# MDL-Based Fitness Functions for Learning Parsimonious Programs

**Byoung-Tak Zhang*** **Heinz Mühlenbein**
German National Research Center for Computer Science (GMD)
Schloss Birlinghoven, D-53754 St. Augustin, Germany
E-mail: {zhang, muehlenbein}@gmd.de

## Introduction

Genetic programming starts with an initial population of computer programs composed of elementary functions and terminals (Koza 1992; Koza 1994; Kinnear 1994). Genetic operators, such as crossover and selection, are used to adapt the shape and size of the programs and evolve increasingly fit populations. This process can be viewed as a search for a highly fit computer program, $A_{best}$, in the whole program space $\mathcal{A} = \{A_1, A_2, ...\}$. The quality of each computer program is measured by running it over a training set $D$ of input-output cases of the unknown process $\tilde{f}$: $D = \{(\mathbf{x}_c, \mathbf{y}_c)\}_{c=1}^N$, where $\mathbf{x}_c \in X$, $\mathbf{y}_c \in Y$ and $\mathbf{y}_c = \tilde{f}(\mathbf{x}_c)$. The domain $X$ and the range $Y$ are defined by the application. The goodness of the program, $A$, is usually measured in terms of the error: $\sum_{c=1}^N (\mathbf{y}_c - f_A(\mathbf{x}_c))^2$, where $f_A$ is the function realized by $A$.

Though this training accuracy can be used as a single measure for fitness, many empirical studies have shown that, as programs grow, it also become less and less likely for them to be general (Kinnear 1993; Tackett 1993). In addition, large structures require more computer resources in space and time for the evolution. Several methods have been proposed to take into consideration of structural complexity in fitness evaluation, but relatively few attempts have been made in the genetic programming community to employ the complexity penalty in a more principled way.

In this paper we use a Bayesian model-comparison method to develop a framework in which a class of fitness measures is introduced for dealing with problems of parsimony based on the minimum description length (MDL) principle (Rissanen 1986). We then describe an adaptive technique for putting this fitness function into practice. It automatically balances the ratio of training accuracy to solution complexity without losing the population diversity needed to achieve a desired training accuracy. The effectiveness of the method is shown in the context of evolving neural networks based on noisy training data. We also discuss the relationship of this work with other MDL based approaches to tree induction.

*Address after August 31, 1995: Department of Computer Science, Kon-Kuk University, Seoul 133-701, Korea. E-mail: btzhang@galaxy.konkuk.ac.kr

## Deriving MDL-Based Fitness Functions

As outlined in the introduction, the goal of genetic programming can be formulated as finding a program or model, $A$, whose evaluation $f_A$ best approximates the underlying relation $\tilde{f}$, where the approximation quality is measured by

$$E(D|A) = \frac{1}{N}\sum_{c=1}^{N}(\mathbf{y}_c - f_A(\mathbf{x}_c))^2. \tag{1}$$

Considering the program as a Gaussian model of the data, the likelihood of the training data is described by

$$P(D|A) = \frac{1}{Z(\beta)}\exp(-\beta E(D|A)), \tag{2}$$

where $Z(\beta)$ is a normalizing constant, and $\beta$ is a positive constant determining the sensitivity of the probability to the error value.

Bayes' rule states that the posterior probability of a model is:

$$P(A|D) = \frac{P(D|A)P(A)}{P(D)} \tag{3}$$

where $P(A)$ is the prior probability of the models and

$$P(D) = \int P(D|A)P(A)dA. \tag{4}$$

The most plausible model given the data is then inferred by comparing the posterior probabilities of all models. Since $P(D)$ is the same for all models, for the purposes of model comparison, we need only compute

$$P(D|A)P(A). \tag{5}$$

A complex model with many parameters will have a broad distribution of priors, i.e. a small $P(A)$ value, and hence a small $P(A|D)$ value. A simpler, more constrained model will have a sharper prior and thus a large $P(A|D)$ value. For the more complex model to be favored over the simpler one, it must achieve a much better fit to the data. Thus Bayesian model-comparison techniques choose between alternative models by trading off this measure of the simplicity of a model against the data misfit. Thus it is reasonable to define the evolutionary process of genetic programming as the maximization of the posterior probability:

$$\begin{aligned} A_{best} &= \arg\max_{A_i \in \mathcal{A}} \{P(A_i|D)\} \\ &= \arg\max_{A_i \in \mathcal{A}} \{P(D|A_i)P(A_i)\}. \end{aligned} \tag{6}$$

Though the Bayesian inference is very useful in theory, it is not very convenient to deal with in practice. Alternatively, we can use the model complexity; according to coding theory (Rissanen 1984), if $P(\mathbf{x})$ is given, then its code length is given as $L(P(\mathbf{x})) = -\log(P(\mathbf{x}))$. Maximizing $P(D|A)P(A)$ is thus equivalent to minimizing the total code length:

$$\begin{aligned} L(A|D) &= L(P(D|A)P(A)) \\ &= -\log(P(D|A)P(A)) \\ &= L(D|A) + L(A), \end{aligned} \tag{7}$$

where $L(D|A) = -\log P(D|A)$ and $L(A) = -\log P(A)$. Here $L(D|A)$ is the code length of the data when encoded using the model $A$ as a predictor for the data $D$, and $L(A)$ is the length of the model itself. This leads to the minimum description length (MDL) principle (Rissanen 1986; Fogel 1991) where the goal is to obtain accurate and parsimonious estimates of the probability distribution. The idea is to estimate the simplest density that has high likelihood by minimizing the total length of the description of the data:

$$\begin{aligned} A_{best} &= \arg\min_{A_i \in \mathcal{A}} \{L(A_i|D)\} \\ &= \arg\min_{A_i \in \mathcal{A}} \{L(D|A_i) + L(A_i)\}. \end{aligned} \tag{8}$$

Minimum complexity estimators are treated in this general form that can be specialized to various cases. The specialization can be done by choosing a set of candidate probability distributions and by choosing a description length for each of these distributions, subject to information-theoretic requirements. If we assume that the squared errors for the data points are independent and normally distributed about a zero mean, then the density function is

$$\begin{aligned} P(D) &= \prod_{i,c} P_i^c \\ &= \prod_{i,c} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{r_i^c}{2\sigma^2}\right), \end{aligned} \tag{9}$$

where $r_i^c$ is the $i$th component of the squared error for the $c$th example, and $\sigma^2$ is the variance of the Gaussian distribution. The cost of coding using this distribution can be computed from the optimal coding theorem.

As illustrated above, an implementation of MDL typically necessitates knowing the true underlying probability distribution or an approximation of it. In general, however, the distribution of underlying data structure is unknown and the exact formula for the fitness function is impossible to obtain. The key point is that both the Bayesian model comparison and MDL principle reduced to the general criterion consisting of accuracy and parsimony (or training error and complexity) of models that should be balanced. We propose to measure the fitness of a program $A$ given a training set $D$ in its most general form as

$$\begin{aligned} F(A|D) &= F_D + F_A \\ &= \beta E(D|A) + \alpha C(A), \end{aligned} \tag{10}$$

where the parameters $\alpha$ and $\beta$ control the trade-off between complexity $C(A)$ and fitting error $E(D|A)$ of the program. In this framework, genetic programming is considered as a search for a program that minimizes $F(A|D)$, or

$$\begin{aligned} A_{best} &= \arg\min_{A_i \in \mathcal{A}} \{F(A_i|D)\} \\ &= \arg\min_{A_i \in \mathcal{A}} \{\beta E(D|A_i) + \alpha C(A_i)\}. \end{aligned} \tag{11}$$

The following section describes a general adaptive technique that balances $\alpha$ and $\beta$ in unknown environments.

## The Fitness Function with an Adaptive Occam Factor

Our basic approach is to fix the error factor at each generation and change the complexity factor adaptively with respect to the error. Let $E_i(g)$ and $C_i(g)$ denote the error and complexity of $i$th individual at generation $g$. For simplicity, we assume $0 \le E_i(g) \le 1$ and $C_i(g) > 0$. Given this, we propose to define the fitness of an individual $i$ at generation $g$ as follows:

$$F_i(g) = E_i(g) + \alpha(g)C_i(g). \tag{12}$$

Here $\alpha(g)$ is called the adaptive Occam factor and expressed as

$$\alpha(g) = \begin{cases} \frac{1}{N^2} \frac{E_{best}(g-1)}{\hat{C}_{best}(g)} & \text{if } E_{best}(g-1) > \epsilon \\ \frac{1}{N^2} \frac{1}{E_{best}(g-1) \cdot \hat{C}_{best}(g)} & \text{otherwise,} \end{cases} \tag{13}$$

where $N$ is the size of training set. User-defined constant $\epsilon$ specifies the maximum training error allowed for the final solution.

Note that $\alpha(g)$ depends on $E_{best}(g-1)$ and $\hat{C}_{best}(g)$. $E_{best}(g-1)$ is the error value of the program which had the smallest (best) fitness value at generation $g-1$. $\hat{C}_{best}(g)$ is the size of the best program at generation $g$ estimated at generation $g-1$ (see (Zhang & Mühlenbein 1995) for more details). $\hat{C}_{best}(g)$ is used for the normalization of the complexity factor. In essense, two adaptation phases are distinguished:

- When $E_{best}(g-1) > \epsilon$, $\alpha(g)$ decreases as the training error falls since $E_{best}(g-1) \leq 1$ is multiplied. This encourages fast error reduction at the early stages of evolution.
- For $E_{best}(g-1) \leq \epsilon$, in contrast, as $E_{best}(g)$ approaches 0 the relative importance of complexity increases due to the division by a small value $E_{best}(g-1) \ll 1$. This encourages stronger complexity reduction at the final stages to obtain parsimonious solutions.

Note also that the equation (13) is a realization of the general form derived from the MDL approach (10) where $\beta$ is fixed and $\alpha$ is expressed as a function of $g$: $\beta = 1.0$ and $\alpha = \alpha(g)$.

The effectiveness of the adaptive Occam method was studied by comparing its performance with that of the baseline fitness function $F_i(g) = E_i(g) = E(D_N|A_i)$ in evolving sigma-pi neural networks for solving parity problems. The complexity of the program was measured as a linear sum of the number of weights, units, and layers of the network. Both methods used the same data sets of 7-parity. The training set consisted of 64 examples with 5% noise. The test set contained $2^7 = 128$ clean data. The population size was $M = 40$. For each method, 20 runs were executed to observe the complexity of the best solution and its training and generalization performance at generation $g_{max} = 100$.

The bar graphs in Figure 1 compare the average network size in terms of the number of units and layers. The corresponding learning and generalization performance of both methods are also compared in Figure 2. The results show that applying the adaptive Occam method achieves significantly better generalization performance. Whereas the solution size in the baseline method increased without bound, controlling the Occam factor as described in the last section could prune inessential substructures to get parsimonious solutions but without losing the training performance. It is interesting to note that the evolution with the Occam factor achieved better learning performance than without it. This is because the search with complexity penalty focuses more on a smaller search space while the search without it may explore too large a space to be practical. Since the evolution time is limited to the maximum of $g_{max}$ generations, using the adaptive Occam factor can find a solution faster than without it.

We also measured the convergence time to local minima up to $g_{max}$ generations, i.e. the total learning time until the generation from which there is no improvement in the size and performance of the best individual. Figure 3 (left) summarizes the convergence time as measured in millions of evaluations of arithmetic operations associated with calculating activation values of neural units. Compared with the standard method, the adaptive Occam method converged more than three times faster for this problem. This can be attributed to the reduction by Occam's razor in the number of

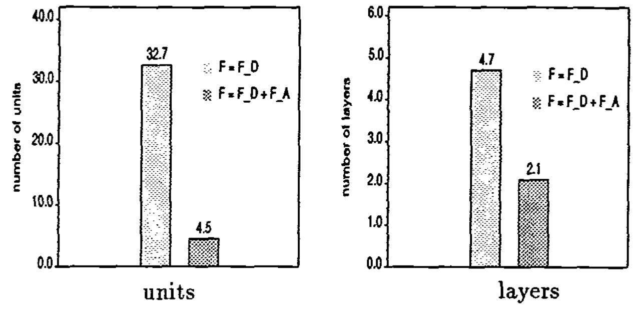

Figure 1: *Comparison of performance with and without complexity penalty, in terms of units and layers.*

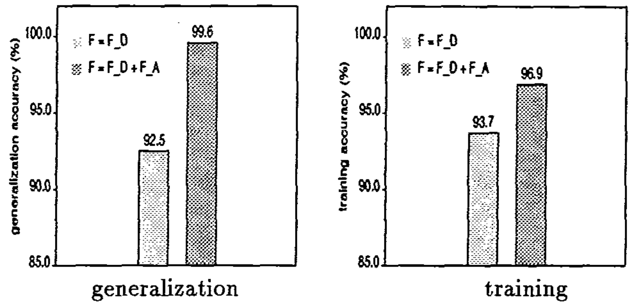

Figure 2: *Comparison of performance with and without complexity penalty, in terms of generalization and training accuracy.*

parameters (weights), as shown in the right figure.

The proposed method has also been applied to the prediction of an environmental system and the result was as good as that obtained by a well-engineered GMDH algorithm (Zhang, Ohm & Mühlenbein 1995).

## Discussion

The minimum description length principle has also been used in other tree-based learning algorithms such as CART (Breiman et al. 1984) and ID3 (Quinlan & Rivest 1989). For the induction of parsimonious decision trees, both CART and ID3 separate growing and pruning phases and the tree complexity is considered only in the pruning phase. This is equivalent to a strategy of "first reducing the error and then reducing the complexity." An implementation of this strategy in the context of genetic programming is not so easy, since, in the genetic programming approach, pruning and growing are interleaved between generations and during the entire learning process. Care must be taken to prune large structures, but without too much loss of structural diversity.

Iba *et al.* have used the MDL principle in genetic programming to evolve GMDH networks (Ivakhnenko 1971) and decision trees (Iba et al. 1993; Iba et al. 1994). As in ID3, the fitness is defined here as simply the sum of error and complexity costs, followed

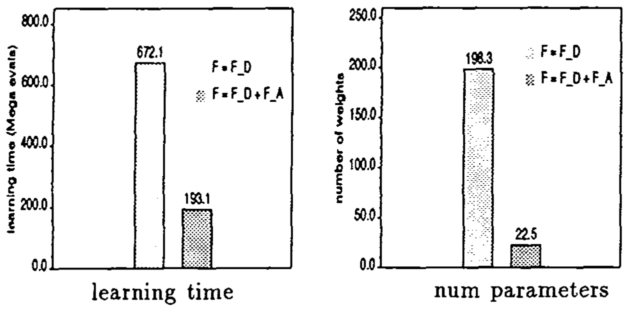

Figure 3: *Comparison of performance with and without complexity penalty, in terms of learning time and number of parameters.*

by a normalization of the total costs. Therefore, the complexity value is as important as the error value in determining the total fitness value of an individual. This works perfectly when the coding scheme exactly reflects the true probability distribution of the environment. One possible drawback in this implementation of the MDL principle in genetic programming is the lack of flexibility in balancing accuracy with parsimony in unknown environments. That is, there is a risk that the network size may be penalized too much, resulting in premature convergence in spite of other diversity-increasing measures, such as a large crossover rate.

The adaptive Occam method described in the previous section tries to avoid premature convergence by normalizing the error and complexity values separately and balancing their relative weight dynamically. The dynamic change of the Occam factor is an improvement over the previous work of the authors (Zhang & Mühlenbein 1993; Zhang & Mühlenbein 1993), where a small constant was used. In early stages of learning, a strong increase in tree complexity is allowed by keeping the Occam factor small, which usually results in fast error reduction. The small Occam factor also results in robust convergence to the desired training accuracy, since premature convergence is avoided due to increased diversity. In later stages, i.e., after the desired level of training performance is achieved, the adaptive Occam approach enforces a strong complexity penalty, which encourages parsimony. Overall, this has the effect of increasing generalization performance without getting stuck in local minima due to premature convergence. The control of the phase transition is not difficult since it is defined by the desired training accuracy which the user requires. Though other MDL-based tree induction methods also reward parsimony, the adaptive Occam approach is different in that it dynamically balances error and complexity costs.

While proposed in a different context, the adaptive fitness function presented in this paper has some similarity in spirit to *competitive fitness functions* (Angeline & Pollack 1993; Siegel 1994). Standard fitness functions return the same fitness for an individual regardless of what other members are present in the population, demanding an accurate and consistent fitness measure throughout the evolutionary process. While the global accuracy can be easily computed when evolving solutions for many simple problems, it is often impractical for problems with greater complexity. In contrast, competitive fitness functions evaluate the fitness values depending on the constituents of the population. Angeline argues that competitive fitness functions provide a more robust training environment than independent fitness functions.

Though the experiments have been done in the context of neural networks, the general method of balancing accuracy and parsimony can be used for the genetic induction of other classes of tree-structured programs as well. This is because the error and complexity values are normalized separately and the same adaptive balancing mechanism can be used for different definitions of error and complexity.

## Acknowledgement

This research was supported in part by the Real World Computing Program under the project SIFOGA.

## References

P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, pages 264–270. Morgan Kaufmann, San Mateo, 1993.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth Int. Group, Belmont, C.A., 1984.

D. B. Fogel. An information criterion for optimal neural network selection. *IEEE Transactions on Neural Networks*, 2(5):490–497, 1991.

H. Iba, H. de Garis, and T. Sato. Genetic programming using a minimum description length principle. In K. E. Kinnear, editor, *Advances in Genetic Programming*, pages 265–284. Cambridge, MA: MIT Press, 1994.

H. Iba, T. Kurita, H. de Garis, and T. Sato. System identification using structured genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, pages 279–286. Morgan Kaufmann, 1993.

A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(4):364–378, 1971.

K. E. Kinnear. Generality and difficulty in genetic programming: Evolving a sort. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, pages 287–294. Morgan Kaufmann, San Mateo, 1993.

K. E. Kinnear, editor. *Advances in Genetic Programming.* Cambridge, MA: MIT Press, 1994.

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

J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press, 1994.

J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, 30(4):629–636, 1984.

J. Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, 14:1080–1100, 1986.

E. V. Siegel. Competitive evolving decision trees against fixed training cases for natural language processing. In K. E. Kinnear, editor, *Advances in Genetic Programming*, pages 409–423. Cambridge, MA: MIT Press, 1994.

W. A. Tackett. Genetic programming for feature discovery and image discrimination. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, pages 303–309. Morgan Kaufmann, San Mateo, 1993.

B.-T. Zhang and H. Mühlenbein. Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex Systems*, 7(3):199–220, 1993.

B.-T. Zhang and H. Mühlenbein. Genetic programming of minimal neural nets using Occam's razor. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 342–349. Morgan Kaufmann, San Mateo, 1993.

B.-T. Zhang and H. Mühlenbein. Synthesis of sigma-pi neural networks by the breeder genetic programming. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 318–323. IEEE Computer Society Press, New York, 1994.

B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

B.-T. Zhang, P. Ohm and H. Mühlenbein. Water pollution prediction with evolutionary neural trees. In *Proceedings of IJCAI Workshop on AI and the Environment.* IJCAI Press, 1995.