A Metric for Genetic Programs and Fitness Sharing

Anikó Ekárt and S. Z. Németh

Computer and Automation Research Institute
Hungarian Academy of Sciences
1518 Budapest, POB. 63, Hungary
E-mail: ekart@sztaki.hu, snemeth@oplab.sztaki.hu

Abstract. In the paper a metric for genetic programs is constructed. This metric reflects the structural difference of the genetic programs. It is used then for applying fitness sharing to genetic programs, in analogy with fitness sharing applied to genetic algorithms. The experimental results for several parameter settings are discussed. We observe that by applying fitness sharing the code growth of genetic programs could be limited.

1 Introduction

The search space of a search problem may contain more than one peak. In artificial genetic search, the population may converge to one of these peaks. In genetic algorithms there are several so-called *niching* techniques that prevent the convergence of the whole population to just one of the peaks [5, 4]. Niching proceeds by penalizing individuals that are "close" to other individuals in the population. It is based on the fact that the search space of genetic algorithms is a metric space (the individuals are represented as real-valued or boolean vectors).

Fitness sharing [4, 5] was introduced as a technique for maintaining population diversity in genetic algorithms. The population is distributed over the different peaks in the search space: in the neighborhood of each peak, a number of individuals proportionate to the height of that peak are allowed.

In this work, we extend 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. As a result of fitness sharing, we obtain smaller genetic programs as solutions.

Code growth is a general problem of genetic programming systems [8, 14] and several methods for limiting code growth have been proposed:

- introducing a maximum allowed depth [8];
- editing out the irrelevant parts [8]; and
- including program size in the fitness measure [7, 17, 14].

We show here that fitness sharing based on our metric induces a preference for smaller programs, and it could be used for limiting code growth in genetic programming systems.

2 State of the Art

For genetic algorithms several niching techniques have been developed [10]. *Crowding* is a restricted replacement method, where the offspring of crossover and mutation replaces the most similar individual from a sample of the population. Another approach is the *restricted competition* among dissimilar individuals during selection. Both algorithms restrict their selection method, but in a different way.

Fitness sharing [5, 4] treats fitness as a shared resource of the population, and thus requires that similar individuals share their fitness. The fitness of each individual f_i is worsened by its niche count m_i. The niche count estimates crowding in the neighborhood of individual i and is calculated over the individuals in the current population:

$$m_i = \sum_{j=1}^{N} S(d(i,j)),$$

N being the population size, $d(i, j)$ the distance of individuals i and j, and S the sharing function. S is a decreasing function, since sharing with a close individual should be greater than sharing with some farther individual. The typical sharing function has the form

$$S(d) = \begin{cases} 1 - (d/\sigma)^{\alpha} & \text{if } d \leq \sigma, \\ 0 & \text{if } d > \sigma. \end{cases}$$

The niche radius σ is fixed by the user at some minimum distance between peaks, and usually $\alpha = 1$. By using fitness sharing, the population does not converge as a whole, but convergence takes place within the niches.

The main drawback of fitness sharing is that the computation of the shared fitness for the entire population in each generation could be very time-consuming.

By using tournament selection, it is possible to overcome this difficulty. Oei, Goldberg and Chang [12] propose a scheme for combining sharing with binary tournament selection. They calculate the shared fitness by continuously updating the fitness as the new population is filled.

Yin and Germay [16] propose the application of a clustering algorithm before sharing. Thus, the population is first divided into niches, and then the shared fitness of any individual is computed only with respect to the individuals that are in its niche.

Horn, Nafpliotis and Goldberg [6] propose a tournament selection scheme, where two individuals are picked randomly from the population, a comparison set is also picked randomly from the population and the two individuals are compared against the individuals from the comparison set. They implement sharing as counting the individuals from the comparison set that are in the niche of the two competing individuals. The individual with the smallest niche count is the best candidate.

The other difficulty of fitness sharing consists in determining the niche radius.

Beyond these difficulties, one has to make the appropriate choice of a genotypic or a phenotypic distance measure for the given problem.

In the case of tree-based genetic programming, a straightforward phenotypic distance measure is the Euclidean distance between the values of the two genetic programs on the fitness cases.

A genotypic distance measure should be a distance function on trees. Several distance-functions on trees have been proposed in different domains of artificial intelligence, such as inductive logic programming, bio-informatics, pattern recognition.

The *edit distance* between labeled trees was defined as the cost of shortest sequence of editing operations that transform one tree to the other [13, 15, 9]. An edit operation consists in inserting or deleting one node (subtree in [13]) or changing the label of one node. The cost of each edit operation could be specified separately [15] or the cost is unique for insertions, deletions, and changes, respectively [9]. The algorithms for computing the edit distance of trees are very time-consuming. In the case of ordered labeled trees (where the order of children of a node is significant) T_1 and T_2, the time complexity of the algorithm for the simplified edit distance [13] is $O(|T_1| * |T_2|)$, $|T|$ being the number of nodes in tree T. In the case of unordered labeled trees, the time complexity is exponential [18]. However, the edit distance has been used in pattern recognition [9] and instance-based learning [2]. But because of time complexity, the edit distance is not widely used in genetic programming [1].

Nienhuys-Cheng [11] defines a metric bounded by 1 that takes into consideration the depth of nodes in the trees:

$$d(p(s_1, s_2, \ldots, s_n), q(t_1, t_2, \ldots, t_m)) = \begin{cases} 1 & \text{if } p \neq q, \\ \frac{1}{2n} \sum_{i=1}^{n} d(s_i, t_i) & \text{if } p = q, \end{cases} \quad (1)$$

p and q being the root labels of the two trees. This metric is used then in inductive logic programming for measuring the quality of approximations to a correct program . This distance can be computed in $O(min(|T_1|, |T_2|))$.

3 The Proposed Metric

Our goal was to define a proper metric for genetic programs that reflects the structural difference of the genetic programs and can be computed efficiently.

We show the experimental results for the metric in the case of a symbolic regression problem.

The distance of two genetic programs is calculated in three steps:

1. The two genetic programs (trees) to be compared are brought to the same tree-structure (only the contents of the nodes remain different).
2. The distances between any two symbols situated at the same position in the two trees are computed.
3. The distances computed in the previous step are combined in a weighted sum to form the distance of the two trees.

We construct the distance function step by step, as follows.

We make the convention that before comparing two trees of different structure, we complete them by NULLs so that the resulting trees have the same structure, as it is shown in Figure 1. In the case of commutative operations (unordered trees) we do not know in advance which pairing of subtrees is better. Trying out each possible pairing makes the distance computation exponential in tree depth. When the trees have different arities, the missing subtree(s) could also be completed by NULLs. In this case, it would be difficult to choose where to insert the missing subtree. Therefore, in all of our experiments we use binary trees and in most of them we consider ordered trees.

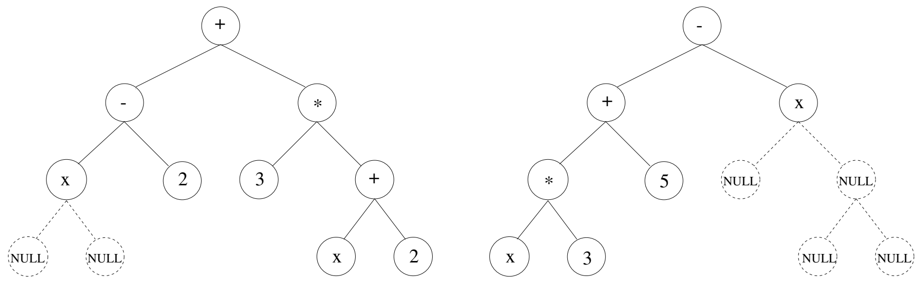

Fig. 1. The trees are completed by NULLs to have the same layout.

The trees that can be constructed by genetic programming contain functions and terminals (variables and constants). The functions may be grouped in several groups according to their similarity (E.g. + and -, * and / for trees encoding arithmetic expressions). The terminal set is initially differentiated into variables and constants, but one could further partition the variable set.

We partition the set of symbols into n sets $A_0, A_1, ..., A_n$; A_0 being the set containing NULL and A_1 the set containing the constants. We define the set A_0 for the single element NULL in order to treat the comparison between some node and NULL in the same way as a comparison between two elements of different sets.

Let $\delta : \{0, 1, \ldots, n\} \times \{0, 1, \ldots, n\} \to (0, +\infty)$ be a function with properties:

$$\delta(i, j) \leq \delta(i, k) + \delta(k, j), \tag{2}$$
$$\delta(i, j) = \delta(j, i), \tag{3}$$
$$\delta(i, i) < \delta(j, k), \;\; j \neq k, \tag{4}$$

for all $i, j, k \in \{0, 1, \ldots, n\}$.

We define the distance between two different elements $x \in A_i$ and $y \in A_j$ as follows:

$$d(x, y) = \begin{cases} 0 & \text{if } x = y, \\ C * \frac{|x-y|}{\underset{v,w \in A_1}{Max} |v-w|} & \text{if } i = j = 1, \\ \delta(i, j) & \text{otherwise,} \end{cases}$$

C being a constant. By choosing $C < \delta(i, j), \forall i, j \in \{0, 1, \ldots, n\}$, the distance of two constants becomes less than the distance of two other elements. With this distance definition, we provide that the distance between two elements of the same set is less than the distance between two elements of different sets.

Now, after defining the distance between elements, we can define the distance between trees. We take into consideration the fact that a difference at some node closer to the root could be more significant than a difference at some node farther from the root (in a similar way to [11]). First we complete the trees by NULLs, so that both trees have the same structure. Let $T_1 = p(s_1, s_2, \ldots, s_m)$ be the tree with root p and subtrees $s_1, s_2, \ldots, s_m$, and $T_2 = q(t_1, t_2, \ldots, t_n)$ the tree with root q and subtrees $t_1, t_2, \ldots, t_m$, respectively. Then, we calculate the distance between trees T_1 and T_2 as:

$$dist(T_1, T_2) = \begin{cases} d(p, q) & \text{if neither } T_1 \text{ nor } T_2 \text{ have any children,} \\ d(p, q) + \frac{1}{K} * \sum_{l=1}^{m} dist(s_l, t_l) & \text{otherwise,} \end{cases} \tag{5}$$

where $K \geq 1$ 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 neighborhoods (see Section 4).

It can be easily proved that all conditions for $dist$ to be a **metric** [3] are met:

$$dist(T_1, T_2) \geq 0,$$
$$dist(T_1, T_2) = 0 \Leftrightarrow T_1 = T_2,$$
$$dist(T_1, T_2) = dist(T_2, T_1) \text{ (symmetry),}$$
$$dist(T_1, T_2) \leq dist(T_1, T_3) + dist(T_3, T_2) \text{ (triangle inequality).}$$

Our metric can be seen as a generalization of metric (1):

- instead of fixing the distance of any two different symbols at 1, we partition the set of symbols into subsets of "similar" symbols and we set the values of distances on the basis of this partitioning;
- we introduce constant K in (5), and thus we can experiment with different shapes of niches.

If $K > m$, our $dist$ is bounded by

$$M = \frac{\underset{i,j \in \{0,1,\ldots,n\}}{Max} \delta(i,j)}{1 - m/K}$$

(generalization of Theorem 5 of [11]). Then, if two trees are the same to at least depth r,

$$dist(T_1, T_2) \leq \frac{m * M}{K^{r+1}}$$

(generalization of Lemma 6 of [11]). If two trees are not the same to depth r,

$$dist(T_1, T_2) \geq \frac{\underset{i,j \in \{0,1,\ldots,n\}}{Min} \delta(i,j)}{K^r}$$

(generalization of Lemma 7 of [11]).

4 Empirical Validation

We used the metric in a symbolic regression problem. We applied fitness sharing with different parameter settings and compared the results. The goal was to evolve the programs that approximate the function $f(x) = 1.5 + 24.3x^2 - 15x^3 + 3.2x^4$ in 100 data points that were randomly selected in the $[0, 1]$ interval. The general parameter setting is shown in Table 1. The presented results are the average of 50 runs in each parameter setting.

We define the distance function for two genetic programs in the case of symbolic regression of a function of one variable, with function set $F = \{+, -, *, /\}$. All of the functions take two arguments, thus the resulting trees are binary. We mainly consider ordered trees. We partition the set of symbols as follows:

$$A_0 = \{NULL\},$$
$$A_1 = \{\text{constants in the range -100 \ldots 100}\},$$
$$A_2 = F,$$
$$A_3 = \{x\}.$$

Table 1. The genetic programming parameter setting

Objective	Evolve a function that fits the data points of the fitness cases
Terminal set	x, real numbers $\in [-100, 100]$
Function set	$+, *, /$
Fitness cases	$N = 100$ randomly selected data points (x_i, y_i), $y_i = 1.5 + 24.3x_i^2 - 15x_i^3 + 3.2x_i^4$
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 (as in [6])
Termination criterion	none
Maximum number of generations	200
Maximum depth of tree after crossover	20
Initialization method	Grow

Then we choose function δ, such that it satisfies requirements (2),(3),(4), for example

$$\begin{aligned}
\delta(0,i) &= \delta(i,0) = 5, \quad i \in \{1,2,3\}, \\
\delta(1,2) &= \delta(2,1) = 2, \\
\delta(2,2) &= 1, \\
\delta(3,i) &= \delta(i,3) = 3, \quad i \in \{1,2\},
\end{aligned}$$

where the other values of the function need not be specified.

4.1 Application of the Metric for Fitness Sharing

We experimented fitness sharing with $K = 1$, $K = 2$ and $K = 10$ in the expression of the distance function of (5). We set the distance of two symbols $d(x, y) \in [0, 5]$, so that the distance of two nodes at the same position in the trees counts for the final distance with a value depending on this position, as it is shown in Table 2.

Thus, if the distance of two trees is a value less than 0.005 in the case of $K = 10$, they only differ at some node at depth 3(or more) or they differ just at the values of constants.

Table 2. The difference of two nodes counted for the distance of the trees containing them

Position(depth)	Range of added value		
	K=10	K=2	K=1
0 (root)	0-5	0-5	0-5
1	0-0.5	0-2.5	0-5
2	0-0.05	0-1.25	0-5
3	0-0.005	0-0.625	0-5

Table 3. The ranges for niche size

K	Niche size
10	$[5 * 10^{-4}, 0.5]$
2	$[5 * 10^{-3}, 0.5]$
1	$[1, 10]$

We experimented with niche sizes corresponding to the values of K, as shown in Table 3. By selecting different values for K, we can experiment different niche shapes. For $K = 1$, for a certain tree, any other tree that differs from it at just one internal node, is in its neighborhood of size $\sigma \geq 1$. A niche size $\sigma < 1$ in this case would restrict only to differences in the values of constants. In the mean time, for $K = 10$, if $\sigma = 1$, for a certain tree, any other tree with the same root is in its neighborhood. But for $K = 10$ and $\sigma = 5 * 10^{-3}$, the neighborhood of a tree contains only the trees with differences at depth 3 or more (and slight differences in the values of constants at depth 1 or 2).

We experimented with our metric (1) on trees with $+$ and $*$ counted as commutative and (2) on ordered trees. In the first case, the final solution of fitness sharing was slightly more accurate than the solution of original genetic programming. Since the computational cost was very high (the distance computation was exponential in tree depth) and there was little improvement in fitness, we decided to conduct the rest of the experiments on ordered trees. In the case of ordered trees, the final solution of fitness sharing was slightly worse than the solution of original genetic programming. The error of the final solution was 0.42 (averaged over 50 runs) for simple genetic programming and 0.45 for fitness sharing with $K = 10$ and niche radius $\sigma = 0.05$. For bigger niche radii, the error increases as an effect of fitness sharing with too many neighbors (E.g., for $K = 10$ and $\sigma = 0.5$, sharing is performed among individuals that can differ at one node at the first depth or any number of nodes at depth more than 1). But the scope of fitness sharing is to maintain diversity in the population.

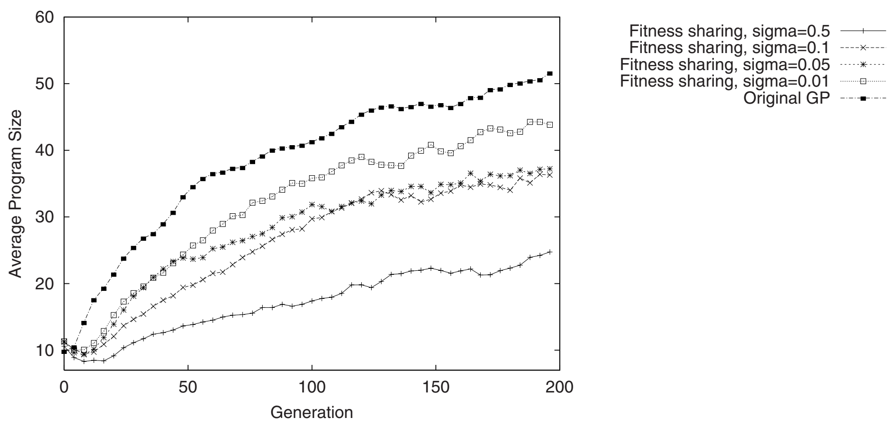

Fig. 2. The average program size for $K = 10$.

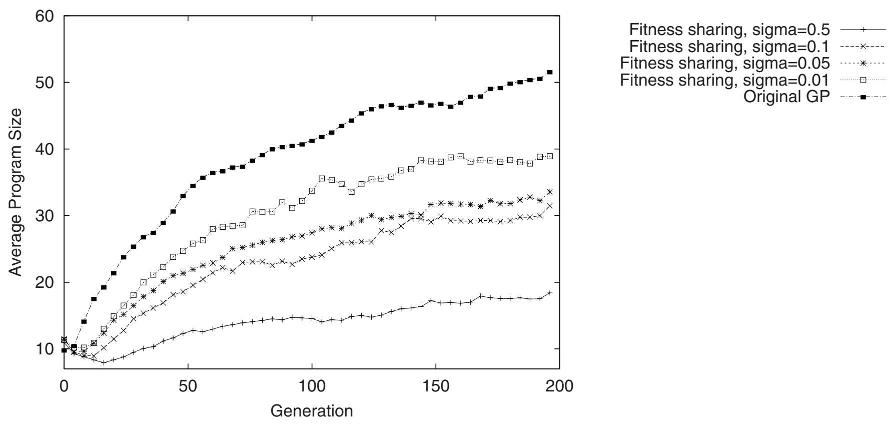

Fig. 3. The average program size for $K = 2$.

If we look at the average program size (Figures 2, 3, 4), we can immediately notice, that the size of programs in the case of fitness sharing does not grow as fast as the size of programs in the case of original genetic programming. There is a niche size depending on K ($K = 10 \Rightarrow \sigma \in [0.05, 0.1]$, $K = 2 \Rightarrow \sigma \in [0.01, 0.1]$, $K = 1 \Rightarrow \sigma \in [1, 2]$), for which code growth is much slower and fitness is not significantly worsened.

In order to see whether the difference in program size between original genetic programming and fitness sharing is significant, we computed the 99% confidence interval for this difference over generations. The evolution of two confidence intervals for $K = 10$ is shown in Figure 5. Since at generation 200 the difference is in the range $[8.18, 20.95]$ with probability 99% when $\sigma = 0.05$, we can conclude that the difference is significant.

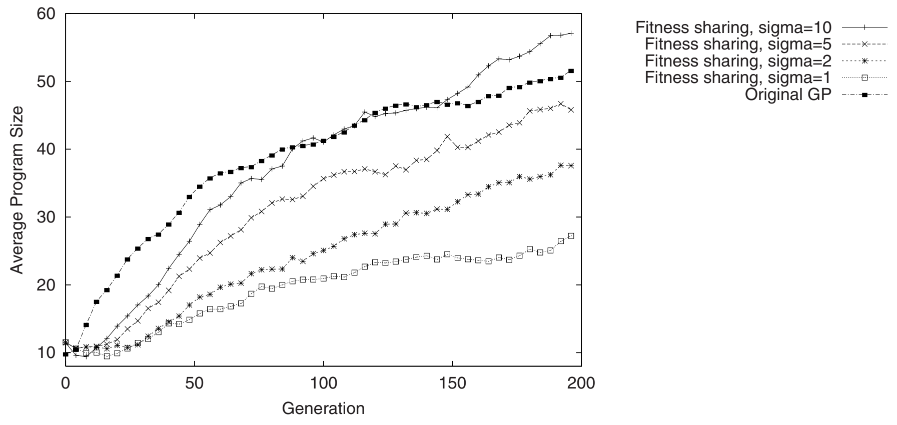

Fig. 4. The average program size for $K = 1$.

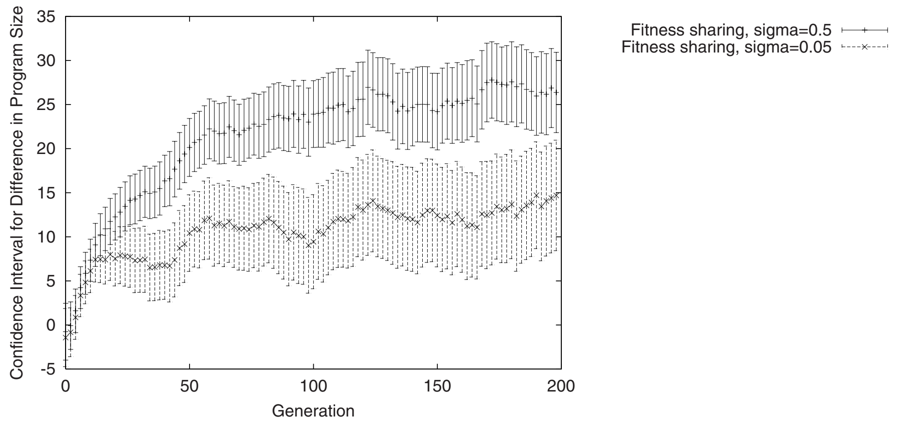

Fig. 5. The 99% confidence interval of the difference in program size between original genetic programming and fitness sharing.

The moderated code growth is explained by the following:

1. By the application of fitness sharing, fitness is shared among not highly fit individuals containing the same nonfunctional code. Their shared fitness is even worse, and they have less chance of being selected.
2. By using tournament selection and fitness sharing (see [6]), if two competing individuals are of similar (unshared) fitness, the smaller one is selected. The individual with less neighbors or farther ones is preferred to the individual with more or closer neighbors. In other words, the individual that has less common parts with the individuals from the comparison set is se-

lected. *But a smaller individual has less common parts with other random individuals, than a bigger one.*

3. The probability of selecting a crossover point closer to the root is higher in small trees than in big trees. A change near the root moves the offspring farther from its parents. Thus, *the offspring of smaller parents are more likely to find new niches.*

4.2 Comparison to Other Metrics

We also conducted experiments with two other metrics: the phenotypic distance and the edit distance [13]. In the case of phenotypic sharing, the results were very similar to original genetic programming. In the case of editing distance, the accuracy of best program was slightly better than in original genetic programming, but the computational cost was high.

5 Conclusions

In the present paper, we introduced a metric for genetic programs that reflects their tree-structure and can be efficiently computed.

We used the metric for applying fitness sharing to genetic programming. By the application of fitness sharing, we could obtain smaller genetic programs of similar (unshared) fitness. Fitness sharing is usually applied for maintaining population diversity. Thus, for interpreting its result, a proper measure of population diversity in genetic programming is needed [1]. Our metric could be used for defining such a measure of population diversity, since it is based on the structural difference of the programs.

Our fitness sharing did not improve accuracy, but the existing literature on the beneficial effects of fitness sharing in *genetic algorithms* suggests that further work might give results in this direction in *genetic programming*, too.

Acknowledgements

This work has been supported by grants No. T023305, T029572 and T25471 of the National Research Foundation of Hungary. The second author was supported by the Bolyai János research fellowship. The authors are grateful to A. Márkus and J. Váncza for many helpful discussions.

References

1. Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann, 1998.

2. Uta Bohnebeck, Tamás Horváth, and Stefan 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. J. R. Giles, *Introduction to the Analysis of Metric Spaces*, Australian Mathematical Society Lecture Series, 1987.
4. David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
5. John H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor: The University of Michigan Press, 1975.
6. Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg, ‘A niched Pareto genetic algorithm for multiobjective optimization’, in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 82–87, (1994).
7. Hitoshi Iba, Hugo de Garis, and Taisuke Sato, ‘Genetic programming using a minimum description length principle’, in *Advances in Genetic Programming*, ed., Kenneth E. Kinnear, 265–284, MIT Press, (1994).
8. John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
9. Shin-Yee Lu, ‘The tree-to-tree distance and its application to cluster analysis’, *IEEE Transactions on PAMI*, **1**(2), 219–224, (1979).
10. Samir Mahfoud, ‘Niching methods for genetic algorithms’, Illigal report 95001, University of Illinois at Urbana-Champaign, (1995).
11. Shan-Hwei Nienhuys-Cheng, ‘Distance between Herbrand interpretations: a measure for approximations to a target concept’, in *Proceedings of the 7th International Workshop on Inductive Logic Programming*, eds., N. Lavraĉ and S. Dẑeroski, volume 1297 of *LNAI*, pp. 213–226. Springer-Verlag, (1997).
12. C. K. Oei, David E. Goldberg, and S. J. Chang, ‘Tournament selection, niching and the preservation of diversity’, Illigal report 91011, University of Illinois at Urbana-Champaign, (1991).
13. Stanley M. Selkow, ‘The tree-to-tree editing problem’, *Information Processing Letters*, **6**(6), 184–186, (1977).
14. Terence Soule, James A. Foster, and John Dickinson, ‘Code growth in genetic programming’, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, eds., John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, pp. 215–223, (1996).
15. Kuo-Chung Tai, ‘The tree-to-tree correction problem’, *Journal of the ACM*, **26**(3), 422–433, (1979).
16. Xiaodong Yin and Noël Germay, ‘A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization’, in *Artificial Neural Nets and Genetic Algorithms*, eds., Rudolf F. Albrecht, Nigel C. Steele, and Colin R. Reeves, pp. 450–457, (1993).
17. Byoung-Tak Zhang and Heinz Mühlenbein, ‘Balancing accuracy and parsimony in genetic programming’, *Evolutionary Computation*, **3(1)**, 17–38, (1995).
18. Kaizhong Zhang, Rick Statman, and Dennis Shasha, ‘On the editing distance between unordered labeled trees’, *Information Processing Letters*, **42**, 133–139, (1992).