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

School of Mathematics, Statistics and Computer Science
Computer Science

A Study of Good Predecessor Programs for Reducing Fitness Evaluation Cost in Genetic Programming

Huayang Xie, Mengjie Zhang, Peter Andreae

Technical Report CS-TR-06/3
January 2006

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

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

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

School of Mathematics, Statistics and Computer Science
Computer Science

PO Box 600
Wellington
New Zealand

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

A Study of Good Predecessor Programs for Reducing Fitness Evaluation Cost in Genetic Programming

Huayang Xie, Mengjie Zhang, Peter Andreae

Technical Report CS-TR-06/3
January 2006

Abstract

Good Predecessor Programs (GPPs) are the ancestors of the best program found in a Genetic Programming (GP) evolution. This paper reports on an investigation into GPPs with the ultimate goal of reducing fitness evaluation cost in tree-based GP systems. A framework is developed for gathering information about GPPs and a series of experiments is conducted on a symbolic regression problem, a binary classification problem, and a multi-class classification program with increasing levels of difficulty in different domains. The analysis of the data shows that during evolution, GPPs typically constitute between less than 33% of the total programs evaluated, and may constitute less than 5%. The analysis results further shows that in all evaluated programs, the proportion of GPPs is reduced by increasing tournament size and to a less extent, affected by population size. Problem difficulty seems to have no clear influence on the proportion of GPPs.

Keywords Fitness evaluation, good predecessor programs, population clustering

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

Fitness evaluation is the most time consuming operation in Evolutionary Computation (EC) [1, 2]. Therefore, reducing the fitness evaluation cost is a key to improving the efficiency of EC. It has attracted increasing interest in both Genetic Algorithms (GAs) and Genetic Programming (GP), including fitness inheritance [3], fitness estimation [4, 5, 6], fitness case selection based fitness approximation [1, 7, 8], fitness evaluation avoidance [9], population shrinking [10] and dynamic population [11, 12, 13, 14].

We are primarily interested in manipulating the population to minimise the number of programs to be evaluated in the conventional tree-based GP system [15]. Our goal is to develop a new approach to improve both the efficiency and the effectiveness of the conventional GP system. Contrary to the common notion that almost all programs possibly contribute to the success of finding the best program, we hypothesis that in a GP run, there is only a small fraction of programs that are ancestors of the best program found. We define such programs as *Good Predecessor Programs* (GPPs). If this hypothesis is true, we will move on to develop an approach using GPPs for reducing the cost of fitness evaluation. More precisely, our whole research project consists of three stages. The first stage is to investigate GPPs to determine whether the fraction of GPPs during evolution is sufficiently small to be worth trying to identify them. The second stage will consist of looking for ways of identifying GPPs (or non-GPPs) at each generation along the evolutionary process. The final stage is to use these identifying features during evolution in order to reduce the fitness evaluation cost as much as possible while improving, or at least preserving, the effectiveness of the conventional GP system.

1.1 Goals

This paper focuses on the first stage of the research project, and involves first developing a framework to locate GPPs and gathering sufficient information of GPPs from the evolutionary process, then analysing the output of the framework to test our hypothesis. In particular, we address the following research questions:

- Whether there is only a small fraction of all programs during evolution that contribute to finding the best program; in other words, whether the number of GPPs is significantly less than the total number of programs evaluated during evolution;
- What influence, if any, genetic parameters and problem difficulty have on the size of the fraction.

The remainder of the paper is organised as follows: section 2 presents some related work; section 3 describes the framework; section 4 describes the experimental design and configuration; section 5 presents the experiment results and analyses; and section 6 gives conclusions and future work.

2 Related Work

There have been a number of approaches to reducing the cost of fitness evaluation in both GAs and GP by identifying individuals whose fitness does not need to be evaluated directly.

2.1 Studies in GAs

Sastry *et al* [3] introduced the notion of fitness inheritance and showed some very promising results in reducing the number of evaluations for OneMax problems when the population size is fixed. Ziegler and Banzhaf [2] used a meta-model of the fitness function to replace the time consuming evaluations during tournament selection in analysing evolving walking patterns for quadruped robots.

Kim and Cho [5] used k-means to cluster the whole population and used Euclidean distance to estimate the fitness values of other cluster members from the fitness value based on the cluster representative to save the fitness evaluation cost. Their method was tested on the Griewangk function, the De Jong functions, the Rastrigin function and the Schwefel function. Jin and Sendhoff [4] also used k-means to cluster the whole population. Only the chromosome closest to the cluster centre was evaluated. Fitness values of other chromosomes were estimated by a neural network ensemble. Their approach was tested on the Ackley function, the Rosenbrock function, and the Sphere function.

2.2 Studies in GP

Altenberg and Tackett [7, 8] used a small fraction of training fitness cases to evaluate a large number of offspring produced by their brood recombination crossover operator. Giacobini *et al* [1] used a statistical method to select a fraction of all fitness cases to evaluate programs in order to reduce the computational cost. They introduced a measure they called *entropy* in their study and concluded that once the number of fitness cases is greater than the entropy, a normal convergence behaviour can be observed in their boolean function and discrete step function problems.

Jackson [9] introduced a fitness evaluation avoidance method to avoid evaluating offspring generated by so-called fitness-preserving crossover. In his method, all nodes in a program are initially marked as *not-visited.* When a fitness case is fed to a fitness function and causes a node of the program to be evaluated, the node is then marked as *visited.* If a program P_1 is selected for crossover and the root of a sub-tree from another program P_2 replaces a not-visited node of P_1, then the generated child could not act differently from its parent P_1, as the inserted sub-tree will never be executed. Therefore, there is no need to re-evaluate the fitness of the offspring. The method's effectiveness depends on the fraction of nodes in the programs that are not evaluated for any of the fitness cases. For the boolean function set that Jackson used, this fraction is high; for function sets without *if* or short-circuited boolean operators, the fraction would be low, and other techniques for saving fitness evaluation would be needed.

In previous work [6], we clustered the whole population by a heuristic called *fitness-case-equivalence*, and selected a cluster representative for each cluster. The fitness value of the representative was calculated on all training cases and then directly assigned to other members in the same cluster in order to save the fitness evaluation cost.

Luke *et al* [10] proposed a shrinking strategy using a *diagonal layout* to gradually decrease the population size towards zero during a GP run. The method employs a large population at the beginning, then reduces the size linearly at each generation. They concluded that "Decreasing the population size is always as good as, and frequently better than, various fixed-sized population strategies" [10]. Fernandez *et al* [12] developed a method for solving the code bloat problem by taking advantage of the dynamic population. The method removes some individuals at every generation and compensates for the increase in the size of other individuals. They claimed that the method can save computing time while looking for solutions. Recently, Rochat *et al* [13] introduced a combination of two techniques, *island model* [16] and *plague* [17], to dynamically change the population size at run time in order to improve the quality of programs.

While almost all research in manipulating the population suggested that changing population could improve or at least preserve the effectiveness of GP, there was little explanation on why this works. In addition, the changing ratios in almost all related studies were selected without sufficient justification. Therefore, instead of just conducting many experiments on different problem domains to show that the method that we are developing works, we start with a study to test the hypothesis upon which our whole research project is based.

3 The Framework

The *GPP set* of a single GP run is the collection of all GPPs of the best program generated in a GP run. It consists of all programs in each generation that are ancestors (according to the genetic operators of crossover, mutation, and reproduction) of the program with the highest fitness value.

Our framework constructs the GPP set by recording program ancestry during evolution, and then tracing the best program found in a GP run all the way back to the initial population. The resulting GPP set is then analysed to extract high level important information in order to answer research questions. Figure 1 illustrates the relationships among the three components of the framework.

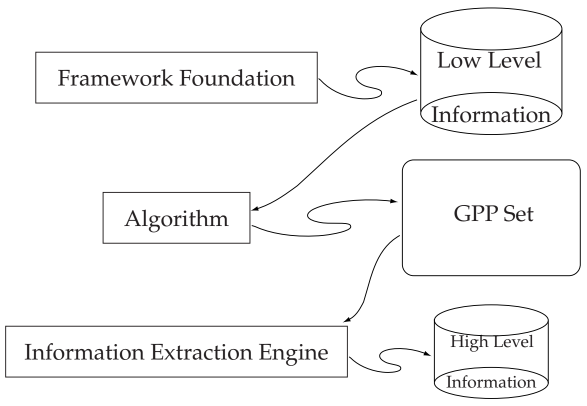

Figure 1: The structure of the framework.

A comprehensive log system is used as the framework foundation to record all necessary low level information into a detailed program log file. In order to be able to identify every program and its ancestors, we add several important *properties* to a program, including a globally unique identifier. Each entry in the log file contains the following information which can be used to provide evidence for answering our research questions:

- program ID,
- the generation in which it was created
- how the program was created/generated (new, crossover, mutation, or reproduction),
- IDs of its parents (if any)
- its size (number of nodes)
- its fitness value
- the program as a LISP expression

Table 1 shows a few example records of a detailed program log file. The following is a brief interpretation only focusing on how programs are generated. Other information can be easily understood. Programs with IDs 1 and 2 at the initial generation were randomly created (and therefore had no parents). Program 105 was generated in the 1st generation by reproducing

Table 1: Sample records in a detailed program log file.

ID	Gen	How	Parents	Size	Fitness	Program
1	0	new	-1:-1	4	34.75	If(x,3.60,x)
2	0	new	-1:-1	7	37.20	If(Sin(x),x,Mul(x,x))
⋮	⋮	⋮	⋮	⋮	⋮	⋮
105	1	repd	2:-1	7	37.20	If(Sin(x),x,Mul(x,x))
106	1	xovr	1:76	6	28.57	If(x,Add(1.74, x), x)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
218	2	muta	105:-1	8	39.74	If(Sin(x),Abs(x),Mul(x,x)))

Program 2 from the initial generation (and therefore has only one parent). Program 106 was generated by applying crossover to Programs 1 and 76 from the initial generation. Program 218 at 2nd generation was generated by mutating Program 105 from the 1st generation.

The log system also keeps track of the ID of the best program found along the evolutionary process. The best program can appear at any generation from the initial to the last. Where there is more than one program with the same fitness value, the system records only the first one found, since this will be in the earliest generation. Note that, to simplify this study so as to concentrate on what we would like to explore, the best program is elected only on the basis of its fitness value.

Once a run is completed, the algorithm constructs the GPP set by a depth first search through the log file, starting at the record of the best program and following links to parent programs, adding all the programs it finds to the GPP set.

3.1 High Level Information Extraction

While the GPP set supports the extraction of much more high level information, in this study, we only extracted the numbers of GPPs at each generation in order to identify the fraction of programs that are directly involved in producing the best program.

Figure 2 shows the number of GPPs across all generations in a sample run in our experiments. The sample run was configured with a maximum generations of 200 and a population size of 200. The x-axis shows the generation number and the y-axis shows the fraction of GPPs. In this run, the best program was found in generation 196. (Note that the evolution process was not terminated until generation 200, but no improvement was found in the last four generations.)

In the initial generation, the number of GPPs of the best program is roughly a quarter (26%) of the population. After a little fluctuation, the fraction of GPPs quickly climbed up to a peak of almost a half (47%) in generation 35. The fraction constantly flucturates during evolution, but tends to shrink towards the end of the evolution.

In this sample run, over 50% of the programs at each generation did not contribute to the final best program, and therefore evaluating their fitness was “wasted”. This suggests that there is considerable opportunity for reducing the cost of fitness evaluation if we could identify these non GPPs. Of course, a single run is not necessarily indicative of typical behaviour, and the next section describes further experiments on a range of problems.

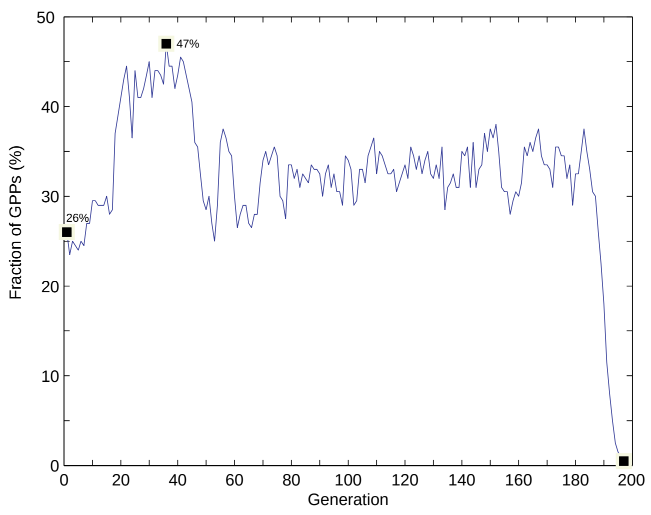

Figure 2: Fraction of GPPs in a sample run.

4 Experiment Design and Configuration

To obtain a robust measure of the fraction of GPPs in a population, we needed a range of quite different GP scenarios. We chose a symbolic regression problem, a binary classification problem, and a multi-class classification problem from different domains and with increasing levels of difficulty (low, medium and high). We also wanted to identify the effect of the GP parameters on the fraction of GPPs. There are many possible parameters that could be investigated, but we focused on two parameters — tournament size and population size — because we expected them to be more likely to influence the fraction of GPPs. Other genetic parameters will be examined in the future using the same experimental framework.

The study covers four different tournament sizes — 20, 10, 4, and 1, and six different population sizes — 100, 200, 500, 1000, 2000, 5000. These values were chosen mainly because they are commonly used. Note that tournament size 1 is equivalent to the random selection, meaning no selection pressure.

4.1 Data sets

The symbolic regression problem (Regression) is shown in equation (1). We generated 100 fitness cases by assigning equally spaced real numbers in (-10,10] to x.

$$f(x) \quad = \quad \begin{cases} x^2 - x & , x \geq 0 \\ sin(x) + \frac{1}{x} & , x < 0 \end{cases} \tag{1}$$

The binary classification problem involves determining whether examples represent a normal liver or a liver disorder. The dataset is BUPA Liver Disorders dataset (BUPA) chosen from the UCI Machine Learning repository [18]. BUPA consists of 345 data examples, each described by six numeric features.

The multi-class classification problem involves classifying four types of vehicles: *opel, saab, bus* and *van*. The dataset is called Vehicle Silhouette (Vehicle), which was also chosen from the UCI Machine Learning repository. Vehicle consists of 846 data examples, each described by 18 numeric features.

For each classification problem, the data set is evenly and randomly split into the training and test sets for each individual GP run.

4.2 Classification

The programs output real values; for the classification tasks, these values must be converted into appropriate class labels.

For the liver disorder binary classification problem, the classification of an example is given by the sign of the program output — if the output of a program on a fitness case is negative, the program classifies the fitness case as *disorder*, otherwise, as *normal*.

For the vehicle multi-class classification problem, we use the following classification rule [19], where r is the program output. It is highly possible that the rule is not optimal. However, it is not the focus of this study to properly set the rule in order to find an optimal solution of the vehicle problem.

$$class = \begin{cases} opel & , \quad r \geq 10 \\ saab & , \quad 0 \leq r < 10 \\ bus & , \quad -10 \leq r < 0 \\ van & , \quad r < -10 \end{cases} \tag{2}$$

4.3 Function Set

The function set for the three problems is listed below:

$$\text{Function Set} \quad = \{+, -, *, /, abs, sqrt, if, sin\} \tag{3}$$

The $+$, $-$, and $*$ operators have their usual meanings — addition, subtraction, and multiplication. The / operator represents "protected" division which is the usual division operator except that a division by zero gives a result of zero. Each of these four functions takes two arguments. The *if* function takes three arguments: if the first argument is positive, the *if* function returns its second argument; otherwise, it returns its third argument. The remaining unary functions also have their usual meanings. Note that zero will be returned if a function encounters an invalid argument.

4.4 Terminal Sets

We have three terminal sets, one for each problem, with a different number of variables or features in each set. The terminal set for Regression includes a single variable x. The terminal set used in BUPA includes six numerical features. The terminal set used in Vehicle includes 18 numerical features extracted from images of the types of vehicles. More details about those features can be found in [18].

Each terminal set also includes real valued random constants in the range [-5.0, 5.0]. Note that, once random constant numbers are generated for a program, they will remain unchanged during the evolutionary recombination (crossover and reproduction). However, they may be replaced by new random constant numbers when the mutation operator is applied.

4.5 Fitness Function

For the symbolic regression problem, the fitness of a program was given by the root mean square (RMS) of the differences between the outputs of a program and the expected outputs for each fitness case.

For the classification problems, we used the classification error rate on the training data set as the fitness function. The classification error rate of a program is the fraction of the fitness cases in the training set that are incorrectly classified by the program. Therefore, the best fitness value is zero, meaning that no fitness case has been incorrectly classified.

4.6 Other Genetic Parameters and Termination Criteria

The ramped half and half method is used to create new programs. The minimum depth of creation is three and the maximum is five. The crossover rate, the mutation rate, and the reproduction rate are 85%, 10%, and 5% respectively. Elitism is also used in our GP system.

The learning/evolutionary process is terminated when either of the two following conditions is met:

- The problem has been solved. This means the RMS of errors of the best program is zero for the symbolic regression problem, or the classification error rate of the best program on the training data set becomes zero percent for classification problems.
- The number of generations reaches the pre-defined value. In this experiment, the maximum limit of generation is 200. This means if a certain run fails to reach an optimal solution on or before the last generation, the program will be terminated automatically.

4.7 Experiment Configuration

We tested six population sizes and four tournament sizes on each of the three problems, giving 72 experiments in total. In each experiment, we repeated the evolutionary process 100 times randomly and independently. Since precise CPU time measurement is not important for the study, the experiments were run on a Sun Grid Engine (`http://gridengine.sunsource.net`) involving tens of workstations with identical hardware and software.

5 Results and Analysis

This section presents the experimental results. We analysed the ratio of GPPs to the total evaluated programs in a GP run to investigate whether there is only a small fraction of programs relating to the success of finding the best program during evolution. We also analysed the relationships between the GPP ratio and the three factors — tournament size, population size, and problem difficulty — to investigate whether the GPP ratio is influenced by any of these factors. Note that the Bivariate Correlation Analysis [20] is used to provide empirical evidences for supporting or rejecting those relationships. The test calculates a correlation coefficient ($r \in [-1, 1]$) between two variables. $r = -1$ suggests two variables have a perfect negative correlation, $r = 1$ suggests a perfect positive correlation, and $r = 0$ suggests no correlation at all. The confidence level is calculated from the probability value (p). In this study, the commonly used 95% confidence level is used, meaning that p does not exceed 0.05.

5.1 The Average GPP Ratio

The results of the experiments are presented in Table 2 and Figure 3. The table presents the mean and standard deviation of the GPP ratio for the 100 runs of each experiment; the figure presents the dynamic behaviour of the GPP ratio for several particular experiments.

Table 2 gives detailed figures summarising the mean and standard deviation (shown after the $\pm$ sign) of the GPP ratio over 100 runs of each of the 72 experiments. The table is organised into four parts according to the four tournament sizes. Within each part, population sizes are shown as rows and problems are shown as columns. For example, the first cell of the top-left part of the table shows that, for the symbolic regression problem, and in a run with a tournament size of 20 and a population size of 100, on average, only 15.13% programs are GPPs amongst the total evaluated programs.

Table 2: Average ratio of GPPs to all programs evaluated during evolution (%).

Pop Size	Regression	BUPA	Vehicle	Regression	BUPA	Vehicle
	tournament size 20			tournament size 10		
100	15.13 ± 7.89	10.70 ± 6.99	12.52 ± 6.95	16.95 ± 7.69	13.25 ± 8.04	15.64 ± 7.28
200	13.62 ± 7.06	10.33 ± 6.03	11.55 ± 6.38	15.09 ± 7.02	12.47 ± 6.04	13.09 ± 6.14
500	9.25 ± 4.71	7.72 ± 4.14	8.91 ± 4.77	12.50 ± 5.19	9.78 ± 3.98	11.11 ± 4.28
1000	8.90 ± 4.18	6.86 ± 3.44	6.68 ± 2.98	11.95 ± 4.46	8.14 ± 3.47	10.35 ± 3.67
2000	6.85 ± 3.55	5.48 ± 3.17	4.92 ± 2.37	9.40 ± 4.01	7.99 ± 2.62	9.25 ± 2.47
5000	**3.89** ± 3.22	3.99 ± 2.14	4.35 ± 1.77	6.46 ± 3.88	7.42 ± 2.53	8.86 ± 2.11
	tournament size 4			tournament size 1		
100	23.30 ± 7.41	18.69 ± 7.60	22.53 ± 7.65	30.78 ± 14.20	30.97 ± 14.81	**33.30** ± 13.09
200	23.40 ± 5.68	19.82 ± 6.71	22.41 ± 5.44	29.41 ± 15.62	29.29 ± 13.17	32.67 ± 12.91
500	23.20 ± 4.97	19.11 ± 5.17	21.31 ± 4.50	26.49 ± 15.55	28.74 ± 14.91	27.86 ± 15.92
1000	22.90 ± 4.15	18.81 ± 4.94	21.01 ± 4.56	23.76 ± 16.64	24.04 ± 15.45	31.56 ± 14.05
2000	19.20 ± 6.05	19.56 ± 4.64	21.71 ± 3.53	25.03 ± 16.36	26.22 ± 14.18	30.95 ± 15.09
5000	16.38 ± 6.77	18.38 ± 4.85	21.44 ± 3.42	21.10 ± 15.94	23.02 ± 15.41	32.51 ± 14.02

Figure 3 illustrates several randomly chosen sample runs from the experiments of the three problems for each of the six population sizes with the arbitrarily chosen tournament size of 20. There are six charts in the figure corresponding to the six different population sizes respectively. Each chart shows a run for each of the three problems, where the dash line stands for Regression, the thin solid line stands for BUPA, and the thick solid line stands for Vehicle.

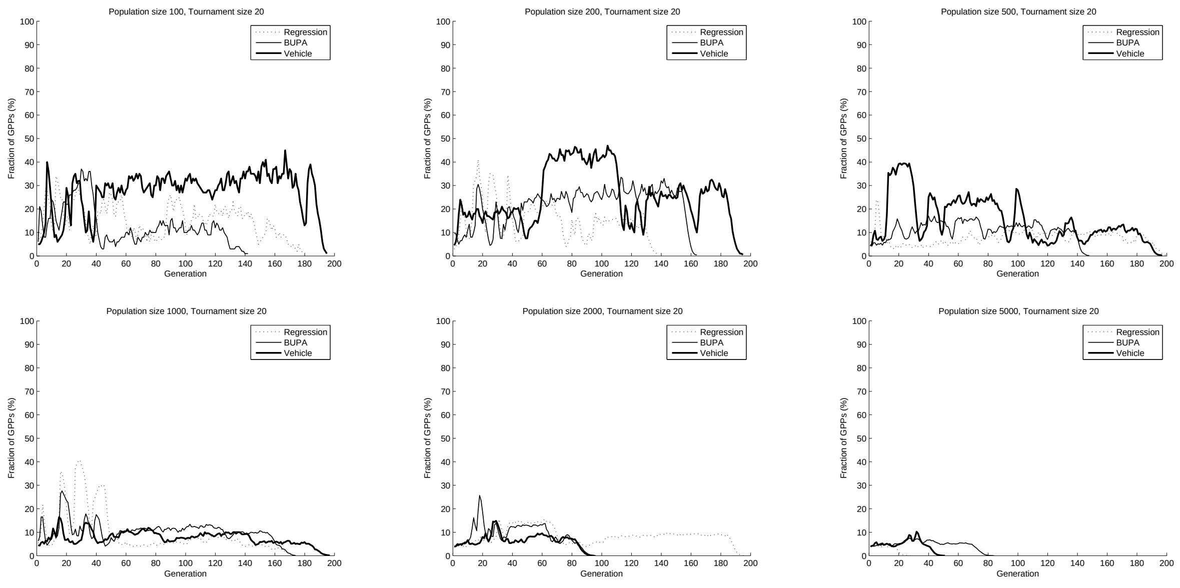

Figure 3: More sample runs with tournament size 20 for Regression, BUPA, and Vehicle problems using six different population sizes

From a preliminary consideration of this data, it is clear that many of the evaluated programs do not contribute to the best program in a run. For random selection (tournament size of 1) the GPP ratio can frequently rise to about 50%, but with just some fitness-based selection (tournament size of 4), this drops to below 33%, even with small population sizes. With large populations and strong fitness-based selection (large tournament sizes), the GPP ratio dropped below 10% for most runs. Therefore, there are many programs whose fitness was evaluated "unnecessarily". Being able to identify these programs before evaluating their fitness could reduce the total fitness evaluation cost very significantly.

It is also clear that the GPP ratio varies with different parameters. We were particularly interested in the effect of tournament size, population size, and problem difficulty. Figure 4 presents a plot of the GPP ratios against population size for each problem category and for each tournament size. Further examination of this data led to the relationships discussed below.

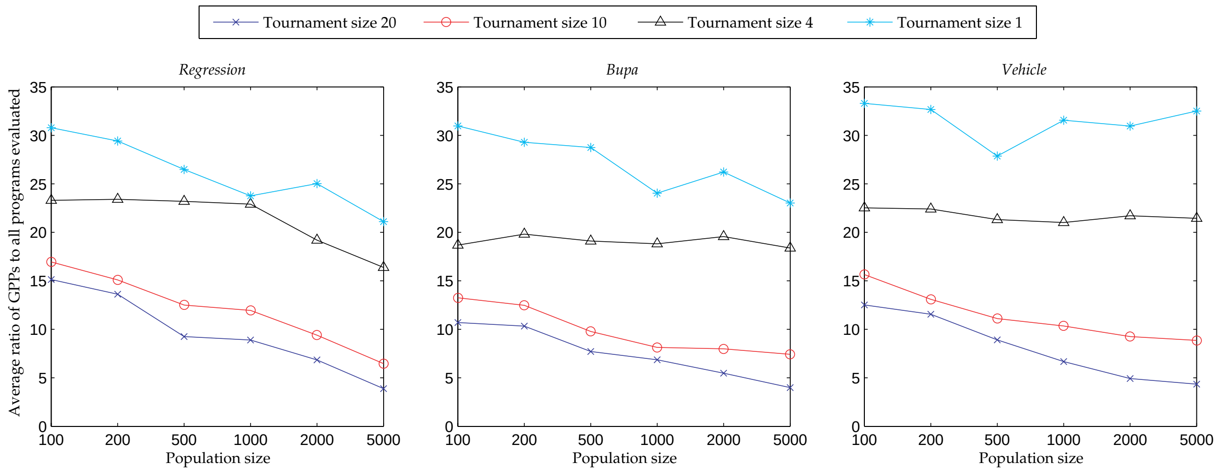

Figure 4: Average GPP ratio against population size for each problem and tournament size.

5.2 GPP Ratio and Tournament Size

The four lines in each graph of Figure 4 make it clear that the GPP ratio decreases with increasing tournament size for all problems and all population sizes. Bivariate Correlation Analysis gives a strong negative correlation ($r = -0.852$) between tournament size and the GPP ratio, and the correlation is significant at a 0.01 level. Hence the relationship between the GPP ratio and tournament size is statistically supported.

A possible explanation is that bigger tournament sizes decrease the chance of any low fitness program winning a tournament and contributing to the next generation. The ancestors of the next generation are likely to be confined to a small set of high fitness programs, many of which will win multiple tournaments and dominate the new population, and hence constrain the set of GPPs to be small relative to the whole population.

Interestingly, of the four different tournament sizes, the results demonstrate that size 4 has a special characteristic: the GPP ratios across different population sizes in experiments with the tournament size 4 are more stable than those in experiments with other tournament sizes on all three problems. It suggests that a tournament size around 4 may be able to provide a more consistent evolutionary process, regardless of population size, than any of the other tournament sizes change. We will further investigate this special characteristic in the future.

5.3 GPP Ratio and Population Size

The relationship between the GPP ratio and population size is not as clear from Figure 4. Bivariate correlation gives a correlation coefficient of -0.219 and the significance level is 0.065, indicating a less robust negative correlation between the GPP ratio and population size. However, the negative correlation between the GPP ratio and population size is much stronger for larger tournaments (20 or 10) across all three problems. For the smaller tournaments (4 or 1), the lines are either fluctuating or almost flat. The results suggest that there is a negative correlation between the GPP ratio and population size but the correlation is masked at small tournament sizes.

5.4 GPP Ratio and Problem Difficulty

As stated in section 4, Regression, BUPA and Vehicle represent three levels of difficulties - low, medium and high. Figure 4 shows no clear relationship between the GPP ratio and problem difficulty. Furthermore, Bivariate Correlation Analysis yields a correlation coefficient of 0.044 (where that 0 means no correlation at all) and insignificant at the 0.717 level (where the significance level should be less than 0.05). Therefore no correlation was suggested between the GPP ratio and problem difficulty in our experiments.

However, we are not confident that there is no correlation with problem difficulty. The three problems represent only a small sample of coarse levels of problem difficulties, and may not be sufficient to yield a sound correlation. Also, the correlation with problem difficulty may be masked by other factors, including other un-investigated genetic parameters. It will require further experiments to determine whether there is or is not any correlation with problem difficulty.

6 Conclusions and Future Work

The goal of this paper was to study the feasibility of using GPPs for reducing fitness evaluation cost in the conventional tree-based GP system. A framework was developed to gather information about GPPs. A series of experiments was conducted on a symbolic regression problem, a binary classification problem, and a multi-class classification program with increasing levels of difficulty in different domains. The resulting data was analysed.

The analysis of the ratio of the GPPs shows that with strong fitness-based selection and large population sizes, only a small fraction of programs amongst the total evaluated programs in a run are ancestors of the best program — less than 5% of total programs in some cases.

The analysis of the relationships between the GPP ratio and three factors — tournament size, population size, and problem difficulty — show that the GPP ratio is strongly negatively influenced by tournament size, and also negatively influenced by population size, but less strongly.

We were surprised that there is no evidence for a correlation between the GPP ratio and problem difficulty in our experiments. We will choose more problems representing finer difficulty levels in order to further investigate the relationship between the GPP ratio and problem difficulty. We will also study other genetic parameters and use multivariate statistical methods to further examine the relationship between the GPP ratio and problem difficulty.

We will also extend our experiments to cover other benchmarks commonly used in GP studies, including artificial ant and even parity problems, to provide more general conclusions.

As our ultimate project goal is to develop a new approach that uses the analysis of GPPs to reduce the fitness evaluation cost as much as possible without affecting the effectiveness of the conventional GP system, we will further investigate the GPP ratio at each generation, especially at the initial generation, explore ways to identify GPPs and/or the correlates, and develop mechanisms to use GPPs and/or the correlates during the evolutionary process.

Acknowledgment

We would like to thank Neale Ranns for his wonderful GP package (Gouda), his endless technical support and useful discussions.

References

[1] M. Giacobini, M. Tomassini, and L. Vanneschi, “Limiting the number of fitness cases in genetic programming using statistics,” in *PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature.* London, UK: Springer-Verlag, 2002, pp. 371–380.

[2] J. Ziegler and W. Banzhaf, “Decreasing the number of evaluations in evolutionary algorithms by using a meta-model of the fitness function,” in *Genetic Programming, Proceedings of EuroGP’2003*, ser. LNCS, C. Ryan and et al, Eds., vol. 2610. Springer-Verlag, 2003, pp. 264–275.

[3] K. Sastry, D. E. Goldberg, and M. Pelikan, “Don’t evaluate, inherit,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, L. Spector and et al, Eds. San Francisco, California, USA: Morgan Kaufmann, 2001, pp. 551–558. [Online]. Available: citeseer.ist.psu.edu/article/sastry01dont.html

[4] Y. Jin and B. Sendhoff, “Reducing fitness evaluations using clustering techniques and neural networks ensembles,” in *Genetic and Evolutionary Computation Conference*, ser. LNCS, vol. 3102. Springer, 2004, pp. 688–699.

[5] H.-S. Kim and S.-B. Cho, “An efficient genetic algorithms with less fitness evaluation by clustering,” in *Proceedings of IEEE Congress on Evolutionary Computation.* IEEE, 2001, pp. 887–894.

[6] H. Xie, M. Zhang, and P. Andreae, “Population clustering in genetic programming,” in *Proceedings of the 9th European Conference, EuroGP 2006*, ser. LNCS, P. Collet and et al, Eds., vol. 3905. Springer-Verlag, 2006, pp. 190–121.

[7] L. Altenberg, “Emergent phenomena in genetic programming,” in *Proceedings of the Third Annual Conference on Evolutionary Programming*, A. V. Sebald and L. J. Fogel, Eds. World Scientific, 1994, pp. 233–241.

[8] W. A. Tackett, “Recombination, selection, and the genetic construction of computer programs,” Ph.D. dissertation, University of Southern California, Los Angeles, CA, USA, 1994.

[9] D. Jackson, “Fitness evaluation avoidance in boolean GP problems,” in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, D. Corne and et al, Eds., vol. 3. Edinburgh, UK: IEEE Press, 2-5 Sept. 2005, pp. 2530–2536.

[10] S. Luke, G. C. Balan, and L. Panait, “Population implosion in genetic programming.” in *Genetic and Evolutionary Computation – GECCO-2003*, ser. LNCS, E. Cantú-Paz and et al, Eds., vol. 2724. Chicago: Springer-Verlag, 12-16 July 2003, pp. 1729–1739.

[11] M. E. Balazs and D. L. Richter, “A genetic algorithm with dynamic population: Experimental results,” in *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, S. Brave and A. S. Wu, Eds., Orlando, Florida, USA, 13 July 1999, pp. 25–30.

[12] F. Fernandez, M. Tomassini, and L. Vanneschi, “Saving computational effort in genetic programming by means of plagues,” in *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, R. Sarker and et al, Eds. Canberra: IEEE Press, 8-12 Dec. 2003, pp. 2042–2049.

[13] D. Rochat, M. Tomassini, and L. Vanneschi, “Dynamic size populations in distributed genetic programming,” in *Proceedings of the 8th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, M. Keijzer and et al, Eds., vol. 3447. Lausanne, Switzerland: Springer, 2005, pp. 50–61.

[14] M. Tomassini, L. Vanneschi, J. Cuendet, and F. Fernandez, “A new technique for dynamic size populations in genetic programming,” in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation.* Portland, Oregon: IEEE Press, 20-23 June 2004, pp. 486–493.

[15] J. R. Koza, *Genetic Programming — On the Programming of Computers by Means of Natural Selection.* Cambridge: MIT Press, 1992.

[16] R. Tanese, “Distributed genetic algorithms,” in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, Ed. Morgan Kaufmann Publishers, 1989, pp. 434–440.

[17] L. Vanneschi, “Theory and practice for efficient genetic programming,” Ph.D. dissertation, Faculty of Sciences, University of Lausanne, Switzerland, 2004.

[18] C. B. D.J. Newman, S. Hettich and C. Merz, “UCI repository of machine learning databases,” 1998. [Online]. Available: http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html

[19] W. Smart and M. Zhang, “Applying online gradient descent search to genetic programming for object recognition,” *Australian Computer Science Communications (Data Mining, CRPIT 32)*, vol. 26, pp. 133–138, January 2004.

[20] R. H. Lindeman, P. F. Merenda, and R. Z. Gold, *Introduction to Bivariate and Multivariate Analysis.* Scott, Foresman and Company, 1980.

[21] M. Fuchs, “Large populations are not always the best choice in genetic programming,” in *GECCO*, 1999, pp. 1033–1038.