Bayesian Inference, Minimum Description Length Principle, and Learning by Genetic Programming[1]

Byoung-Tak Zhang Heinz Mühlenbein

German National Research Center for Computer Science (GMD)
Schloss Birlinghoven, D-53754 Sankt Augustin, Germany
E-mail: zhang@gmd.de, muehlenbein@gmd.de

1 Introduction

The issue of parsimony has been discussed since the birth of genetic programming [8]. While the 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 [6, 16]. In addition, large structures require more computer resources in space and time for the evolution [14]. One approach to dealing with this problem is to define and reuse submodules [9, 14]. Another approach is to take into account of structural complexity in fitness evaluation. This paper focuses on the latter approach, although a combination of both in a single system would be ideal.

We use a Bayesian inference framework to derive a class of fitness measures for dealing with problems of parsimony based on the minimum description length (MDL) principle [13]. The MDL principle has also been used in other tree-based learning algorithms such as CART [2] and ID3 [11], in which growing and pruning phases are separated 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." However, a direct implementation of such a 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.

In the second part of the paper we describe an adaptive technique for implementing the MDL principle in the framework of genetic programming. It automatically balances the ratio of training

[1]Submitted to the Twelfth International Conference on Machine Learning (ML-95) Workshop on Genetic Programming, Tahoe City, California, July 9 - 12, 1995.

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.

2 Bayesian Inference and the MDL Principle

Genetic programming starts with an initial population of computer programs composed of elementary functions and terminals [8, 10, 7]. Genetic operators, such as crossover and selection, are used to adapt the shape and size of the programs and evolve increasingly fit populations. 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

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

where $D = \{(\mathbf{x}_c, \mathbf{y}_c)\}_{c=1}^{N}$ is the training data and $\mathbf{y}_c = \tilde{f}(\mathbf{x}_c)$.

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 β 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:

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

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 [12], 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:

$$L(A|D) \;=\; L(P(D|A)P(A)) = -\log(P(D|A)P(A)) = L(D|A) + L(A), \tag{7}$$

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 [13, 3] 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:

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

The MDL principle has been used in tree-based machine learning algorithms such as CART [2] and ID3 [11]. 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. Thus, 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 to prefer parsimonious solutions based on the performance on the test data. This is equivalent to a strategy of "first reducing the error and then reducing the complexity." But a direct implementation of this strategy in the context of genetic programming is impossible, since, in the genetic programming approach, pruning and growing are interleaved between generations and during the entire learning process.

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

$$F(A|D) \;=\; F_D + F_A = \beta E(D|A) + \alpha C(A), \tag{9}$$

where the parameters α and β 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

$$A_{best} = \arg\min_{A_i \in \mathcal{A}} \{F(A_i|D)\} = \arg\min_{A_i \in \mathcal{A}} \{\beta E(D|A_i) + \alpha C(A_i)\}. \quad (10)$$

The following section describes a general adaptive technique that balances α and β in unknown environments.

3 A Fitness Function for Dynamic Growing/Pruning

As illustrated above, care must be taken in applying the MDL principle to genetic programming so that redundant structures should be pruned as much as possible, but at the same time premature convergence should be avoided. Avoiding the loss of diversity (or encouraging diversity) is especially important in the early stages, while strong pruning is desireable to get parsimonious solutions and improve generalization performance in the final stage (where a certain level of training performance is achieved).

To cotrol this dynamically, we 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 ith individual at generation g. For simplicity, we assume $0 \leq E_i(g) \leq 1$ and $C_i(g) > 0$. Given this, we propose to define the fitness of an individual i at generation g as follows:

$$F_i(g) = E_i(g) + \alpha(g)C_i(g). \quad (11)$$

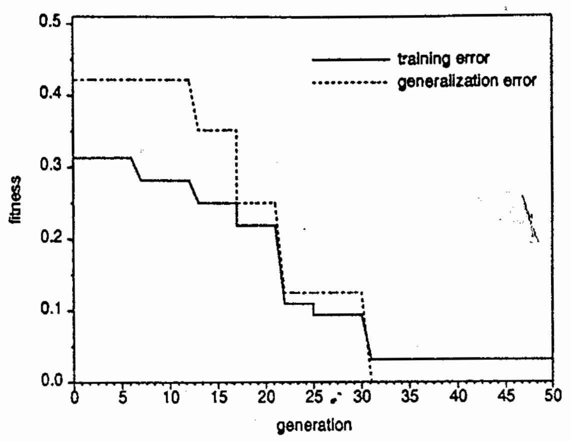
Here $\alpha(g)$ is called the adaptive Occam factor and expressed as

$$\alpha(g) = \begin{cases} \frac{1}{N^2} \frac{E_{best}(g-1)}{\hat{C}_{best}(g)} & \text{if } E_{best}(g-1) > \epsilon \\ \frac{1}{N^2} \frac{1}{E_{best}(g-1) \cdot \hat{C}_{best}(g)} & \text{otherwise,} \end{cases} \quad (12)$$

where N is the size of training set. User-defined constant ϵ 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 [20] 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.



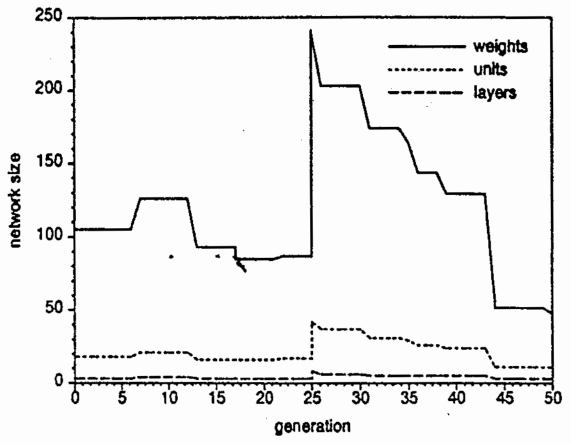


Figure 1: *Evolution of fitness value and network size of the best individuals*

Note also that the equation (12) is a realization of the general form derived from the MDL approach (9) where β is fixed and α is expressed as a function of g: $\beta = 1.0$ and $\alpha = \alpha(g)$.

The performance of the method was studied on the genetic programming of sigma-pi neural networks [19] for solving the parity problem of 7 inputs. The training set consisted of 64 examples which were chosen randomly from $2^7 = 128$ data points and inserted noise by changing the output value with 5% probability. The generalization performance of the best solution in each generation was measured by the complete data set of 128 noiseless examples. The population was initialized for every individual to contain sigma and pi units with 50% probability each. The depth of initialized network was limited to 3. The truncation rate was 50%. The population size and the maximum generation limit were $M = 40$ and $g_{max} = 100$. The parameter ϵ was set to 0.1, requiring the training error should be at most 0.1 or, in other words, at least 90% of the training examples are desired to be learned correctly.

Figure 1 (left) shows the performance evolution of the best individuals in each generation, in terms of training and generalization errors. The network complexity of corresponding individuals are shown in the right figure. The stability of the accuracy-parsimony balancing has been analyzed by observing the error portion of the total fitness: until the error falls below $\epsilon = 0.1$ at generation 26, the error term dominates the fitness, while after that point the relative domination of error term goes down under 1.0 to make stronger complexity reduction, though, without risking too much error increase.

The comparison of the fitness and network size confirms that the network size change, i.e. growing or pruning, has a very close relationship with the training error reduction. For instance, from generation 6 to 7 during which the training error was reduced from 0.31 to 0.28, the network size increased from 3-18-105 (layers-units-weights) to 4-21-126. Other examples of growing and pruning are listed in Table 1.

g	Values at generation g		Values at generation $g+1$		Comments
	$E_{best}(g)$	$C_{best}(g)$	$E_{best}(g+1)$	$C_{best}(g+1)$	
6	0.31	3-18-105	0.28	4-21-126	growing
12	*0.28*	*4-21-126*	*0.25*	*3-16-93*	*pruning*
17	0.25	3-16-93	0.22	3-16-85	pruning
21	0.22	3-16-85	0.11	3-17-87	growing
25	0.11	3-17-87	0.09	8-42-241	growing
30	0.09	6-37-203	0.03	5-31-174	pruning

Table 1: *Error update versus complexity change*

The effectiveness of the adaptive Occam method was studied by comparing its performance with that of the baseline fitness function $F_i(g) = E_i(g) = E(D_N|A_i)$. 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 as described above. For each method, 20 runs were executed to observe the complexity of the best solution and its training and generalization performance at generation $g_{max} = 100$.

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 and improved generalization performance but without losing the training performance. Compared with the standard method, the adaptive Occam method converged more than three times faster for this problem.

4 Discussion

Iba *et al.* also have used the MDL principle in genetic programming to evolve GMDH networks and decision trees [5, 4]. As in ID3, the fitness is defined here as simply the sum of error and complexity costs, followed 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 [17, 18], 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* [1, 15]. 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

[1] 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.

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

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

[4] 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. MIT Press, Cambridge, Cambridge, 1994.

[5] 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.

[6] 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.

[7] K. E. Kinnear, editor. *Advances in Genetic Programming.* MIT Press, Cambridge, Cambridge, 1994.

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

[9] J. R. Koza. Hierarchical automatic function definition in genetic programming. In D. Whitley, editor, *Foundations of Genetic Algorithms (FOGA-92)*, pages 24–29. Morgan Kaufmann, 1992.

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

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

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

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

[14] J. Rosca and D. H. Ballard. Learning by adapting representations in genetic programming. In *Proceedings of IEEE International Conference on Evolutionary Computation (ICEC-94), World Congress on Computational Intelligence*, pages 407–412. IEEE Computer Society Press, New York, 1994.

[15] 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. MIT Press, Cambridge, Cambridge, 1994.

[16] 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.

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

[18] 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.

[19] 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 (ICEC-94), World Congress on Computational Intelligence*, pages 318–323. IEEE Computer Society Press, New York, 1994.

[20] B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1), 1995 (forthcoming).