DESIGN AND USAGE OF A NEW BENCHMARK PROBLEM FOR GENETIC PROGRAMMING

Emin Erkan Korkmaz[1] and Göktürk Üçoluk[2]

[1] Computer Engineering Department
Middle East Technical University
06531, Ankara, TURKEY
korkmaz@metu.edu.tr,
WWW home page: http://www.ceng.metu.edu.tr/~korkmaz

[2] Computer Engineering Department
Middle East Technical University
06531, Ankara, TURKEY
ucoluk@metu.edu.tr,
WWW home page: http://www.ceng.metu.edu.tr/~ucoluk

Abstract. Not so many benchmark problems have been proposed in the area of *Genetic Programming* (GP). In this study, a new artificial benchmark problem is designed for GP. The different parameters that can be used to tune the difficulty of the problem are analyzed. Also, the initial experimental results obtained on different instances of the problem are presented.

1 Introduction

Benchmark Problems are important in order to judge the effectiveness of a new approach proposed in a research area. It is possible to get more insight about the contribution of a methodology by using a tunable benchmark problem. Not so many benchmark problems have been proposed in the area of *Genetic Programming* (GP). However, there are some problems widely used by different GP researchers. Usually, the results obtained in these commonly used domains are presented in order to denote the effectiveness of an approach. The problem of programming an artificial ant to follow the *Santa Fe trail* [5], the symbolic regression problems [4] and the N-parity problem [2, 1, 8, 9] are example real world domains which have been used as benchmark problems in GP.

However, using a real-world domain as a benchmark problem has certain drawbacks. It is usually difficult to tune a real-world problem. The different instances of the problem can be disproportionate with each other in terms of difficulty and the size of the search space. The results obtained on a specific instance of the problem might be misleading in terms of the general behavior of the methodology used. Hence the usage of a tunable artificial benchmark problem gains importance for testing the effectiveness of a method.

This paper presents the design and usage of a new benchmark problem for GP. In the following section, previous work in the area is presented. Then, in section 3, the design of the problem is introduced. In section 4, the initial experimental results obtained on different instances of the problem are presented. The experiments are carried out in order to determine the behavior of GP on the problem. Lastly, in section 5, the conclusion is given.

2 Related Work

As noted in the previous section, it is difficult to find benchmark problems designed for GP. However certain attempts do exist in the literature. For instance, in [7], certain problems which can be used as benchmark tests are proposed.

The most notable work in the area is the benchmark problem proposed in [6]. The formalization used by [6] shares some characteristics with the *Royal Road Problem*. This is a commonly used benchmark problem for tuning the genetic parameters in the field of GA, [3]. The benchmark problem proposed in [6] is an important attempt to transfer the issues addressed in creation of the Royal Road Problem, to the field of GP. These issues focus on the important aspects needed to consider a problem as a benchmark for the evolutionary computation research. The goal is to define a problem, which

- possesses a controllable level of difficulty,
- has clearly defined building blocks,
- can be reduced to various instances to be used to tune the genetic parameters.

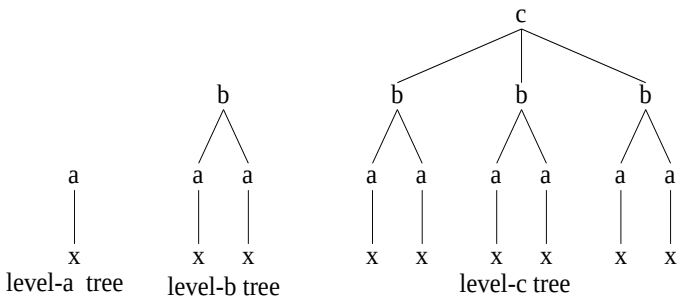

Fig. 1. Perfect trees of different levels.

The formalization in [6] is based on the definition of a *"perfect tree"* of some depth. For instance a level-a tree is perfect when its root is the function a with a single child. Similarly a perfect level-b tree's root should be function b having two perfect level-a trees as children. Certainly a perfect level-c tree will have three perfect level-b trees connected to root c and so on. The terminal set consists of a single element x. Note that the series of functions $a, b, c, d...$ are defined with increasing arity. The perfect trees of levels a, b and c are presented in Figure 1.

The raw fitness of a tree is defined as the score of its root. On the other side the score of each function is calculated by adding the weighted scores of its children. The weight is a constant larger than one ($FullBonus$) if a child is a perfect tree of the appropriate level. When the child is not a perfect tree, the weight turns out to be a constant smaller or equal to one ($Penalty$ or $Partialbonus$) depending on the child's configuration. If the child has the correct root $Partialbonus$ is used. However, if the root of the child is incorrect too, $Penalty$ is used as the weight. If the root itself is the correct root of a perfect tree, then the obtained score is multiplied by $CompleteBonus$. It is stated that typical values that can be used are: $FullBonus = 2, PartialBonus = 1, Penalty = \frac{1}{3}$, and $CompleteBonus = 2$. It is suggested that perfect trees of different depths can be used as benchmark problems for GP.

The proposed definition in [6] has certain drawbacks. It can be claimed that the most important factor that effects the performance of GP is the *deception* of the problem at hand. This notion can be observed in a variety of real world domains. In such domains, the genetic search is easily misled to a local optimum by the building blocks and such problems are called *deceptive problems*. Hence a qualified benchmark problem should provide a mechanism which has control over the deceptiveness of the problem. Deception usually appears as an outcome of *epistasis*. In other words, the interdependency among the subparts of the chromosome should have an influence on the global fitness of the chromosome. However, the formalization provided by [6] enables each child to contribute to the global fitness independent of other children. Furthermore, since the functions are defined with increasing arity, the search space increases rapidly for high levels. The problem is too difficult after level d and e. On the other hand, levels a, b and c are too simple. It is possible for the solution to appear in the initial random population in these levels. Lastly using more than one constant makes the definition unnecessarily complicated.

3 A New Benchmark Problem

Considering the drawbacks mentioned in the previous section, it has been decided to simplify and reorganize the definition of perfect tree. The aim is to obtain a tunable benchmark problem where epistasis can easily be controlled.

The new perfect tree is defined to be a full binary tree of some depth. The function set consists of n functions with arity two ($F = \{X_1, X_2, ..X_n\}$). The terminal set consists of a single terminal element t, ($T = \{t\}$). The raw fitness of a tree is again defined as the score of its root. The fitness of a terminal node is simply defined as 1. The fitness of an internal node is defined as the sum of the fitness values of its two children. The two constraints used to create epistasis for the problem are the following.

When the children of an internal node are not terminal elements:

(i) The index of the parent function should be smaller than the index of its children.
(ii) The index of the right-hand child should be larger than the index of the left-hand child.

The fitness function for an internal node is defined as:

$$f(P) = \begin{cases} f(C_1) + f(C_2) & (i) \\ C_{epis} \cdot f(C_1) + f(C_2) & (ii) \\ f(C_1) + C_{epis} \cdot f(C_2) & (iii) \\ C_{epis} \cdot f(C_1) + C_{epis} \cdot f(C_2) & (iv) \end{cases} \quad (1)$$

In Equation 1, part (i) is chosen if both of the children are terminal elements or none of the constraints are violated. If the first constraint is violated by any child, the fitness of that child is multiplied with a constant smaller than one. This constant is called the *epistasis constant* (C_{epis}). Parts (ii) and (iii) in Equation 1 reflect this situation. Lastly part (iv) is used when both of the children violate the first constraint or when the second constraint cannot be achieved.

Note that these two constraints create interdependency among the subparts of a chromosome. When the epistasis constant is decreased the interdependency increases. It becomes impossible for a subpart of the chromosome to contribute to the global fitness independent of other parts. A well-fit chromosome can easily be ruined when a node that violates a constraint appears after a recombination operation. Hence, fitnesses of similar chromosomes might differ remarkably and the search space becomes discontinuous. What is more is that, when the function set is kept small, it becomes more probable to be stuck in a local minima. Note that the index of the functions has to increase in the down direction of a chromosome. Therefore a wrong choice close to the root might make it impossible to form a perfect tree.

Note that three different parameters exist for tuning the difficulty of the benchmark problem proposed. The first parameter is the depth of perfect tree to be searched. The epistasis constant (C_epis) and the size of the function set are the other two parameters which can be used to tune the problem.

4 Experimental Results

The experiments are carried out by using a perfect tree with a fixed depth of four. Increasing the depth of the perfect tree results an exponential enlargement in the search space. Therefore, such a moderate size is chosen. The change in the performance of GP is observed based on the other two parameters; namely the epistasis constant and the size of the function set.

In figure 4, a possible solution for depth four is given. Note that seven different functions have been used to build the solution and other solutions would exist if a larger function set is used.

Consider the two parameters for controlling the difficulty of the problem. When you enlarge the function set, the number of possible solutions increases, hence it becomes easier to reach to a solution in the search space. On the other side when the epistasis constant is decreased the interdependency increases and it turns out to be more difficult to reach to the solution. Two different sets of experiments are carried out in order to get insight about the behavior of GP on the benchmark problem. In the first phase, the function set is fixed and the

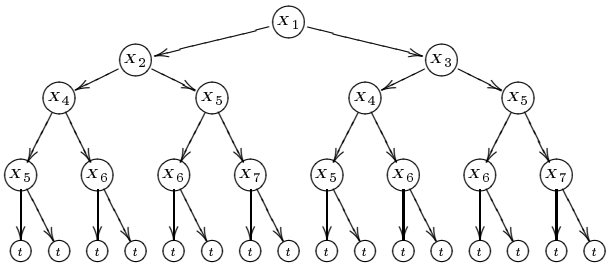

Fig. 2. A possible solution of depth four.

behavior is observed based on the change in the epistasis constant. Then in the second phase the reverse procedure is carried out and different function sets are used with a fixed epistasis constant.

The same genetic parameters are used for the two phases and these parameters are set as follows

- Population size = 100
- Crossover at function point fraction = 0.1
- Crossover at any point fraction = 0.7
- Reproduction fraction = 0.1
- Mutation fraction = 0.1
- Number of Generations = 3000
- Selection Method: Fitness Proportional.

The results are obtained by using the average of 100 different runs. Note that the largest value that can be obtained by the fitness function proposed in Equation 1 evaluates to 16 for the perfect tree of depth four. If the best fitness value is to be set as zero, then the fitness function would be

$$\hat{f}(T) = 16 - f(Root(T)), \tag{2}$$

where f is the function in Equation 1, T is a tree and *Root* returns the root node of a given tree.

In figure 4 the behavior of GP is presented based on the change in epistasis constant. The presented graphs are the output of three different constants; 0.5, 0.25 and 0.002. The last constant is chosen to be close to zero and as seen in the figure the worst performance is obtained with this constant. The constant 0.5 can be considered to be quite high and GP has the best performance in this instance. Also a third constant is chosen (0.25) in between the difficult and easy instances. As seen in the figure, it is possible to obtain instances with different difficulty levels based on the epistasis constant used. It is observed that the change in performance related to the change in epistasis constant is quite uniform.

In figure 4, the epistasis constant is fixed as 0.25 and the effect of the function set size on the performance of GP is analyzed. The three function sets used for the trials are $F = \{X_1, X_2, ..., X_7\}$, $F = \{X_1, X_2, ..., X_7, X_8\}$ and $F = \{X_1, X_2, ..., X_7, X_8, X_9\}$. As seen in the figure, function set size can be used as a parameter which would change the performance of GP on the problem. A change in this parameter is observed to have a gradual effect on the result and does not lead to a chaotic/disproportional behavior. This is also an asset compared to the work of [6].

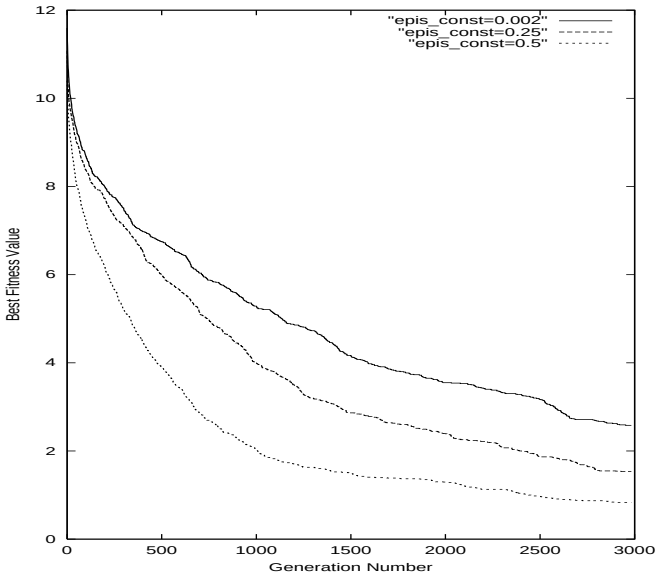

Fig. 3. Comparison of three different epistasis constants. The size of the function set used is 8 for all of the three trials. Best fitness means obtained throughout 100 runs are presented for each epistasis constant.

5 Conclusion

The goal of this research was to design a tunable benchmark problem which can be used to judge the effectiveness of different methodologies in the field of GP. We have developed a problem where the tree shape has effect on the evaluation function. The proposed problem is inspired by the work of [6] and has certain common characteristics with the definition proposed by [6]. However, it is claimed that it is quite easier to tune and obtain different difficulty levels with the new definition proposed in this study. The experimental results verify that it is possible to obtain different instances which are quite proportionate with respect to each other in terms of difficulty. It is possible to get more insight about the effectiveness of a GP methodology and observe the behavior of the method against the change in epistasis and deception by using the different instances of the problem proposed.

References

1. Chellapilla, K. A Preliminary Investigation into Evolving Modular Programs without Subtree Crossover. Genetic Programming 1998: *Proceedings of the Third Annual Conference*
2. Chris Gathercole and Peter Ross. Tackling the Boolean Even N Parity Problem with Genetic Programming and Limited-Error Fitness. Genetic Programming 1997:

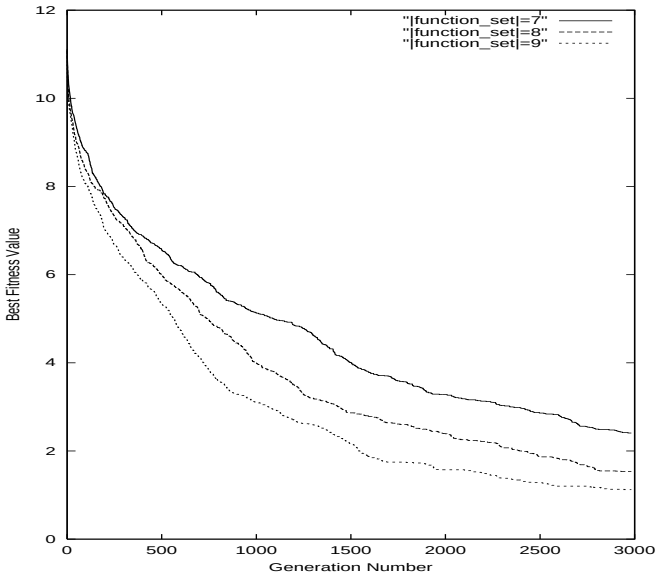

Fig. 4. Comparison of three different function sets. The epistasis constant used for all of the three trials is 0.25. Best fitness means obtained throughout 100 runs are presented for each function set.

Proceedings of the Second Annual Conference, pp. 119–127, Stanford University, CA, USA, 1997.

3. Janes, T. A Description of Holland's Royal Road Function. *Evolutionary Computation*, 2(4), pp. 409-415, 1994.
4. Koza, J. R. Genetic Programming. On the Programming of Computers by Means of Natural Selection. *The MIT Press*, London. 1992.
5. W. B. Langdon. Better Trained Ants. *Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming*, Paris, France, 1998.
6. W.F. Punch, D. Zongker and E.D. Goodman. The Royal Tree Problem, a Benchmark for Single and Multi-population Genetic Programming. Appears *in Advances in Genetic Programming g 2*, MIT Press, pg 299-316, 1996.
7. Tackett, W. A. *Recombination, Selection and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, April 1994.
8. Terence Soule and James A. Foster. Code Size and Depth Flows in Genetic Programming. Genetic Programming 1997: *Proceedings of the Second Annual Conference*, Stanford University, CA, USA, pp. 313–320, 1997.
9. Man Leung Wong and Kwong Sak Leung. Evolving Recursive Functions for the Even-Parity Problem Using Genetic Programming. *Advances in Genetic Programming 2*, MIT Press, pp. 221–240, 1996.