To appear in
"Advances in GP"
(MIT Press, 1994).

ETL-TR-93-15
received August 4, 1993

Genetic Programming Using a Minimum Description Length Principle

Hitoshi IBA[1] Hugo de GARIS[2] Taisuke SATO[1]

1)Machine Inference Section, Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba-city, Ibaraki, 305, Japan
{iba,sato}@etl.go.jp, +81-298-58-5918
2)ATR Human Information Processing Research Laboratories,
2-2 Hikari-dai, Seiko-cho, Soraku-gun, Kyoto, 619-02, Japan.
degaris@hip.atr.co.jp, +81-7749-5-1079

Abstract This paper introduces a Minimum Description Length (MDL) principle to define fitness functions in Genetic Programming (GP). In traditional (Koza-style) GP, the size of trees was usually controlled by user-defined parameters, such as the maximum number of nodes and maximum tree depth. Large tree sizes meant that the time necessary to measure their fitnesses often dominated total processing time. To overcome this difficulty, we introduce a method for controlling tree growth, which uses an MDL principle. Initially we choose a "decision tree" representation for the GP chromosomes, and then show how an MDL principle can be used to define GP fitness functions. Thereafter we apply the MDL-based fitness functions to some practical problems. Using our implemented system called "STROGANOFF", we show how MDL-based fitness functions can be applied successfully to problems of pattern recognition.

1 Introduction

Most of Genetic Programming (GP) [1] computation time is taken up in measuring the fitness of GP trees. Naturally, the amount of time depends directly upon the size of the trees [Koza92]. The size of GP trees was usually controlled by user-defined parameters, such as the maximum number of nodes or maximum tree depth. We tried Koza-style GP to obtain a solution tree to the 6-multiplexor problem [Higuchi93], (see equation (1)), using the function set [AND, OR, IF, NOT], and using the address and data variables as terminal set. The goal or target tree is shown in Fig.1. However, due to the lack of a fitness function to control the explosive growth (i.e. depth) of trees, the tree growth got out of hand and the result was unsatisfactory. To reduce the size of trees, Koza proposed a hierarchical automatic function definition (rather like a macro call) to replace duplicate subtrees [Koza93]. This method is heavily dependent on the diversity of very large population sizes (e.g. 4000), which are beyond normal computer resources.

To overcome this difficulty, we introduce a Minimum Description Length (MDL) principle [Rissanen78] to define fitness functions in GP, so as to control the growth of trees. We choose

[1]Note that throughout this paper, the term Genetic Programming (GP) is used according to the Koza definition [Koza92], and not according to the deGaris definition [deGaris93]. Unfortunately, there are two definitions of GP in the literature.

a "decision tree" representation for the chromosomes, and show how an MDL principle can be used to define the GP fitness functions. We then apply the MDL-based fitness functions to some practical problems. In earlier papers [Iba93a,b], we presented a software system called "STROGANOFF" for system identification. This system integrate both analog and digital approaches, and its fitness evaluation was based upon an MDL principle. Using STROGANOFF, we show how our MDL-based fitness functions can also be applied successfully to problems of pattern recognitions.

2 GP using an MDL principle

In this section, we describe how "decision trees" can be evolved with GP to solve Boolean concept learning problems. Boolean concept learning has recently been developed in an attempt to formalize the field of machine learning [Anthony92]. We use Boolean concept learning as a means to treat the validity of MDL-based fitness functions. The reason why we chose a "decision tree" representation for the GP chromosomes is discussed later in Section 4.

2.1 Decision Trees and Genetic Programming

"Decision trees" were proposed by Quinlan for concept formation in machine learning [Quinlan83]. Generating efficient decision trees from preclassified (supervised) training examples, has generated a large literature. Decision trees can be used to represent Boolean concepts. Fig.2 shows a decision tree which solves the 6-multiplexor problem, where a_0, a_1 are the multiplexor addresses and d_0, d_1, d_2, d_3 are the data. The target concept is

$$output = \overline{a_0}\ \overline{a_1} d_0 + a_0 \overline{a_1} d_1 + \overline{a_0} a_1 d_2 + a_0 a_1 d_3 \tag{1}$$

A decision tree is a representation of classification rules, e.g. the subtree at the left in Fig.2 shows that the output becomes false (i.e. 0) when the variable a_0 is 0, a_1 is 0, and d_0 is 0.

Koza discussed the evolution of decision trees within GP framework and conducted a small experiment called a "Saturday morning problem" [Koza90]. However, as we shall see in later experiments (e.g. Fig5(b)), Koza-style simple GP fails to evolve effective decision trees due to difficulties in defining suitable fitness functions. To overcome this problem, we introduces fitness functions based on an MDL principle.

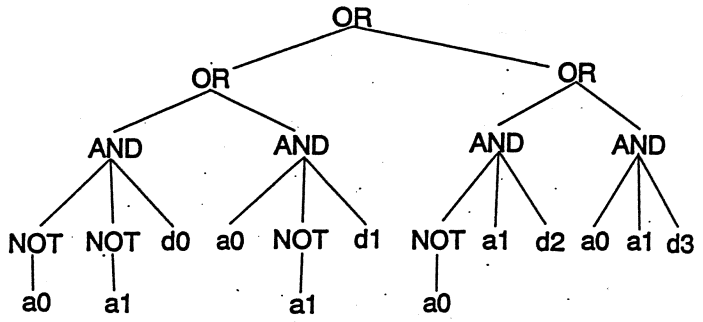


Fig.1 A Goal Tree in Koza-style GP

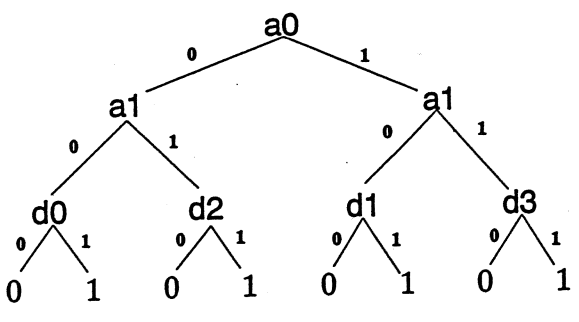


Fig.2 A Desired Decision Tree

2.2 MDL-based fitness functions

We introduce MDL-based fitness functions for evaluating the fitness of GP tree structures. This fitness definition involves a tradeoff between the details of the tree, and the errors. In general, the MDL fitness definition for a GP tree (whose numerical values is represented by "mdl") is defined as follows:-

$$mdl = (Tree\_Coding\_Length) + (Exception\_Coding\_Length) \quad (2)$$

The "mdl" of a decision tree is calculated using the following method [Quinlan89]. Consider the decision tree in Fig.3 for the 6-multiplexor problem (X, Y and Z notations are explained later).

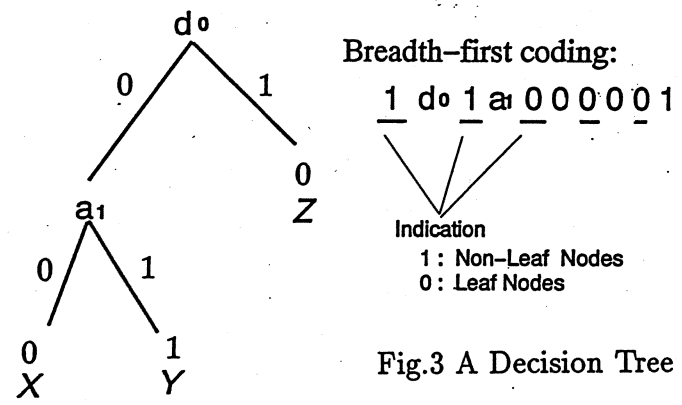


Fig.3 A Decision Tree for the 6-Multiplexor Problem

Using a breadth-first traversal, this tree is represented by the following string [1]:-

$$1\ d_0\ 1\ a_1\ 0\ 0\ 0\ 0\ 0\ 1 \quad (3)$$

To encode this string in binary format,

$$2 + 3 + 2 \times log_2 6 + 3 \times log_2 2 = 13.17\ bits \quad (4)$$

bits are required since the codes for each non-leaf (d_0, a_1) and for each leaf (0 or 1) require $log_2 6$ and $log_2 2$ bits respectively, and $2+3$ bits are used for their indications. In order to code exceptions (i.e. errors or incorrect classifications), their positions should be indicated. For this purpose, we divide the set of objects into classified subsets. For the tree of Fig.3, we have the following three subsets (which we call X, Y and Z from left to right):-

Name	Attributes	# of elements	# of correct cl.	# of incorrect cl.
X	$d_0 = 0 \wedge a_1 = 0$	16	12	4
Y	$d_0 = 0 \wedge a_1 = 1$	16	8	8
Z	$d_0 = 1$	16	12	20

Table 1 Classified Subsets for Encoding Exceptions

For instance, X is a subset whose members are classified into the leftmost leaf ($d_0 = 0 \wedge a_1 = 0$). The number of elements belonging to X is 16. 8 members of X are correctly classified (i.e. 0). Misclassified elements (i.e. 4 elements of X, 8 elements of Y, and 20 elements of Z) can be coded with the following cost:-

$$L(16, 4, 16) + L(16, 8, 8) + L(32, 20, 32) = 65.45\ bits, \quad (5)$$

[1]Since in our decision trees, left (right) branches always represent 0 (1) values for attribute-based tests, we can omit those attribute values.

where

$$L(n,k,b) = log_2(b+1) + log_2(\binom{n}{k}). \tag{6}$$

$L(n,k,b)$ is the total cost for transmitting a bit string of length n, in which k of the symbols are 1's and b is an upper bound on k. Thus the total cost for a decision tree in Fig.3 is 78.62 ($= 65.45 + 13.17$) bits.

In general, the coding length for a decision tree with n_f attribute nodes and n_t terminal nodes is given as follows:-

$$Tree\_Coding\_Length = (n_f + n_t) + n_t log_2 Ts + n_f log_2 Fs \tag{7}$$

$$Exception\_Coding\_Length = \sum_{x \in Terminals} L(n_x, w_x, n_x) \tag{8}$$

Where Ts and Fs are the total numbers of terminals and functions respectively. In equation (8), summation is taken over all terminal nodes. n_x is the number of elements belonging to the subset represented by x. w_x is the number of misclassified elements of n_x members.

In order to use this MDL value as a fitness function, we need to consider the following points:-

1. The optimum mdl value is found by minimization, i.e. the smaller, the better.
2. Although mdl values of "decision trees" are non-negative, the general definition of mdl (equation (2)) allows arbitrary values, i.e. range from $-\infty$ to ∞.

Since the range of "mdl" is not known beforehand, a scaling technique must be used to translate the mdl value into a form suitable for a fitness function. For this purpose, we use a "scaling window" [Schraudolph92]. Let $Pop(t)$ be a population of chromosomes at the t-th generation, and let N be its population size, i.e.

$$Pop(t) = \{g_t(1), g_t(2), \cdots g_t(N)\}. \tag{9}$$

Each individual $g_t(i)$ has a data structure consisting of three components, i.e. its genotype, its mdl, and its fitness.

$$g_t(i): \quad \text{Genotype} \quad Gtype_t(i) \tag{10}$$

$$\text{mdl value} \quad mdl_t(i) \tag{11}$$

$$\text{fitness value} \quad fitness_i(t), \tag{12}$$

then our GP algorithm using MDL-based fitnesses works as follows:-

Step1 (Initialization) Let $t := 1$ (generation count).
Step2 Call **GP** and generate $Pop(t)$.
Step3 Calculate the mdl value $\{mdl_t(i)\}$ for each genotype $\{Genotype_t(i)\}$.
Step4 $Worst(t) := \max_{i=1\cdots N}\{mdl_t(i)\}$.
Step5 $Wmdl := \max\{Worst(t), Worst(t-1), \cdots Worst(t-Wsize)\}$.
Step6 For i:=1 to N do $fitness_t(i) := Wmdl - mdl_t(i)$.
Step7 $t := t+1$. Go to Step2.

Where $Wsize$ designates the size of scaling window. With this technique, the mdl value is translated into the windowed fitness value, so that the smaller an mdl value is, the larger its fitness is.

2.3 Evolving decision trees

In this section, we present results of the experiments to evolve decision trees for the 6-multiplexor problem using GP. The following parameters were used.

Population size	100
Probability of Graph crossover	0.6
Probability of Graph mutation	0.0333
Terminal set	{0, 1}
Non-terminal set	$\{a_0, a_1, d_0, d_1, d_2, d_3\}$

Table 2 GA Parameters

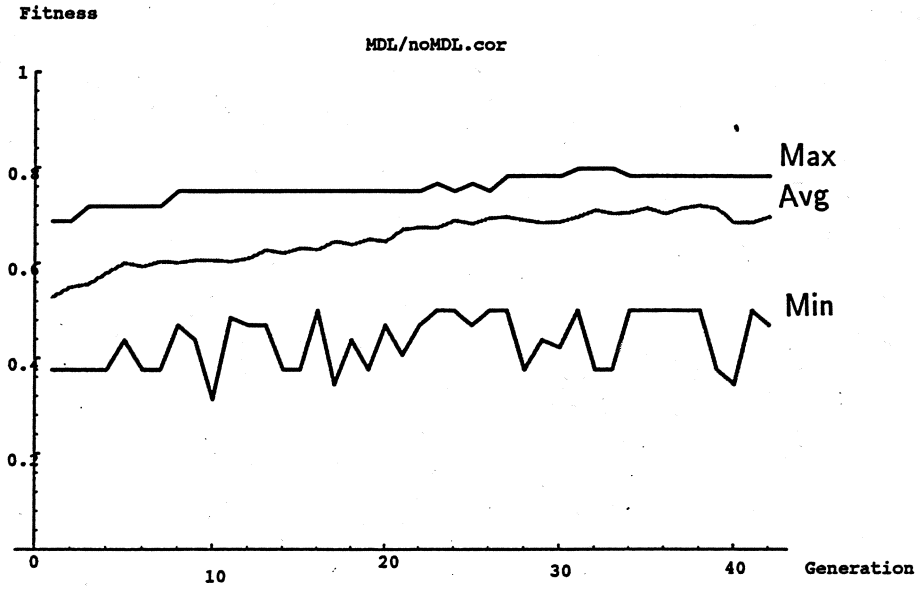
Where 0 (1) value in the terminal set represents a positive (negative) example, i.e. true (false) value. Symbols in the non-terminal set are attribute-based test functions. For the sake of explanation, we use S-expressions to represent decision trees from now on. The S-expression $(X\ Y\ Z)$ means that if X is 0 (false) then test the second argument Y and if X is 1 (true) then test the third argument Y. For instance, $(a_0\ (d_1\ (0)\ (1))\ (1))$ is a decision tree which expresses that if a_0 is 0 (false) then if d_0 is 0 (false) else 1 (true) and that if a_1 is 1 (true) then 1 (true).

Fig.4 shows results of experiments (correct classification rate vs. generations) using a traditional (non-MDL) fitness function (a), and using an MDL-based fitness function (b), where the traditional (non-MDL) fitness is defined as the rate of correct classification. The desired decision tree was acquired at the 40-th generation when using an MDL-based fitness function. However, the largest fitness value at the 40-th generation when using a non-MDL fitness function was only 78.12%. Fig.5 shows the evolution of the "mdl" values in the same experiment as Fig.4(b). The acquired structure at the 40-th generation using an MDL-based fitness function was as follows:-

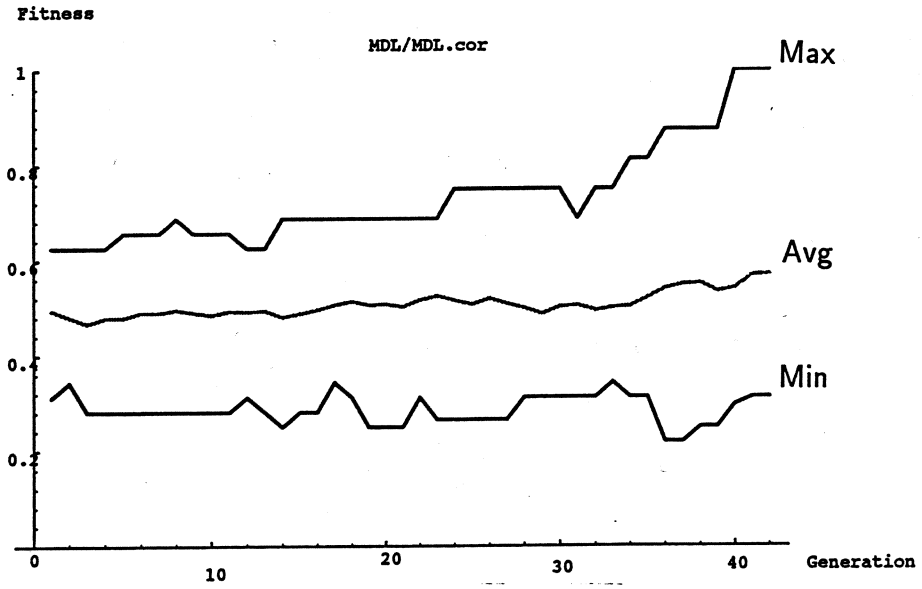
```
(A0
   (A1
      (D0
         (A1 (D0 (0) (0)) (D2 (0) (A0 (1) (1))))
         (1))
      (D2 (0) (1)))
   (A1 (D1 (0) (1)) (D3 (0) (1))))
```

whereas the typical genotype at the same generation (40-th) using a non-MDL fitness function was as follows:-

```
(D1
   (D2
      (D3 (0)
         (A0 (0)
            (D3 (A0 (0) (0))
               (D0 (D0 (1) (D1 (A0 (0) (1)) (D0 (D0 (0) (0)) (A1 (1) (1)))))
                  (D1 (0) (0))))))
      (A0
         (A0
            (D1 (1)
```



(a) Non-MDL Fitness



(b) MDL-based Fitness

Fig.4 Experimental Result (Fitness vs. Generations)

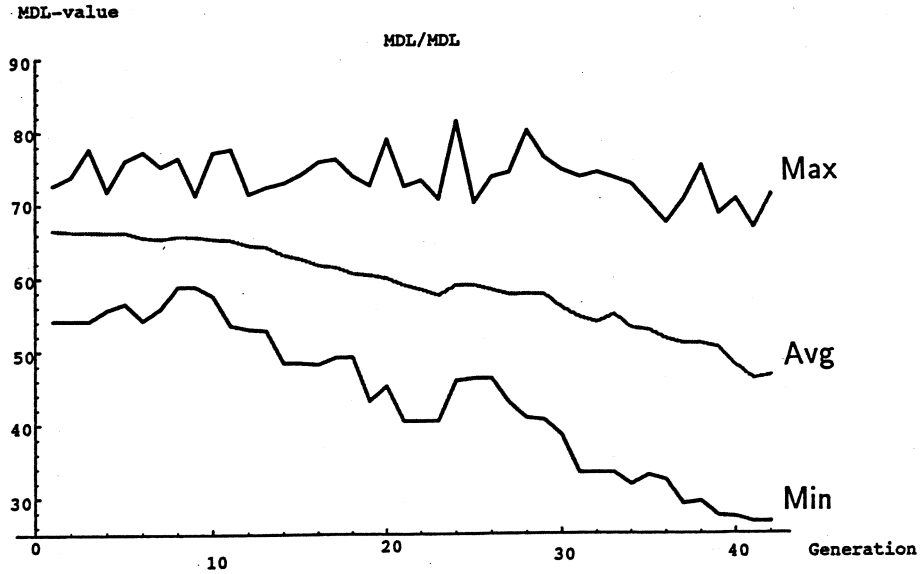


Fig.5 MDL vs. Generations

```
                (D1 (A0 (D1 (1) (1)) (A0 (D0 (D2 (1) (D1 (D1 (0) (0)) (1))) (0)) (1)))
                   (0)))
             (A0
                (A0
                   (D2
                      (A1 (1)
                         (D0
                            (D3 (0)
                               (A1 (D0 (A0 (1) (1)) (D3 (D0 (1) (0)) (A0 (0) (1))))
                                  (A1
                                     (D2
                                        (D2 (D0 (D2 (A1 (D3 (0) (D3 (0) (0))) (A1 (0) (1)))
                                                    (D3 (D3 (0) (0)) (1)))
                                                (D2 (1) (D0 (1) (0))))
                                            (0))
                                        (D0 (D3 (D2 (A0 (D3 (0) (0)) (1)) (1)) (1)) (1)))
                                     (1))))
                               (D0 (D3 (A0 (1) (A0 (0) (A1 (0) (A0 (1) (1))))) (D3 (1) (0))) (1))))
                      (1))
                   (A0 (A0 (0) (0)) (A1 (0) (D0 (1) (1)))))
                (A1
                   (A0 (1)
                      (D3 (D1 (D2 (D1 (0) (A0 (1) (D3 (D1 (0) (1)) (1)))) (0)) (1)) (1)))
                   (0))))
          (0)))
    (D2 (A1 (A1 (1) (0)) (A1 (1) (0))) (1)))
```

As can be seen, the non-MDL fitness function did not control the growth of the decision trees, whereas using an MDL-based fitness function led to a successful learning of a satisfactory data structure. Thus we can conclude that an MDL-based fitness function works well for the 6-multiplexor problem.

3 Evolving trees with an MDL-based fitness function

This section describes the application of MDL-based fitness functions to more practical problems, using our implemented system called "STROGANOFF' (i.e. STructured Representation On Genetic Algorithms for NOn-linear Function Fitting). This system uses a hierarchical multiple regression analysis method (GMDH, Group Method of Data Handling) and a structured GA-based search strategy [Iba93a]. The aim of our system is to establish an adaptive learning method for system identification problems. System identification techniques are used in many fields to predict the behaviors of unknown systems, given input-output data. This problem is defined formally in the following way. Assume that the single valued output y, of an unknown system, behaves as a function f of m input values, i.e.

$$y = f(x_1, x_2, \cdots, x_m) \tag{13}$$

Given N observations of these input-output data pairs, i.e.

INPUT				OUTPUT
x_{11}	x_{12}	$\cdots$	x_{1m}	y_1
x_{21}	x_{22}	$\cdots$	x_{2m}	y_2
		$\cdots$		$\cdots$
x_{N1}	x_{N2}	$\cdots$	x_{Nm}	y_N

the system identification task is to approximate the true function f with $\overline{f}$. Once this approximate function $\overline{f}$ has been estimated, a predicted output $\overline{y}$ can be found for any input vector $(x_1, x_2, \cdots, x_m)$, i.e.

$$\overline{y} = \overline{f}(x_1, x_2, \cdots, x_m), \tag{14}$$

This $\overline{f}$ is called the "complete form" of f.

GMDH is a multi-variable analysis method which is used to solve system identification problems [Ivakhnenko71]. This method constructs a feedforward network (as shown in Fig.6(a)) as it tries to estimate the output function $\overline{y}$. The node transfer functions (i.e. the Gs in Fig.6(a)) are quadratic polynomials of the two input variables (e.g. $G(z_1, z_2) = a_0 + a_1 z_1 + a_2 z_2 + a_3 z_1 z_2 + a_4 z_1^2 + a_5 z_2^2$) whose parameters a_i are obtained using regression techniques. GMDH uses the following algorithm to derive the "complete form" $\overline{y}$.

Step1 Let the input variables be $x_1, x_2, \cdots, x_m$ and the output variable be y. Initialize a set labeled VAR using the input variables, i.e. $VAR := \{x_1, x_2, \cdots, x_m\}$.

Step2 Select any two elements z_1 and z_2 from the set VAR. Form an expression $G_{z1,z2}$ which approximates the output y (in terms of z_1 and z_2) with least error using multiple regression techniques. Regard this function as a new variable z,

$$z = G_{z1,z2}(z_1, z_2). \tag{15}$$

Step3 If z approximates y better than some criterion, set the "complete form" (i.e. $\overline{y}$) as z and terminate.

Step4 Else $VAR := VAR \cup \{z\}$. Go to Step2.

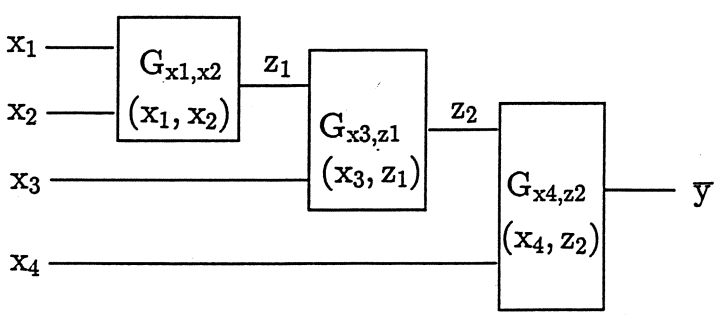


Fig.6(a) GMDH Network

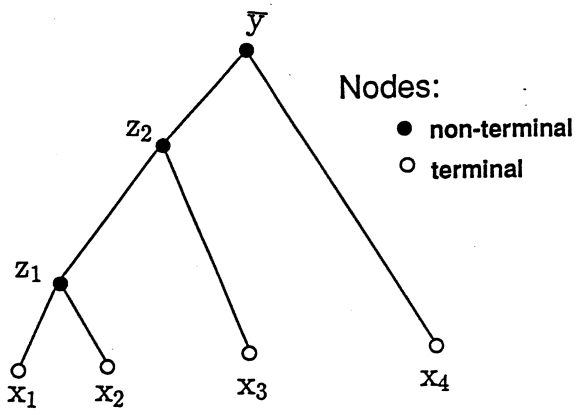


Fig.6(b) Equivalent Binary Tree

The "complete form" (i.e. $\overline{y}$) given by the GMDH algorithm can be represented in the form of a tree. For example, consider the GMDH network of Fig.6(a), where the "complete form" $\overline{y}$ is given as follows:-

$$z_1 = G_{x1,x2}(x_1, x_2) \tag{16}$$

$$z_2 = G_{x3,z1}(x_3, z_1) \tag{17}$$

and

$$\overline{y} = G_{x4,z2}(x_4, z_2) \tag{18}$$

where (16) and (17) are subexpressions given in Step2 of the GMDH algorithm. By interpreting these subexpressions as non-terminals and the original input variables $\{x_1, x_2, x_3, x_4\}$ as terminals, we get the tree shown in Fig.6(b). This tree can be written as a (Lisp) S-expression,

(NODE1
  (NODE2
    (NODE3 (x_1) (x_2))
    (x_3)
  (x_4)))

where NODE1, NODE2 and NODE3 represent intermediate and output variables $\overline{y}, z_2, z_1$ respectively. Each internal node records the information corresponding to its G expression.

In this paper, each G expression takes the following quadratic form.

$$G_{z1,z2}(z_1, z_2) = a_0 + a_1 z_1 + a_2 z_2 + a_3 z_1 z_2 + a_4 z_1^2 + a_5 z_2^2 \tag{19}$$

If z_1 and z_2 are equal (i.e. $z_1 = z_2 = z$), G is reduced to

$$G_z(z) = a'_0 + a'_1 z + a'_2 z^2 \tag{20}$$

The coefficients a_i are calculated using a least mean square method. Details are described in [Iba93a].

We used an MDL-based fitness function for evaluating the tree structures. As mentioned in Section 2, this fitness definition involved a tradeoff between the details of the tree, and the errors (see equation (2)). The MDL fitness definition for a GMDH tree is defined as follows [Tenorio90]:-

$$Tree\_Coding\_Length = 0.5k log N \tag{21}$$

$$Exception\_Coding\_Length = 0.5N log S_N^2 \tag{22}$$

where N is the number of data pairs, S_N^2 is the mean square error, i.e.

$$S_N^2 = \frac{1}{N} \sum_{i=1}^{N} | \overline{y_i} - y_i |^2 \tag{23}$$

and k is the number of parameters of the tree, e.g. the k-value for the tree in Fig.6(b) is $6 + 6 + 6 = 18$.

STROGANOFF was applied to several applications such as pattern recognition and time series prediction [Iba93a]. We briefly present the results of an experiment called "SONAR" (i.e. classifying sonar echoes as coming from a submarine or not) [Gorman88]. In the "SONAR" experiment, the number of data was 208 and the number of features was 60. The following parameters were used.

Population size	60
Probability of Graph crossover	0.6
Probability of Graph mutation	0.0333
Terminal set	$\{(1),(2),(3),\cdots(60)\}$

Table 3 GA Parameters (2)

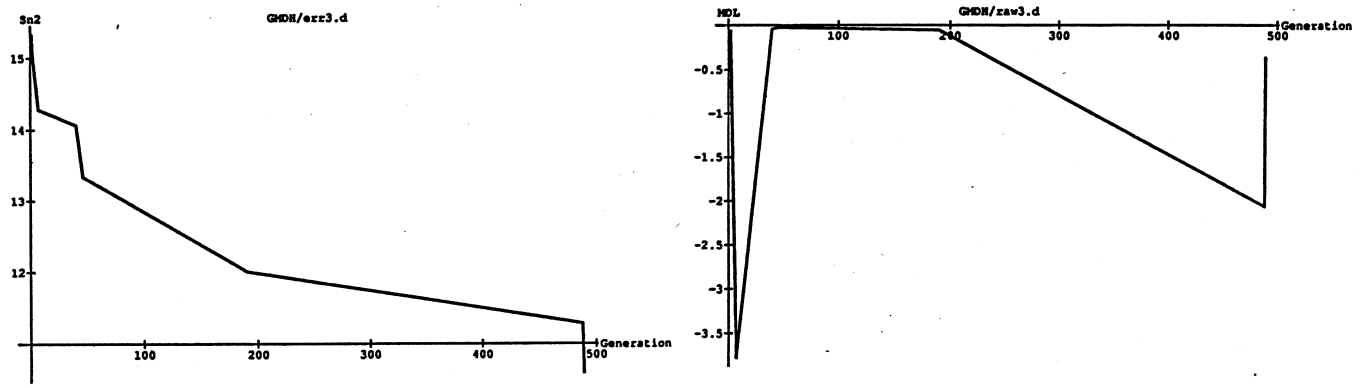
Where (i) in the terminal set signifies the i-th feature. The final expression $\overline{y}$ was then passed through a threshold function.

$$\textbf{Threshold}(\overline{y}) = \begin{cases} 0, & \overline{y} < 0.5, \quad \text{negative example} \\ 1, & \overline{y} \geq 0.5, \quad \text{positive example} \end{cases} \tag{23}$$

Exp.1

We randomly divided 208 data points into two groups, 168 training data and 40 testing data.

Fig.7 shows the results, i.e. mean square error and MDL vs. the number of generations. The elite evolved structure at the 489th generation classified the training data with 92.85% accuracy, and the testing data with 78.94% accuracy. This result is comparable to that of [Gorman88].



(a) Error-of-fit vs. Generations (b) MDL vs. Generations

Fig.7 Pattern Recognition (Sonar Data)

The resulting elite structure evolved at the 489th generation is shown below with node coefficients:-

```
(NODE35040
   (NODE35041 (54)
      (NODE35042
         (NODE35043 (34)
            (NODE35044 (22)
               (NODE35045
                  (NODE35046
                     (NODE35047 (46) (NODE35048 (40) (NODE34446 (39) (32))))
                     (NODE34650
                        (NODE34623
                           (NODE34624
                              (NODE34625 (36)
                                 (NODE34626
                                    (NODE34627
                                       (NODE34628 (35)
                                          (NODE34629 (NODE34630 (42) (20))
```

```
                                        (NODE34631 (48) (NODE34632 (NODE34633 (52) (23)) (48)))))
                                    (NODE34634 (NODE34635 (NODE34636 (28) (24)) (41))
                                        (NODE34637 (9) (17))))
                                (NODE34638
                                    (NODE34639 (NODE34640 (17) (43)) (NODE34641 (26) (38)))
                                    (NODE34642 (1) (30)))))
                            (31))
                        (40))
                    (26)))
                (NODE34667 (NODE34668 (44) (NODE34453 (52) (52)))
                    (NODE34454
                        (NODE34455
                            (NODE34456
                                (NODE34457
                                    (NODE34458 (60)
                                        (NODE34459 (2)
                                            (NODE34460
                                                (NODE34461 (59) (NODE34462 (4) (NODE34463 (36) (42))))
                                                (4))))
                                    (NODE34464 (NODE34465 (54) (NODE34466 (28) (14)))
                                        (NODE34467 (48)
                                            (NODE34468 (NODE34469 (NODE34470 (51) (22)) (13))
                                                (NODE34471
                                                    (NODE34472 (NODE34473 (34) (NODE34474 (46) (47))) (57))
                                                    (20))))))
                                (NODE34475 (NODE34476 (52) (NODE34477 (53) (20)))
                                    (NODE34478 (42) (3))))
                            (NODE34479
                                (NODE34480
                                    (NODE34481 (29)
                                        (NODE34482 (NODE34483 (NODE34484 (58) (58)) (19))
                                            (NODE34485 (58) (35))))
                                    (NODE34486 (57)
                                        (NODE34487 (16)
                                            (NODE34488 (41)
                                                (NODE34489 (NODE34490 (NODE34491 (31) (43)) (37))
                                                    (NODE34492 (18) (16)))))))
                                (NODE34493 (47) (14))))
                        (NODE34494 (23) (NODE34495 (57) (23)))))))
    (6)))
(NODE34151 (NODE33558 (NODE33548 (NODE33549 (50) (49)) (25)) (56))
```

Node	NODE34152	NODE33549	NODE33548
a_0	1.484821	0.89203346	0.27671432
a_1	-3.2907495	2.631569	0.79503393
a_2	-12.878513	-14.206657	-0.66170883
a_3	1.5597038	-256.3501	0.95752907
a_4	3.294608	275.04077	-0.395895
a_5	55.61087	111.96045	0.11753845

Table 4 Node Coefficients

Exp.2

To compare the performance of STROGANOFF with those of neural networks, we conducted an experiment using the same conditions as described in [Gorman88]. The combined set of 208 cases was divided randomly into 13 disjoint sets so that each set consists of 16. For each experiment, 12 of these sets were used as training data, while the 13th was reserved for testing. The experiment was repeated 13 times in order for every case to appear once as a test set.

The reported performance was obtained by averaging the 13 different test sets, with each set run 10 times.

The results of this experiment are as follows:-

	Training data		Testing data	
	Avg.	Std.	Avg.	Std.
STROGANOFF	88.9	1.7	85.0	5.7
NN (0)	89.4	2.1	77.1	8.3
NN (2)	96.5	0.7	81.9	6.2
NN (3)	98.8	0.4	82.0	7.3
NN (6)	99.7	0.2	83.5	5.6
NN (12)	99.8	0.1	84.7	5.7
NN (24)	99.8	0.1	84.5	5.7

Table 5 STROGANOFF vs. NN

where the table shows averages and standard deviations of accuracy for test and training data. NN (i) indicates a neural network with a single hidden layer of i hidden units. The neural data was reported by Gorman and Sejnowski [Gorman88].

The following points from the table should be noticed.

1. STROGANOFF is not necessarily as good as neural networks at learning training data.
2. Judging from the neural testing data results, neural networks may suffer from an overfitting of the training data.
3. STROGANOFF established an adequate generalization resulting in good results for the testing data.

Thus we conclude that the MDL-based fitness functions can be used for effective control of tree growth. These methods appear to achieve good results for the testing data by not fitting the training data too strictly. This generalizing ability of STROGANOFF is desirable not only for pattern recognition tasks but for other applications as well [Iba93a,b].

4 Discussion

Previous sections have shown the validity of MDL-based fitness functions for Genetic Programming. However MDL cannot be applied to every kind of problems to be solved by GP. Trees evolved in the previous experiments had the following characteristics in common.

Size-based Performance	The more the tree grows, the better its performance (fitness).
Decomposition	The fitness of a substructure is well-defined itself.

The first point is a basis for evaluating the tradeoff between the tree description and the error. Decision trees and GMDH networks are typical examples.

The second point claims that the fitness of a subtree (substructure) reflects that of the whole structure, i.e. if a tree has good substructures, its fitness is necessarily high. For instance,

when applying an original Koza-style GP to the 6-multiplexor problem, the function set [AND, OR, IF, NOT] and the terminal set of address and data variables were used. Consider the substructure $(OR\ x_1\ x_2)$ of the tree shown in Fig.8(a). The whole value of this tree is heavily dependent on the root node as shown below:-.

f	$x_1 \vee x_2$	returned value	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1
root label		AND	OR

Table 6 Truth Table (1)

This table shows that changing the root node from an AND to an OR causes a vast change in the truth value of the whole tree. Hence the fitness of the substructure cannot be evaluated itself. It is affected by the whole structure, especially by the upper nodes. On the other hand, consider the decision tree in Fig.8(b). The truth table is as follows:-

x	d_1	returned value
0	0	f
0	1	f
1	0	0
1	1	1

Table 7 Truth Table (2)

As can be seen, the truth value of the substructure $(d_1\ (0)\ (1))$ is kept unchanged whatever the root node x is, i.e. if x is 1 then the returned value is always d_1. In this way, the semantics of a substructure are kept, even if its upper nodes are changed. It follows that the fitness of a substructure is well-defined itself. The same argument is true of a GMDH network (Fig.8(c)). In a GMDH network, if subnetworks z_1 and z_2 are combined into x with $x = G(z_1, z_2)$, then the fitness of x is better (i.e. no worse) than both z_1 and z_2 because of the layered regression analysis. Hence a GMDH tree which has good substructures is expected to have high fitness.

MDL-based fitness functions work well for controlling tree growth when the above two conditions are satisfied. The first point guarantees that MDL can be used to evaluate the trade off between the tree description and the error. The second point claims that good subtrees work as building-blocks for MDL-based fitness functions. Hence these points should be considered carefully before applying MDL-based fitness functions to GP. Representation of chromosomes is essential when designing effective GP techniques.

5 Conclusion

This paper has proposed an MDL principle to define fitness functions for GP. A "decision tree" representation was chosen for the chromosomes. We then showed how the MDL principle could be used to define GP fitness functions. Using our STROGANOFF system, we showed that

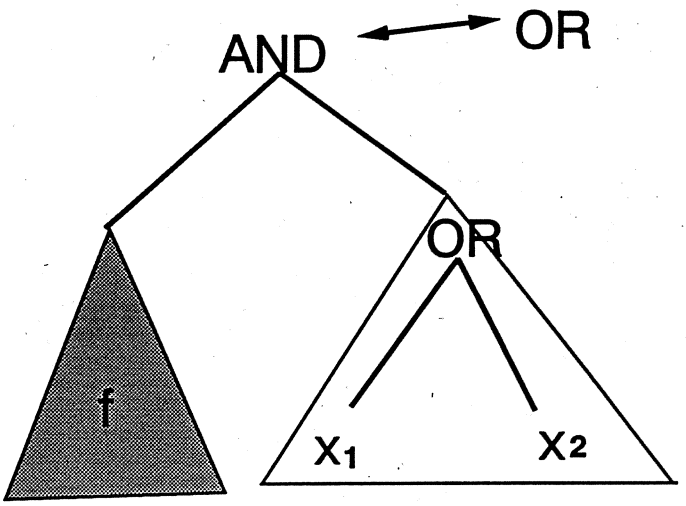


Fig.8(a) Koza-style Tree

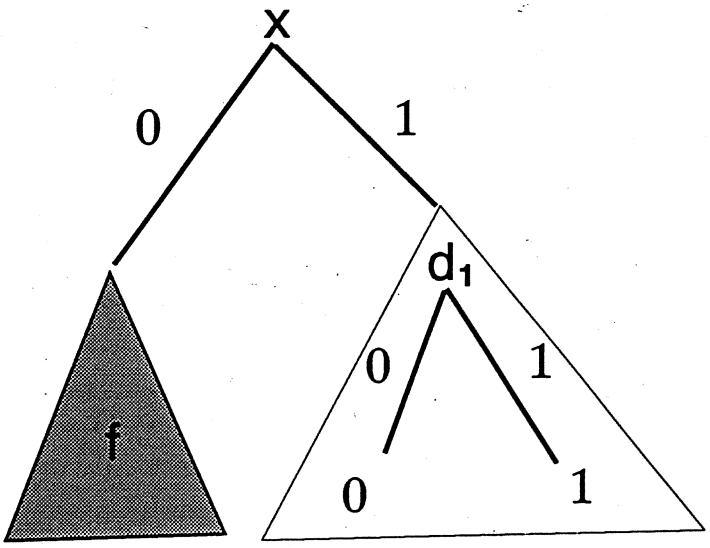


Fig.8(b) Decision Tree

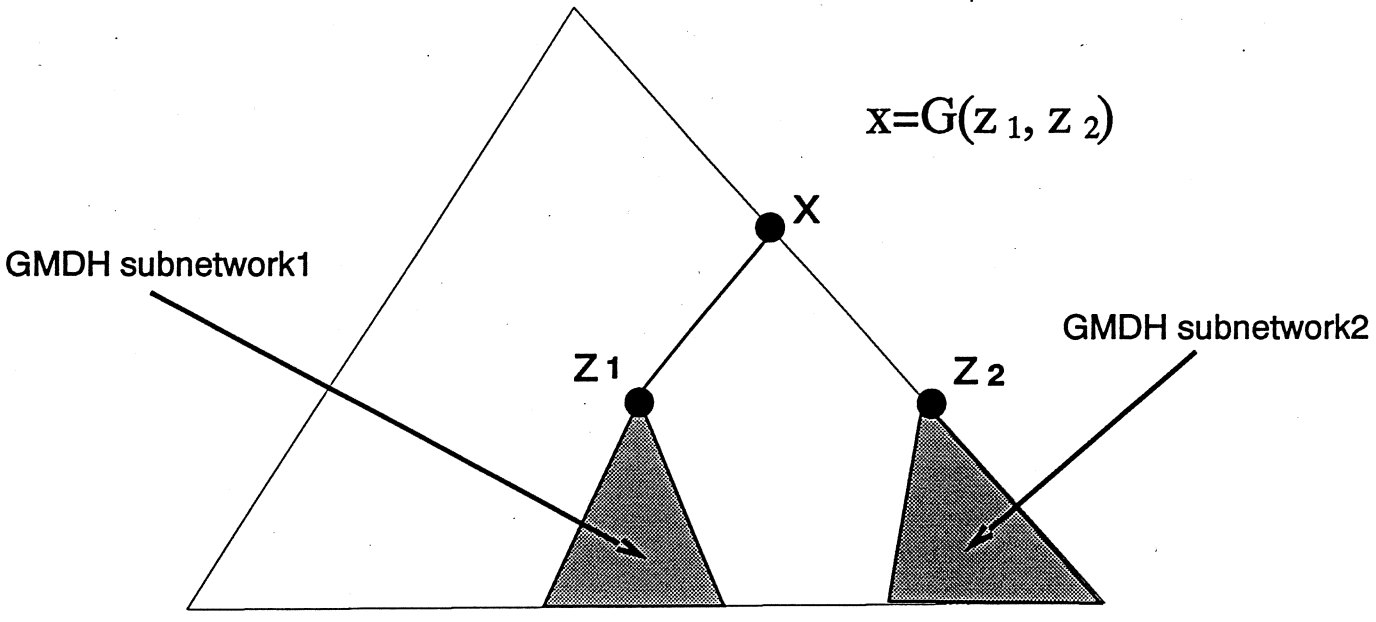


Fig.8(c) GMDH Network