positions over the training images. However, for a set of large images, this can create a huge number of training examples making the training time unsuitably long. While we can reduce the total number of training examples using a combination of hand-chosen and

randomly chosen examples [16], in this approach, we focus on investigating whether some examples are better than others and how we pick up better examples.

A. Four Training Data Types

The traditional approaches usually use *positive* and *negative* examples. The former refers to the exact object examples and the latter refers to those for the background [9], [10], [15]. However, this did not consider those with a portion of objects and a portion of background. In this approach, we identified four basic types of training examples, as shown in figure 6. The *exact centre* type (figure 6a) refers to the positive object examples which sit exactly the centre of the sweeping window. This type of examples has only a very small number. For example, in each of our training images, we have only 16 such examples out of approximately half a million pixel positions. The *background* type (figure 6d) refers to the positions (x) which do not contain any piece of objects. This type typically has a huge number of examples. The *close to centre* type refers to the examples that have the centre of the sweeping window falling down within the bounds of an object (figure 6b). The *include objects* type refers to the examples that contain some pixels of an object but are not considered as the *close to centre* type.

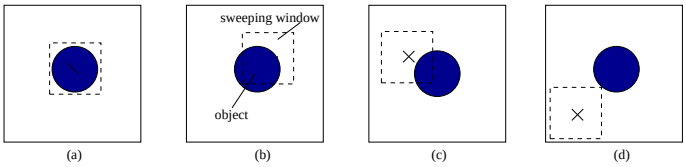


Fig. 6. Examples of training data types caused by different input window positions.

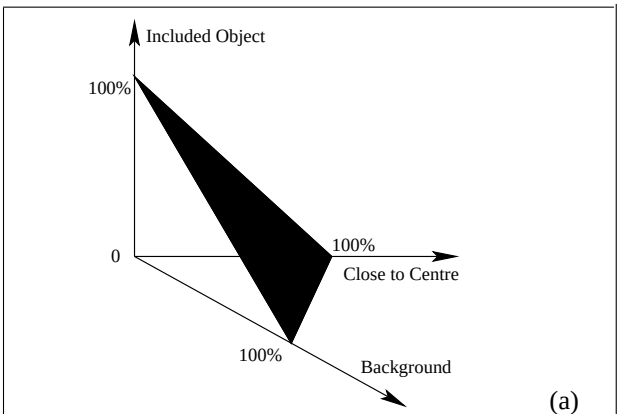
B. Optimisation of Training Data

For a problem domain, we assume that there is some proportion of these four types which is optimal (or close to optimal) for object detection. From previous research, we found that the exact centre type is always important for object detection. As the number of examples of this type is very small, we will always use this type of examples in the experiments and assume that the best results can only be achieved by including them. In the remainder of investigation, we will vary the proportions among the rest three types to find the optimal combinations.

Based on this idea, if we use C, I and B to refer to percentage of the examples for the three types *close to centre*, *include objects* and *background*, then we have:

$$C + I + B = 100\%$$

This has the nice feature that it represents only a plane effectively reducing the parameter search space from 3D to 2D, as shown in figure 7 (a). We experimented with 28 separate proportions sampled from the plane in figure 7 (a), as shown in figure 7 (b), where each entry represents value for I for a given C and B. For example, the first two entries in the first row show that, using no background ($B = 0$), we will examine 100% C with 0% I, and 83% C with 17% I type objects, respectively.



(a)

B\C	I for each C and B (%)						
	100	83	67	50	33	17	0
0	0	17	33	50	67	83	100
17		0	17	33	50	67	83
33			0	17	33	50	67
50				0	17	33	50
67					0	17	33
83						0	17
100							0

(b)

Fig. 7. Training data proportions set.

C. Results

For each experiment with a sampled proportion, we did 100 runs. These were made up of 10 different random seeds when extracting the training data from source images, by 10 different random seeds for the GP system. Other parameters are the same as before.

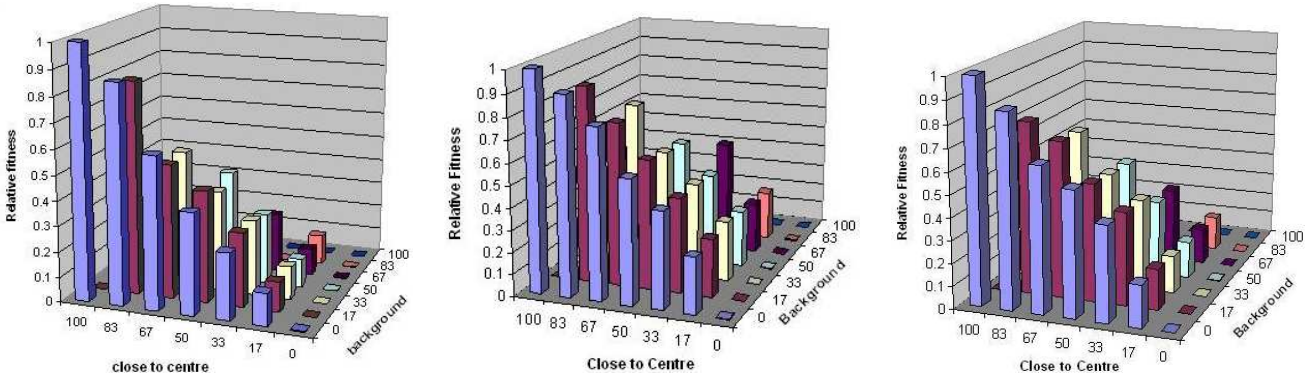
The average results on the *test set* are shown in figure 8. In the figure, the x and y axes are the C and B, and the z is the *relative* fitness for the these problems (1.0 or 100% means the ideal case).

As shown in the figure, for all the three data sets, the value of C, or the percentage of the objects for the *Close to Centre* type played an important role using our new fitness function. The best detection results were achieved with 100% examples for the *close to centre* type and the worst results were produced when we do not use any example in this type at all. The more object examples used in this type, the best results achieved. However, the *Background* type objects were not critical for these data sets. These examples did not seem to have clear bad or good influence.

These results suggest that, when using the new RLWF fitness function for these object detection, good fitness results can be achieved with only the two types, *Exact Centre* and *Close to Centre*, and most if not all object examples for the other two types *Include Object* and *Background* can be taken out from the training set.

Inspection of this reveals that, this is not only because the first two types of objects might contain the most useful information for object detection, but more importantly, because the new RLWF fitness function is capable of learning well from these two types of examples and can cope well with the goal of finding object centres from large images. This is mainly due to the fact that the RLWF fitness function consider the relative effect of the detected "objects" in different locations.

A further inspection of the use of the old fitness function



(a) (b) (c)

Fig. 8. Results of optimisation. (a) Easy; (b) Medium difficulty; (c) Hard.

reveals that the old fitness function must use object examples from all the four types. This is because the old fitness function cannot capture the relative effect information from the objects of the first two types only. This also suggests that the new fitness function is more effective than the old one for object detection, particularly when only are training examples from the first two types available.

VI. Conclusions

The goal of this paper was to develop a new fitness function for object detection and investigate its influence on optimising the training data. Rather than using a clustering process to determine the number of objects detected by the GP systems, the new fitness function introduced a weight called localisation fitness to represent the goodness of the detected objects and used weighted F-measures. To investigate the training data with this fitness function, we categorise the training data into four types. This approach is examined and compared to that with the old clustering based fitness function on three coin detection problems of increasing difficulty.

The results suggest that the new fitness function outperforms the old one by producing far fewer false alarms and spending much less training time. Further investigation on the four types of the training object examples suggests that the first two types of objects can be used to produce good detection results and that the new fitness function is effective in optimising the training data for object detection.

In the future, we will apply the new approach to other object detection problems particularly with non-circular objects.

Acknowledgement

This work was supported in part by the Marsden Fund at Royal Society of New Zealand under grant No. 05-VUW-017 and University Research Fund 6/9 at Victoria University of Wellington.

References

[1] P. D. Gader, J. R. Miramonti, Y. Won, and P. Coffield, "Segmentation free shared weight neural networks for automatic vehicle detection," *Neural Networks*, vol. 8, no. 9, pp. 1457–1473, 1995.

[2] H. L. Roitblat, W. W. L. Au, P. E. Nachtigall, R. Shizumura, and G. Moons, "Sonar recognition of targets embedded in sediment," *Neural Networks*, vol. 8, no. 7/8, pp. 1263–1273, 1995.

[3] M. W. Roth, "Survey of neural network technology for automatic target recognition," *IEEE Transactions on neural networks*, vol. 1, no. 1, pp. 28–43, March 1990.

[4] A. M. Waxman, M. C. Seibert, A. Gove, D. A. Fay, A. M. Bernandon, C. Lazott, W. R. Steele, and R. K. Cunningham, "Neural processing of targets in visible, multispectral ir and sar imagery," *Neural Networks*, vol. 8, no. 7/8, pp. 1029–1051, 1995.

[5] W. Banzhaf, P. Nordin, R. E. Keller, and F. 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.

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

[7] ——, *Genetic Programming II: Automatic Discovery of Reusable Programs*. London, England: Cambridge, Mass. : MIT Press, 1994.

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

[9] W. A. Tackett, "Genetic programming for feature discovery and image discrimination," in *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, S. Forrest, Ed. University of Illinois at Urbana-Champaign: Morgan Kaufmann, 17-21 July 1993, pp. 303–309.

[10] M. Zhang, P. Andreae, and M. Pritchard, "Pixel statistics and false alarm area in genetic programming for object detection," in *Applications of Evolutionary Computing, Lecture Notes in Computer Science, LNCS Vol. 2611*, S. Cagnoni, Ed. Springer-Verlag, 2003, pp. 455–466.

[11] M. Zhang, V. Ciesielski, and P. 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*, vol. 2003, no. 8, pp. 841–859, 2003.

[12] M. Zhang, X. Gao, and W. Lou, "Looseness controlled crossover in gp for object classification," in *Proceedings of IEEE Congress on Evolutionary Computation, a part of IEEE Congress on Computational Intelligence*, S. Lucas, Ed., Vancouver BC, Canada, 16–21 July 2006, pp. 4428–4435.

[13] M. Zhang and W. Smart, "Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification," *Pattern Recognition Letters*, vol. 27, no. 11, pp. 1266–1274, Aug. 2006, evolutionary Computer Vision and Image Understanding.

[14] W. Smart and M. Zhang, "Classification strategies for image classification in genetic programming," in *Proceeding of Image and Vision Computing Conference*, D. Bailey, Ed., Palmerston North, New Zealand, November 2003, pp. 402–407.

[15] D. Howard, S. C. Roberts, and R. Brankin, "Target detection in SAR

imagery by genetic programming," *Advances in Engineering Software*, vol. 30, pp. 303–311, 1999.
[16] U. Bhowan, "A domain independent approach to multi-class object detection using genetic programming," BSc Honours research thesis, School of Mathematical and Computing Sciences, Victoria University of Wellington, 2003.
[17] M. V. Shirvaikar and M. M. Trivedi, "A network filter to detect small targets in high clutter backgrounds," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 252–257, Jan 1995.
[18] S.-H. Lin, S.-Y. Kung, and L.-J. lin, "Face recognition/detection by probabilistic decision-based neural network," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 114–132, Jan 1997.
[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. H. W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.
[20] W. A. Tackett, "Recombination, selection, and the genetic construction of computer programs," Ph.D. dissertation, Faculty of the Graduate School, University of Southern California, Canoga Park, California, USA, April 1994.
[21] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Proceedings of the Congress on Evolutionary Computation*, vol. 2. COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea: IEEE Press, 27-30 May 2001, pp. 1070–1077. [Online]. Available: http://goanna.cs.rmit.edu.au/ toml/cec2001.ps
[22] A. Teller and M. Veloso, "A controlled experiment : Evolution for learning difficult image classification," in *Proceedings of the 7th Portuguese Conference on Artificial Intelligence*, ser. LNAI, C. Pinto-Ferreira and N. J. Mamede, Eds., vol. 990. Berlin: Springer Verlag, 3–6 Oct. 1995, pp. 165–176.
[23] ——, "PADO: Learning tree structured algorithms for orchestration into an object recognition system," Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU-CS-95-101, 1995.
[24] M. Zhang and V. Ciesielski, "Genetic programming for multiple class object detection," in *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99)*, N. Foo, Ed. Sydney, Australia: Springer-Verlag Berlin Heidelberg, December 1999, pp. 180–192, lecture Notes in Artificial Intelligence (LNAI Volume 1747).
[25] W. Kantschik, P. Dittrich, M. Brameier, and W. Banzhaf, "Metaevolution in graph GP," in *Genetic Programming, Proceedings of EuroGP'99*, ser. LNCS, R. Poli, P. Nordin, W. B. Langdon, and T. C. Fogarty, Eds., vol. 1598. Goteborg, Sweden: Springer-Verlag, 26-27 May 1999, pp. 15–28.
[26] W. Kantschik and W. Banzhaf, "Linear-graph GP—A new GP structure," in *Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002*, E. Lutton, J. A. Foster, J. Miller, C. Ryan, and A. G. B. Tettamanzi, Eds., vol. 2278. Kinsale, Ireland: Springer-Verlag, 3-5 2002, pp. 83–92. [Online]. Available: citeseer.nj.nec.com/kantschik02lineargraph.html
[27] P. A. Whigham, "Grammatically-based genetic programming," in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, J. P. Rosca, Ed., Tahoe City, California, USA, 9 July 1995, pp. 33–41. [Online]. Available: http://citeseer.ist.psu.edu/whigham95grammaticallybased.html
[28] D. Andre, "Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them," in *Advances in Genetic Programming*, K. E. Kinnear, Ed. MIT Press, 1994, pp. 477–494.
[29] G. Robinson and P. McIlroy, "Exploring some commercial applications of genetic programming," in *Evolutionary Computation, Volume 993, Lecture Note in Computer Science*, T. C. Fogarty, Ed. Springer-Verlag, 1995.
[30] J. F. Winkeler and B. S. Manjunath, "Genetic programming for object detection," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Eds. Stanford University, CA, USA: Morgan Kaufmann, 13-16 July 1997, pp. 330–335.
[31] D. Howard, S. C. Roberts, and C. Ryan, "The boru data crawler for object detection tasks in machine vision," in *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, ser. LNCS, S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, Eds., vol. 2279. Kinsale, Ireland: Springer-Verlag, 3-4 Apr. 2002, pp. 220–230.
[32] C. T. M. Graae, P. Nordin, and M. Nordahl, "Stereoscopic vision for a humanoid robot using genetic programming," in *Real-World Applications of Evolutionary Computing*, ser. LNCS, S. Cagnoni, R. Poli, G. D. Smith, D. Corne, M. Oates, E. Hart, P. L. Lanzi, E. J. Willem, Y. Li, B. Paechter, and T. C. Fogarty, Eds., vol. 1803. Edinburgh: Springer-Verlag, 17 Apr. 2000, pp. 12–21.
[33] D. Howard, S. C. Roberts, and C. Ryan, "Evolution of an object detection ant for image analysis," in *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, E. D. Goodman, Ed., San Francisco, California, USA, 9-11 July 2001, pp. 168–175.
[34] F. Lindblad, P. Nordin, and K. Wolff, "Evolving 3d model interpretation of images using graphics hardware," in *Proceedings of the 2002 IEEE Congress on Evolutionary Computation, CEC2002*, Honolulu, Hawaii, 2002.
[35] P. Nordin and W. Banzhaf, "Programmatic compression of images and sound," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Stanford University, CA, USA: MIT Press, 1996, pp. 345–350.
[36] R. Poli, "Genetic programming for feature detection and image segmentation," in *Evolutionary Computing*, ser. Lecture Notes in Computer Science, T. C. Fogarty, Ed. University of Sussex, UK: Springer-Verlag, 1-2 Apr. 1996, no. 1143, pp. 110–125.

APPENDIX
GENETIC PROGRAMMING BASICS

Constructing a GP system involves making design decisions for a number of elements of the GP system, including the representation of programs, the construction of an initial population of programs, the evaluation of programs and the construction of new population of programs. This appendix briefly describes the basic aspects of GP, including program representation, program generation, the primitive set, the fitness function, the selection mechanism, the genetic operators and the overall GP algorithm. More detailed description on GP can be seen from [6], [5].

A. Program Representation

Much of the GP work was done using LISP or LISP-like representations of the programs. A sample computer program for the algebraic equation $(x - 1) - x^3$ can be represented in LISP as the S-expression `(- (- x 1) (* x (* x x)))`. The tree representation is shown in figure 9.

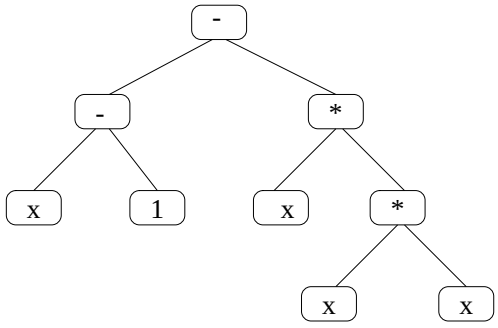


Fig. 9. A simple tree representation for a sample LISP program.

The programs are constructed from *a terminal set* and *a function set* which vary according to the problem domain. Terminals and functions are also called *primitives*, and the terminal set and the function set are combined to form a *primitive set*.

Functions in a function set form the internal nodes of the tree representation of a program. In general, there are two kinds of functions used in genetic programming. The first class refers to standard functions, such as the four arithmetic

operations. The second class comprises specific functions which vary with the problem domain.

Terminals have no arguments and form the leaves of the parse tree. Typically, terminals represent the inputs to the GP program, the constants supplied to the GP program, or zero-argument functions with side-effects executed by the GP program [5]. In any case, a terminal returns an actual numeric value without having to take an input.

B. Program Generation

There are several ways of generating programs to initialise a GP population, including *full*, *grow* and *ramped half-and-half* [6]. In the full method, functions are selected as the (internal) nodes of the program until a given depth of the program tree is reached. Then terminals are selected to form the leaf nodes. This ensures that full, entirely balanced trees are constructed. When the grow method is used, nodes are selected from either functions or terminals. If a terminal is selected, the generation process is terminated for the branch and moves on to the next non-terminal branch in the tree. In the ramped half-and-half method, both the full and grow methods are combined. Half of the programs generated for each depth value are created by using the grow method and the other half using the full method.

C. Fitness Function

Fitness is the measure of how well a program has learnt to predict the output from the input during simulated evolution. The fitness of a program generated by the evolutionary process is computed according to the fitness function. The fitness function should be designed to give graded and continuous feedback about how well a program in a population performs on the training set.

D. Selection Mechanism

The selection mechanism determines which evolved program will be used for the genetic operators to produce new individuals for the next generation during the evolutionary process. Two of the most commonly used selection methods are *proportional selection* and *tournament selection*.

In the proportional selection method [6], an individual in a population will be selected according to the proportion of its own fitness to the total sum of the fitness of all the individuals in the population. Programs with low fitness scores would have a low probability of having any genetic operators applied to them and so would most likely be removed from the population. Programs which perform particularly well in an environment will have a very high probability of being selected.

The tournament selection method [5] is based on competition within only a subset of the population, rather than the whole population. A number of programs are selected randomly according to the tournament size and a selective competition takes place. The better individuals in the tournament are allowed to replace the worse individuals. In the smallest possible tournament, two individuals can compete. The winner is allowed to reproduce with mutation and the result is returned to the population, replacing the loser of the tournament.

E. Genetic Operators

There are three fundamental genetic operators: reproduction, mutation and crossover.

Reproduction is the basic engine of Darwinian theory [6], which involves just simply copying the selected program from the current generation to the new generation. This allows good programs to survive during evolution.

Mutation operates only on a single selected program and introduces new genetic code in the new generation. This operator removes a random subtree of a selected program, then puts a new subtree in the same place. The goal here is to keep the diversity of the population in evolution.

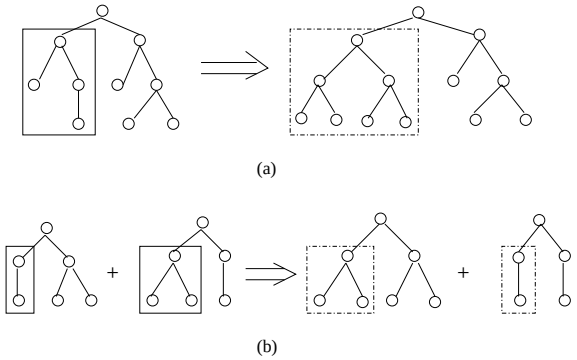


Fig. 10. Effect of genetic operators in genetic programming. (a) Mutation in GP: Replaces a random subtree; (b) Crossover in GP: Swaps two random subtrees.

Crossover takes advantage of different selected programs within a population, attempting to integrate the useful attributes from them. The crossover operator combines the genetic material of the two selected parents by swapping a subtree of one parent with a subtree of the other, and introducing two newly formed programs into the population in the next generation.

F. The GP Algorithm

The learning/evolutionary process of the GP algorithm is summarised as follows:

1) Initialise the population.
2) Repeat until a termination criterion is satisfied:
   - 2.1 Evaluate the individual programs in the current population. Assign a fitness to each program.
   - 2.2 Until the new population is fully created, repeat the following:
     - Select programs in the current generation.
     - Perform genetic operators on the selected programs.
     - Insert the result of the genetic operations into the new generation.
3) Present the best individual in the population as the output — the learned/evolved genetic program.