Maintaining the Diversity of Genetic Programs

Anikó Ekárt[1] and **S. Z. Németh**[2]

[1] School of Computer Science, The University of Birmingham,
Birmingham, B15 2TT, United Kingdom
A.Ekart@cs.bham.ac.uk

[2] Computer and Automation Research Institute, Hungarian Academy of Sciences,
1518 Budapest, P.O.B. 63, Hungary
snemeth@sztaki.hu

Abstract. The loss of genetic diversity in evolutionary algorithms may lead to suboptimal solutions. Many techniques have been developed for maintaining diversity in genetic algorithms, but few investigations have been done for genetic programs. We define here a diversity measure for genetic programs based on our metric for genetic trees [3]. We use this distance measure for studying the effects of fitness sharing. We then propose a method for adaptively maintaining the diversity of a population during evolution.

1 Introduction

The search space of a search problem may contain several local extremes (peaks or valleys, depending on whether the problem concerns maximization or minimization). In artificial genetic search, the population may converge to one of these extremes, i.e., the individuals of the population get closer and closer, and arbitrarily close, to this extreme. Several so-called *niching techniques* have been introduced for preventing the early convergence to a local optimum [5]. A population of genetic programs is diverse if it contains samples of as many regions of the search space as possible. The niching methods preserve the diversity of a population by keeping the individuals reasonably away from each other.

Fitness sharing [5] was introduced as a technique for maintaining population diversity in genetic algorithms. Fitness sharing treats fitness as a shared resource of the population, and thus requires that similar individuals share their fitness. The fitness of each individual is worsened by its neighbours in the search space: the more and the closer neighbours an individual has, the worse its fitness becomes. As a consequence, the population is divided in subpopulations that group around the extremes.

We have extended the applicability of fitness sharing to tree-based genetic programming by defining a distance function for genetic programs that reflects the structural difference of trees [3]. We mean by structural difference of trees both the difference in the form of the trees and the difference in their node contents. Based on our distance function, we define here a new diversity measure for genetic programs. We expect to achieve better sampling of the search space by controlling the distance between individual programs. By using fitness sharing, we maintain the diversity of the population at a certain level depending on the size of program neighbourhood. In order to prevent premature convergence to some local optimum, we keep the population diverse in early stages of evolution. By adaptively changing the neighbourhood size, we can control diversity and convergence at the different stages of evolution.

The paper is organized as follows. In Section 2 we briefly present the possible distance measures for tree-based genetic programming. In Section 3 we give an overview of diversity measures for genetic programming and subsequently, in Section 4 we introduce the new diversity measure. We then describe in Section 5 the adaptive diversity maintenance. We show the experimental results in Section 6 and finally draw the conclusions.

2 Distance Measures for Genetic Programs

In genetic algorithms that represent the individuals as binary strings, a simple distance measure is the Hamming distance (the number of differing positions). In tree-based genetic programming it is more difficult to define a proper distance measure that can be computed efficiently. There are mainly two approaches: the *edit distance* and the distance based on structural comparison.

The *edit distance* between labeled trees was defined as the cost of shortest sequence of editing operations that transform one tree to the other [11, 14, 15]. An editing operation could be inserting or deleting a node or changing its label. In the case of ordered labeled trees T_1 and T_2, the time complexity of the algorithm for the simplified edit distance [14] is $O(|T_1| \times |T_2|)$, $|T|$ being the number of nodes in tree T. The edit distance has been used in pattern recognition [11], instance-based learning [2], and also genetic programming [6, 12]. However, as Banzhaf et al. [1] point out, the edit distance is not widely used in genetic programming because of its time complexity.

We defined a metric for genetic programs that reflects the structural difference of the genetic programs and can be computed in $O(|T_1| + |T_2|)$ [3]. The distance of two genetic programs is calculated in three steps:

1. The two genetic programs (trees) are completed with empty nodes, so that both trees have the same form.
2. For each pair of nodes at matching positions in the two trees, the difference of their contents is computed.
3. The differences computed in the previous step are combined in a weighted sum to form the distance of the two trees T_1 and T_2 with roots R_1 and R_2, respectively:

$$dist(T_1, T_2) = \begin{cases} d(R_1, R_2) & \text{if neither } T_1 \text{ nor } T_2 \text{ have any children,} \\ d(R_1, R_2) + \frac{1}{K}\sum_{i=1}^{m} dist(child_i(R_1), child_i(R_2)) & \text{otherwise,} \end{cases}$$

where $d(X, Y)$ is the difference of nodes X and Y, $child_i(X)$ is the ith child of node X and $K \geq 1$ is a constant, signifying that a difference at any depth r in the compared trees is K times more important than a difference at depth

$r + 1$. By choosing different values for K, we could define different shapes for the neighbourhoods.

We define the measure of diversity as the *typical distance* of two individual programs in a population (see Section 4).

Igel and Chellapilla [6] claim that the longer is the path from a node in a genetic tree to the root node, the less its influence is on the fitness. This can be a justification for the adequacy of our distance metric, since we give less importance to differences in less influential nodes.

3 Diversity Measures for Genetic Programs

A diversity measure on any collection of objects must reflect how different these objects are from each other.

Feldt [4] gives a comprehensive survey of diversity measures in genetic programming. Their complete presentation is beyond the scope of our paper, we discuss only the most relevant ones.

In genetic programming, a proper diversity measure reflects the structural difference of the genetic programs in a generation.

Koza [9] defines the *variety* of a population as "the percentage of individuals for which no exact duplicate exists elsewhere in the population". Langdon [10] defines variety as the number of unique individuals in a population. These definitions do not consider *how* different the individuals are. But variety is simple and has the useful property that if variety has a low value then any other measure of diversity will also have a low value. Keijzer [7] computes the variety of subtrees in a population.

Rosca [13] points out that a good diversity measure could be the percentage of structurally distinct programs. But he uses fitness and program size as measures of diversity. Since at any time there can be members of the population that have similar fitness, but very different structure, and similar size programs with very different structure, fitness and program size are not very reliable diversity measures.

Keller and Banzhaf [8] use an edit distance for determining the structural difference of the genetic trees. They propose a mapping of the space of genetic programs on a two dimensional distance space and then define the diversity of a population in this distance space. However, when applying the above mapping, so-called *collisions* can occur, i.e., several genetic programs could be mapped into the same position in the two dimensional distance space.

4 The Proposed Diversity Measure

The more uniformly the individuals of a population sample the search space, the more diverse the population is. Therefore, we base our diversity measure on the pairwise distances between programs in the population.

The most straightforward measure is *the mean value of the pairwise distances of individuals in a population*:

$$Diversity(P) = \frac{2}{N(N-1)} \sum_{1 \leq i < j \leq N} dist(gp_i, gp_j),$$

where N denotes the size of population P, and gp_i the genetic tree for individual i.

This diversity measure does not include niches, but it clearly indicates diversity: the larger is the mean value of the pairwise distances of individuals, the more distributed are the programs in the search space.

Another diversity measure that better reflects the *typical distance* of two individuals in a population is a weighted arithmetic mean. If there are several similar distance values, they will have a greater contribution to the sum than the isolated distance values. We first group the distances in several classes C_k according to their order of magnitude: $C_k = [d_k, d_{k-1})$ if $0 < k < l$, $C_0 = [d_0, +\infty)$, and $C_l = [0, d_l)$. We choose the values for d_k (i.e., the orders of magnitude) such that a distance of d_k between two trees reflects a difference at some node at depth k. A distance belongs to C_k if the two corresponding programs

- differ only in nodes at depths greater than $k-1$ and
- are different at least in one node at depth k (Programs differing only at depths greater than l are considered similar enough to be grouped together. In our experiments, we typically used $l = 5$).

Then the diversity can be defined as

$$Diversity(P) = \sum_{\substack{1 \leq i < j \leq N \\ dist(gp_i, gp_j) \in C_k}} w_k \times dist(gp_i, gp_j).$$

The distances in group C_k contribute to the sum with a weight proportional to their number:

$$w_k = \frac{2\, N_k}{N(N-1)},$$

where N_k is the number of elements in C_k and $\frac{N(N-1)}{2}$ is the total number of distances between pairs of individuals in the population.

Our diversity measure is independent of the technique used for maintaining population diversity and can be used for verifying the effects of any such technique.

5 Adaptive Maintenance of Diversity

By studying the effects of fitness sharing [5] on genetic programs (see Section 6), we have observed, as expected, that the objectives *diversity* and *fitness* are conflicting to some extent:

- with weaker diversity maintenance convergence was faster, but diversity also dropped faster; and
- with stronger diversity maintenance average fitness convergence was slower.

In other words, by *constantly* forcing a very diverse population it might be difficult to find a solution. Therefore, we propose keeping different levels of diversity at the different stages of evolution:

- at the beginning of evolution a higher diversity is required, since the initial random programs could be very far from any solution;
- toward the end of evolution convergence must be encouraged, so a lower level of diversity could be acceptable.

When applying fitness sharing, the level of diversity is regulated by the size of neighbourhood σ, higher diversity levels being obtained with larger neighbourhood sizes.

We propose an adaptive diversity maintenance that starts with a high diversity level (large σ) and modifies it regularly (every 5th generation) according to achievements (details of implementation are shown in Section 6):

if *loss_of_diversity*
    then *increase* σ
if *gain_in_fitness*
    then *decrease* σ

So, if a loss of diversity is encountered, by the increase of σ we expect to improve diversity. When fitness is improving, by the decrease of σ we expect to accelerate convergence. We expect that maintaining higher diversity at the beginning and lower diversity at the end of evolution will lead to better results.

6 Experimental Results on Symbolic Regression Problems

We conducted experiments on several symbolic regression problems. The test problems were randomly selected from the set of polynomials of degree up to 7 having real roots in the $[-1,1]$ interval. The goal was to evolve programs that approximate the selected functions in the $[-1,1]$ interval. For each problem, the training was performed on 50 randomly selected data points in the $[-1,1]$ interval. The resulted best programs were then tested on randomly selected data points that were different from the training data. The mean error on the test data served as the basis for comparing the accuracy of different methods. The parameter setting is shown in Table 1. Since the results for polynomials of the same degree are very close, we show here representative examples of polynomials of degree 2, 5, and 7, respectively.[1]

[1] The values of the polynomials are also in the $[-1,1]$ interval, but genetic programming always finds solutions better than the constant function $f(x)=0$. For example,

Table 1: The genetic programming parameter setting

Objective	Evolve a function that approximates the selected polynomial function in the $[-1, 1]$ interval
Terminal set	x, real numbers $\in [-1, 1]$
Function set	$+, *, /$
Fitness cases (Training data)	$N = 50$ randomly selected data points (x_i, y_i), #1 $y_i = x_i(x_i + 0.3)$ #2 $y_i = (x_i + 0.2)^2(x_i - 0.5)(x_i + 0.5)(x_i - 0.7)$ #3 $y_i = (x_i + 0.9)(x_i - 0.9)(x_i - 0.6)^2(x_i + 0.8)(x_i + 0.3)(x_i + 0.4)$
Test data	$K = 20$ sets of $N = 50$ randomly selected data points, each different from the training data
Raw fitness and also standardized fitness	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(gp(x_i) - y_i)^2}$, $gp(x_i)$ being the output of the genetic program for input x_i
Population size	100
Crossover probability	90%
Probability of Point Mutation	10%
Selection method	Tournament selection, size 10
Maximum number of generations	100
Initialization method	Ramped Half and Half
Diversity control	Every 5th generation

We found that, when fitness sharing was applied, the accuracy of the solution was better than in the case of original genetic programming. For instance, the error of the final solution (averaged over 50 runs with the same parameter setting) was 0.035 for original genetic programming and 0.017 for fitness sharing with $K = 10$ and niche radius $\sigma = 0.1$. For larger niche sizes, the error increases as an effect of fitness sharing with too many neighbours. But *the scope of fitness sharing is to maintain diversity in the population.* We show in Figure 1 the evolution of diversity according to our first diversity measure in the case of original genetic programming and fitness sharing using several settings.

Original genetic programming converges to populations of programs with very similar structure (diversity is mostly below 0.01 after generation 49), but not necessarily similar fitness. When applying fitness sharing, the populations (at any generation) contain individuals that are different from each other (for $\sigma = 0.5$, the diversity oscillates around 0.3 after generation 58). This difference depends on the predefined niche size.

The main characteristics of fitness sharing in genetic programming can be summarized as follows:

1. *The diversity of a population is an increasing function of niche size.* Practically, since few individuals are allowed in a niche, when the niche size is large, the evolved individuals are generally far from each other. When the

the error of function $f(x) = 0$ on problem # 2 is 0.106, whereas the error of a solution given by genetic programming is 0.009.

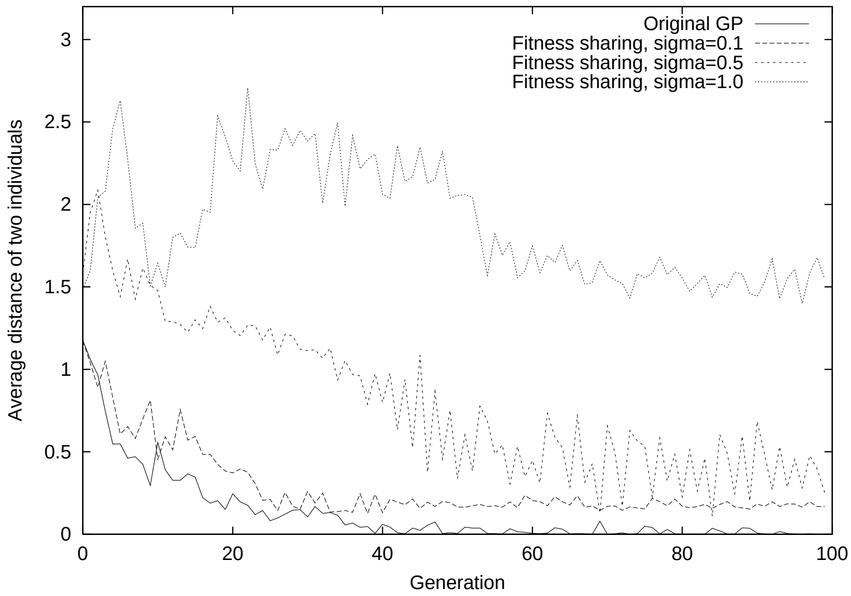

Fig. 1: Population diversity for original genetic programming and fitness sharing ($K = 10$) in the case of problem # 2

niche size is small, the evolved individuals are not very far from each other ($d \geq \sigma$).

2. When the niche size is very small, diversity is low and the population converges to very similar individuals, almost as if no fitness sharing were used (for $\sigma = 0.1$, diversity 0.17, best fitness 0.014, average fitness 0.042 at generation 55). When the niche size is large, and the individuals are generally far from each other, it is more difficult to obtain an accurate solution (for $\sigma = 1$, diversity always above 1.3, best fitness 0.06, average fitness 0.938 at generation 100). *A medium niche size allows at the same time the evolution of different programs and the convergence to accurate solutions (for $\sigma = 0.5$, diversity 0.977, best fitness 0.014, average fitness 0.682 at generation 42).*

Adaptive Diversity Maintenance

We designed the adaptive diversity maintenance on the basis of our findings with fitness sharing and fixed neighbourhood size.

Since diversity increases with neighbourhood size, the adaptive control of diversity maintenance is performed by modifying the neighbourhood size. We apply the diversity control step on a regular basis, every 5th generation. Modifying the neighbourhood size in every generation would produce too many fluctuations, it is better to allow each setting to stabilize and then perform the modification.

There is a *loss_of_diversity* if the diversity at time t is worse (less) than the diversity at time $t-1$, and there is a *gain_in_fitness* if the average fitness at time t is better (i.e., less) than the average fitness at time $t-1$.

We introduce the relative diversity and relative fitness for the consecutive diversity control moments $t-1$, t as:

$$\begin{aligned} rel\_diversity &= \tfrac{diversity(t)}{diversity(t-1)}, \\ rel\_fitness &= \tfrac{fitness(t)}{fitness(t-1)}. \end{aligned}$$

Then we implement the adaptive diversity control described in Section 5 as follows:

$$\begin{array}{l} \text{if } rel\_diversity < 0.1 \\ \qquad \text{then } \sigma = \frac{\sigma}{rel\_diversity} \\ \text{else } \sigma \text{ is unchanged} \end{array}$$

$$\begin{array}{l} \text{if } rel\_fitness < 1 \\ \qquad \text{then } \sigma = \sigma \times rel\_fitness \\ \text{else } \sigma \text{ is unchanged} \end{array}$$

Diversity may decrease if fitness improves, and increasing σ for every loss of diversity could lead to the cancellation of the effect related to the gain in fitness. Therefore, we allow a loss of diversity of one order of magnitude before increasing σ.

The evolution of diversity for original genetic programming, fitness sharing with constant σ and adaptive σ is shown in Figure 2. It can be seen that in the case of adaptive σ the diversity evolves as expected:

- in the initial phase, diversity is maintained at a high level, in order to find good starting points in the search space;
- in the middle phase, as fitness improves, diversity is allowed to lower; and
- in the final phase, a much lower diversity is allowed, since this is the time for convergence.

It is interesting to note that the results shown in Figure 2 are very similar to those shown in Figure 3.7 in [10]. There, the results of fitness sharing are compared to those of traditional genetic programming on the problem of evolving a list structure.

The adaptive method provides automatic adjustment of the niche size. In Table 2 we show the typical evolution of σ. Generally, the final stage of evolution begins around generation 50, when the niche size becomes very small.

7 Conclusion

We defined a new measure of population diversity based on the structural differences of programs. We first used the diversity measure for verifying the effects

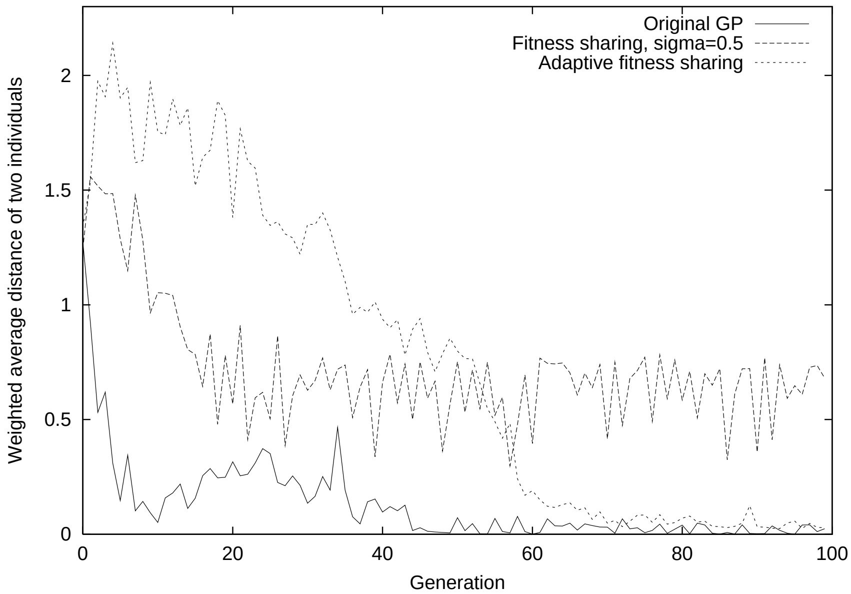

Fig. 2: Population diversity for original genetic programming and diversity maintenance in the case of problem # 2

Table 2: The adjustment of the niche size

Generation	σ
5	0.6106
20	0.1783
35	0.0635
40	0.0340
50	0.0094
60	0.0020
70	0.0013
80	0.0004

of fitness sharing [5]. Our experiments show that fitness sharing maintains population diversity in genetic programming at a level depending on niche size.

Since we could not know in advance, what is the appropriate niche size for a given problem, we proposed an adaptively changing niche size throughout evolution. This adaptation was driven by population diversity and fitness:

– when an alarming loss of diversity was encountered, the niche size was increased, and
– when an improvement of fitness was observed, the niche size was decreased.

In the future, we plan to study the effects of genetic operations on diversity inspired by the work of Langdon on variety [10].

We also plan to extend our diversity measure to other problems. For this, we need to adapt our distance function to the problem domain. In the case of problems that can be solved by using functions with the same arity, this can be done by simply defining the differences between the possible nodes. In the case of problem domains where functions with different arity can occur, we have to redesign the first step of distance computation.

References

1. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann, 1998.
2. U. Bohnebeck, T. Horváth, and S. Wrobel, Term comparisons in first-order similarity measures, in *Proceedings of the 8th International Workshop on Inductive Logic Programming*, ed., D. Page, volume 1446 of *LNAI*, pp. 65–79. Springer-Verlag, (1998).
3. A. Ekárt and S. Z. Németh, A metric for genetic programs and fitness sharing, in *Proceedings of EUROGP'2000*, eds., R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. Fogarty, volume 1802 of *LNCS*, pp. 259–270. Springer-Verlag, (2000).
4. R. Feldt, Using genetic programming to systematically force software diversity, Tech. rep. nr. 296l, Chalmers University of Technology, (1998).
5. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
6. C. Igel and K. Chellapilla, Investigating the influence of depth and degree of genotypic change on fitness in genetic programming, in *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, eds., W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, pp. 1061–1068, (1999).
7. Maarten Keijzer, Efficiently representing populations in genetic programming, in *Advances in Genetic Programming 2*, eds., Peter J. Angeline and Kenneth E. Kinnear, 259–278, MIT Press, (1996).
8. R. E. Keller and W. Banzhaf, Explicit maintenance of genetic diversity on genospaces, Technical report, Dortmund University, (1994).
9. John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
10. William B. Langdon, *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, Kluwer Academic, 1998.
11. S.-Y. Lu, The tree-to-tree distance and its application to cluster analysis, *IEEE Transactions on PAMI*, **1**(2), 219–224, (1979).
12. Una-May O'Reilly, Using a distance metric on genetic programs to understand genetic operators, in *Late Breaking Papers at the 1997 Genetic Programming Conference*, ed., John R. Koza, pp. 199–206, (1997).
13. J. P. Rosca, Genetic programming exploratory power and the discovery of functions, in *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pp. 719–736, (1995).
14. S. M. Selkow, The tree-to-tree editing problem, *Information Processing Letters*, **6**(6), 184–186, (1977).
15. K.-C. Tai, The tree-to-tree correction problem, *Journal of the ACM*, **26**(3), 422–433, (1979).