Evolutionary GMDH-based Identification of Building Blocks for Binary-Coded Systems

Ehsan Nazerfard, Saeed Bagheri Shouraki
Artificial Creatures Lab
Computer Engineering Department
Sharif University of Technology
Tehran, Iran
{nazerfard, sbagheri}@ce.sharif.edu

Vesal Hakami
Intelligent Systems Lab
Computer Engineering Department
Amirkabir University of Technology
Tehran, Iran
vhakami@ce.aut.ac.ir

Abstract

This paper proposes an approach to the problem of building block extraction in the context of evolutionary algorithms (with binary strings). The method is based upon the construction of a GMDH neural network model of a population of promising solutions with the aim of extracting building blocks from the resultant network. The operation of the proposed method is regardless of the order by which building blocks are positioned in strings representing the solutions. The experiments are carried out on some well-known benchmark functions including DeJong's.

1. Introduction

Inductive modeling aims at constructing an efficient and robust model of high dimensional data. As illustrated in figure 1, in a given set of inputs, system state, and outputs, the third component is always deducible with the other two at hand. For instance, given the input training data and the system model, the output associated with the unseen input data is estimative (prediction model). Likewise, a modeling problem is intended to model a system while knowing the set of inputs and outputs. Finally, in a control problem the goal is to find the best inputs for a given system with known outputs.

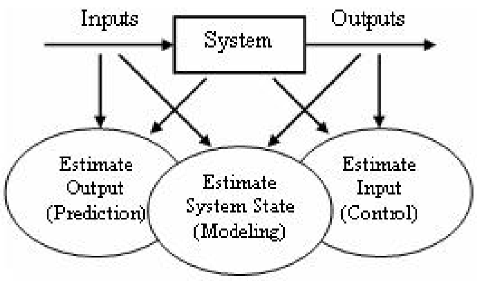
Inputs
System
Outputs
Estimate Output (Prediction)
Estimate System State (Modeling)
Estimate Input (Control)

Figure 1. Inductive Modeling

The inductive modeling strategies can be categorized as either parametric or non-parametric. In parametric methods, the model structure is pre-determined and its parameters are estimated via training data. In other words, the data are summed up in the modeling parameters. Artificial neural networks are typical of parametric approaches. In a majority of neural networks, the model's structure (number of neurons/layers) is assumed to remain fixed, and the modeling parameters are considered as the weights of the connections in the network. In a variation of neural networks, known as polynomial networks, each layer consists of a number of units which are considered as a single polynomial. Here the coefficients of the polynomials represent the network parameters. GMDH can be named here as a common example of polynomial neural networks. Further explanation on GMDH networks is left to the subsequent sections.

Neural networks are among the major categories of a class of computing methods known as soft-computing strategies. Genetic algorithms are another well-known technique in the context of soft-computing. These algorithms can be considered as optimization methods loosely based on the mechanics of artificial selection and recombination operators. By reproducing and combining promising solutions, high-quality partial solutions unite to form newer results. High-quality partial solutions are called building blocks (BBs) [1]. General, fixed, and problem independent recombination operators often break the building blocks or do not mix them efficiently. GA works well only when building blocks of the problem are located tightly in strings representing the solution. On problems with the building blocks spread all over the solutions, the simple GA results in performance degradation [2,4].

This paper introduces a methodology for extracting building blocks of the problem in the context of genetic algorithms. The method relies on the construction of a GMDH neural network model of solutions rather than the application of genetic-type operators i.e. recombination and mutation operators. It is independent of the sequence of the building blocks in strings representing the solution. In this method, a number of solutions with high fitness values are selected in each generation. The selected individuals along with their fitness values are considered as the inputs and outputs of a system. As shown in figure 1, given the input and output for a typical system, its model can be obtained. In the context of our study, the constructed model features the strings with high fitness scale. Throughout the next step, with the extraction of rules from the constructed network, an interpretable knowledge of high fitness strings is accomplished based on which the next generation will be created.

The paper is organized as follows: Section 2 briefly goes over the GMDH neural network. In section 3, the proposed evolutionary-based algorithm is described. Sections 4 and 5 are dedicated to the implementation results and paper conclusions respectively.

2. GMDH Neural Network Overview

This section explains the basic GMDH neural network concepts supposed to be necessary to understand the description of the proposed method.

The group method of data handling (GMDH) was introduced by Ivakhnenko in 1966 as an inductive learning process for complex systems modeling [5]. The GMDH model possesses a forward multi-layer neural network structure. Each layer consists of one or more units with two inputs and one output. Every unit corresponds to Ivakhnenko polynomial forms (1):

$$z = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2$$
$$z = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 + a_4x_1^2 + a_5x_2^2 \qquad (1)$$

It is comprised of two input variables x_1 and x_2, an output variable z, and the coefficients a_i's. Figure 2 illustrates a typical multi-layer GMDH network with 4 inputs, 3 layers and 7 units.

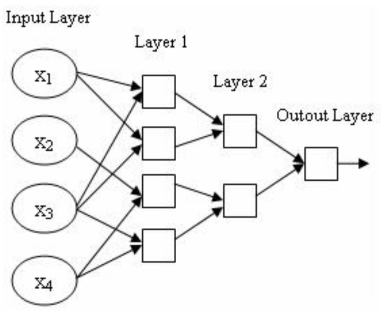

Figure 2. A typical GMDH network with 4 inputs, 3 layers and 7 units

Basically, the GMDH learning algorithm consists of the following steps [6]:

1) Given a learning data sample including a dependent variable y and independent variables x_1, x_2... x_m split the sample into a training set and a checking set.
2) Feed the input data of m variables and generate $(m,2)$ combination units from every two variable pairs at the first layer.
3) Estimate the coefficients of all units in formula (1) using the training set [7].
4) Compute the square error between the real output and the prediction of each unit based on the checking data. For example, the error value for the ith unit can be computed using the following formula:

$$Err_i = \sum_{k=1}^{c}\left(\hat{y}_i(k) - y_i(k)\right)^2 \qquad (2)$$

In (2), $y_i(k)$ and $\hat{y}_i(k)$ are the output of the kth training data and the prediction of ith unit respectively. Also c is the number of training data.

5) Sort out the units by error and eliminate the undesired units.
6) When the minimum error of the current layer becomes larger than that of in the previous layer, remove the current layer and go to step 8.
7) Set the prediction of units in the current layer to new input variables for the next layer, and generate next layer units from every two variable pairs in the current layer. Go to step 3.
8) In this step, choose the unit with minimum error in the last layer as the final output. The ultimate network can be obtained recursively from the path ending to the final output, so that the units with no connection to this output get eliminated.

3. An Evolutionary GMDH-based Algorithm

In this section, a new approach for building block extraction in the context of evolutionary algorithms is proposed. Here, each individual is a binary string to which a fitness value is assigned. Figure 3 illustrates a general schematic of the operation of the algorithm.

The proposed method entails six steps. In step 1, the initial population is generated randomly. In step 2, a set of promising strings (individuals with highest fitness values) are selected. Step 3 (the modeling step) aims at constructing a model from the selected individuals and finding their effective bits. The proposed method uses GMDH neural network for its modeling process. The selected individuals along with their associated fitness values serve as the required input data to construct the network. The individuals are split up into two sets: a training set and a checking set. The polynomial used in the process has the following form:

$$z = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 \qquad (3)$$

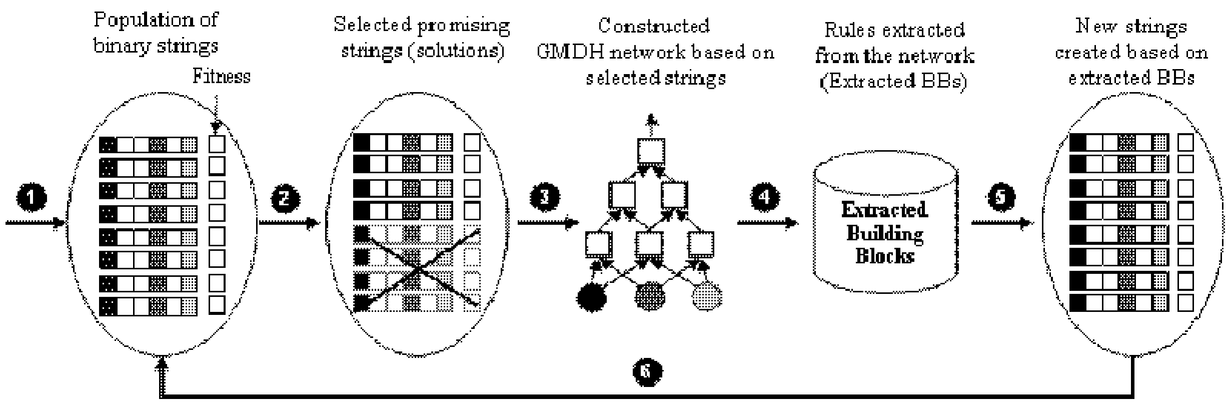
Population of binary strings
Fitness
Selected promising strings (solutions)
Constructed GMDH network based on selected strings
Rules extracted from the network (Extracted BBs)
Extracted Building Blocks
New strings created based on extracted BBs

Figure 3. Schematic of the algorithm

In our implementation, the elimination constraint for undesired units can be defined as the following:

Assume that the number of units in the ith layer is m. After creation of $(m,2)$ units in layer $i+1$, the m*ratio best ones are selected and the others are discarded.

Assume that figure 4 illustrates the final model (GMDH network) of selected individuals. As illustrated in this figure, the output of this network is the estimated fitness value for the selected individuals. Also, the inputs x_i, x_j, and x_k prove to be the effective variables of the selected individuals.

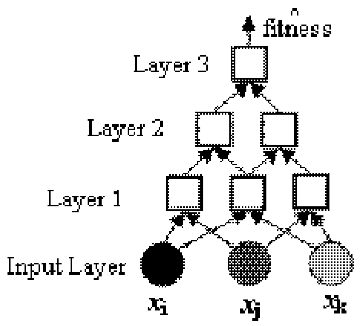

Figure 4. Final GMDH network

3.1. Building Block Extraction

In this step, rules are extracted form the resultant GMDH network of the modeling step. Accordingly, the dependency among variables or specifically, the building blocks of the problem will be discovered. To extract rules from the GMDH network, the extended Fujimoto rule extraction method [6] is employed.

To have an insight of the procedure, consider the network illustrated in figure 4. As shown in the figure, the vector $X = (x_i, x_j, x_k)$ with length L=3 contains the effective variables. All combinations of vector X are indexed as follows:

$$X_1 = (0, 0, 0), X_2 = (0, 0, 1), \ldots , X_{2^L} = (1, 1, 1)$$

Each of these combinations is fed to the GMDH neural network, and its estimated fitness, i.e. $\hat{fitness}$ is computed. Next, these combinations are sorted and re-indexed based on their associated fitness level. The result is presented in table 1.

Table 1. Rule extraction table [6]

X_1	$\hat{fitness}_1$	G_1	
X_2	$\hat{fitness}_2$	G_2	
.	.	.	
.	.	.	
X_k	$\hat{fitness}_k$	G_k	← *max*
.	.	.	
.	.	.	
X_{2^L-1}	$\hat{fitness}_{2^L-1}$	G_{2^L-1}	
X_{2^L}	$\hat{fitness}_{2^L}$	---	

In this table, the following conditions are satisfied:

$$\hat{fitness}_i \geq \hat{fitness}_{i+1} \quad , \quad i = 1..2^L - 1 \qquad (4)$$

The gap between two successive $\hat{fitness}$ is defined as follows:

$$G_i = \hat{fitness}_i - \hat{fitness}_{i+1} \quad , \quad i = 1..2^L - 1 \qquad (5)$$

The maximum gap between two successive $\hat{fitness}$ is defined as G_k:

$$G_k = \max(G_i) \quad , \quad i = 1..2^L - 1 \qquad (6)$$

G_k is defined to find the best set of X_i's for rule extraction. Since the X_i's in Table 1 are sorted in a descending order, the following rule is obtained:

IF X_1 **OR** X_2 **OR** … **OR** X_k **THEN** Fitness is High

To represent this rule by x_i's, the algorithm makes use of the fuzzy karnaugh map which groups X_i's ($i=1..k$) with respect to their $\hat{fitness}$ values. To do so, the $\hat{fitness}$ values are normalized in [0,1] as depicted in formula (6). Hereafter, the normalized values are called *Belief*.

$$Belief_i = \frac{\hat{fitness}}{\max(\hat{fitness})} \quad i = 1..k \qquad (7)$$

Each X_i ($i=1..k$) in Table 1, is associated with a degree of belief. Therefore, the inputs to the fuzzy karnaugh map are the $(X_i, Belief_i)$ pairs. Since the grouping operator in the map is an S-norm type, the fuzzy karnaugh map applies the *max* operator to determine the degree of belief for each group. Figure 5 shows an example of the fuzzy karnaugh map in operation.

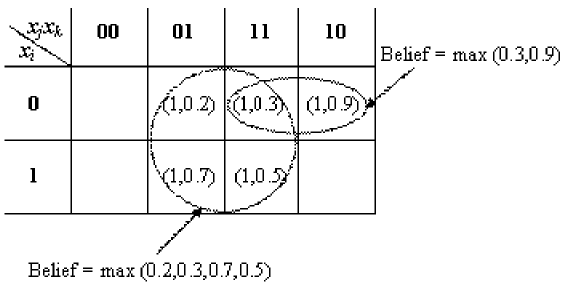

Figure 5. An example of fuzzy karnaugh map with *max* operator

The extracted rules of the sample illustrated in figure 5 are as the following:

Rule 1:
IF $\sim x_i x_j$ **THEN** Fitness is High (Belief = 0.9)
Rule 2:
IF x_k **THEN** Fitness is High (Belief = 0.7)

These rules identify two building blocks of the problem with their associated degree of belief.

Table 2. Extracted Building Blocks (BBs)

Rule Number	Building Blocks	Degree of Belief
1	$\sim x_i x_j$	0.9
2	x_k	0.7

3.2. Building Block Insertion in New Strings

The remaining steps i.e. steps 5 and 6 (see figure 3) are meant to generate new strings with respect to extracted building blocks. New strings are initialized at random. Next, the extracted BBs are applied to them. For instance, the extracted BBs in the previous step (see table 2), are applied to a new string in accordance with their degree of belief.

For instance, the probability of having x_k as 1 in next generation strings is less than 0.7. Likewise, the probability of having x_ix_j as 01 in newly generated strings is less than 0.9.

At this point, the new individuals are added to the current population, replacing some of the old ones. The new population is evaluated and each individual is assigned a fitness value. Unless the termination criteria are met, a new cycle starts over by selecting promising individuals.

4. Experimental Results

This section demonstrates the simulation results and compares the proposed algorithm with simple GA. The experiments are carried out on a variety of different problems with varying degrees of complexity. A few representative results are presented here. The problems chosen are as the following:

- OneMax Test Function, a Unitation Function, in 100 dimensions [3]. (F1 Function)
- DeJong Test Function 2, also called Rosenbrock's Function, in 2 dimensions [8]. (F2 Function)
- DeJong Test Function 3, a Step Function, in 5 dimensions [8]. (F3 Function)
- DeJong Test Function 4, a Quartic Function, in 30 dimensions [8]. (F4 Function)

The characteristics of these benchmarks are listed in Table 2. Also, the length of the solutions for each function is shown in this table.

Table 3. Characteristics of the problem used in the experiments

Test Function	F1	F2	F3	F4
Dimension	100	2	5	30
Type	Max.	Min.	Min.	Min.
Multi/Uni Modal	Uni.	Multi.	Multi.	Multi.
Eval. with Noise	No	No	No	Yes
Solution Length	100	100	250	300
Optimum (m)	100	zero	-30	$0<m<30$

In our experiments, a population of 100 individuals was used. In each iteration the best half of the individuals are selected for the modeling algorithm. The *ratio* parameter was set to 0.45, i.e. the proportion of the number of units in i^{th} layer to the number of units in $(i-1)^{th}$ layer would be 0.45 in the constructed GMDH network.

Three stopping conditions were considered. First, when a fixed number of generations are accomplished. A further stopping condition occurs when the algorithm finds a solution with the objective value about 95% of the optimum (in case, it is known in advance). Finally, the algorithm stops when the best value of the population doesn't improve within a fixed number of iterations.

Table 4 summarizes the results obtained by applying GMDH-based algorithm to each function. All results are averaged over 40 runs.

Table 4. Results obtained by the algorithm

Test Function	F1	F2	F3	F4
Mean Values	84,66	1.9E-5	-26.53	6.43
Best Value	96	2.6E-6	-28	0.13
Worst Value	80	4.3E-3	-24	11.96
Standard Deviation	5.03	1.4e-3	1.7	3.2
Max Generation	500	300	300	500

As shown in table 4, while the best results are gained from F2 (DeJong 2), F1 leads to the worst results (OneMax). This is due to the fact that OneMax has no significant variable and all of its bits are of equal importance. Since our method tries to identify the effective variables in each iteration, the algorithm finds it difficult to optimize.

Figure 6 illustrates the effect of population size on the number of generations required for DeJong Function 4 optimization. No matter whether loose building blocks (the building blocks spread all over the strings representing the solution) or tight BBs (the building blocks located tightly in the string) is used, the algorithm behaves similarly. Whereas, the simple GA results in different behaviors for loose and tight building blocks. As discussed above, the proposed algorithm is independent of the ordering of the variables in the strings representing the solution. GMDH-based algorithm outperforms the GA in both loose and tight building blocks operating on a large population size. In our experiments, the simple GA with one point crossover and truncation selection was used. The crossover and mutation rates were set to 0.9 and 0.05.

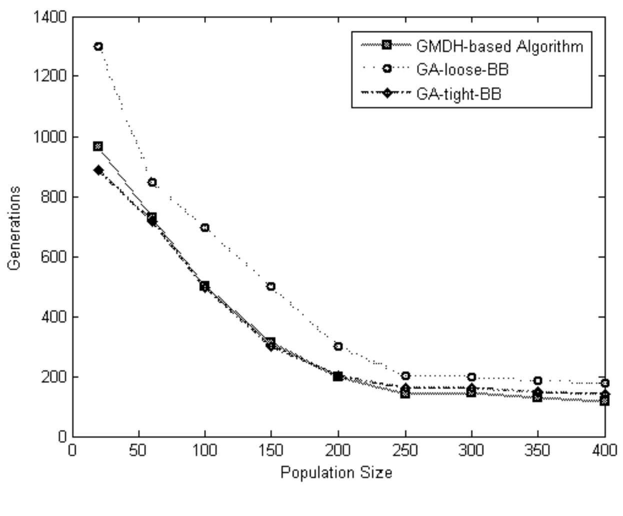

Figure 6. Average number of generations required for optimizing DeJong Function 4

5. Conclusions

This paper introduced a methodology for extracting building blocks in the context of evolutionary algorithms with binary strings. It is based on the rule extraction from the constructed GMDH network of the selected strings for a given population. The experimental work entails a comparative study of the proposed method with simple genetic algorithm. The problems over which the experiments were carried out were of type static function optimization. These problems were adopted on account of their frequent appearance in the GA literature. The experiments have shown that the proposed algorithm outperforms the simple GA in problems with loose building blocks.

6. References

[1] Goldberg, D.E., "Genetic Algorithms in Search, Optimization and machine Learning", Addison Wesley: Reading, MA, 1989

[2] Thierens, D., "Analysis and design of genetic algorithms", Doctorial dissertation, Leuven, Belgium, 1995

[3] Pelikan, M., Goldberg, D.E., Cantú-Paz, E. "BOA: The Bayesian optimization algorithm", In Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, Vol. 1, Morgan Kaufmann Publisher: San Francisco, CA, pp 525-532, 1999

[4] Harik, G.R., Goldberg, D.E., "Learning Linkage", Foundation of Genetic Algorithms, Vol. 4, pp 247-262

[5] Ivakhnenko, A.G., "The Group Method of Data Handling – A Rival of the Method of Stochastic Approximation", Soviet Automatic Control, Vol. 13, No.3, pp. 43-55, 1966

[6] Fujimoto, K., Nakabayashi, S., "Applying GMDH Algorithm to Extract Rules from Examples", Systems Analysis Modeling Simulation, Vol. 43, No. 10, pp 1311-1319, October 2003

[7] Howland, J.C., Voss, M.S., "Natural Gas Prediction Using The Group Method of Data Handling", proceedings of 7th IASTED International Conference and Soft Computing, Banff, Alberta, Canada, July 2003

[8] Digalakism, J.G., Margaritis, K.G., "An Experimental Study of Benchmarking Functions for Genetic Algorithms", International Journal of Computer Mathematics, Vol. 7, pp 403-416, April 2002