
Rule-based Optimization by Learning Neural Network Model of the Promising Solutions

Ehsan Nazerfard, Saeed Bagheri Shouraki

Artificial Creatures Lab
Computer Engineering Department
Sharif University of Technology
Tehran, Iran
nazerfard@ce.sharif.edu, bagheri-s@sharif.edu

Abstract

In this paper an evolutionary algorithm is proposed which makes use of the neural network model of promising solutions in its generation of new results. It is called the Rule-based GMDH Evolutionary Algorithm (RGEA). RGEA relies on Group Method of Data Handling (GMDH) neural network to construct a model of promising solutions and uses extracted rules from the constructed neural network to identify building blocks of the problem. Basically the paper is an attempt to overcome the well-known limitations of the Genetic Algorithms to optimize problems with tight linkage.

1 INTRODUCTION

Search strategies can be classified as either complete or heuristic. The underlying idea in the complete search is the systematic examination of all possible points of the search space. Heuristic algorithms themselves take on the form of either a deterministic or a non-deterministic search scheme. A deterministic search always results in identical solutions under the same situation, whereas a non-deterministic search is characterized by taking randomness as a key element to escape from local optimums. Due to its stochastic nature a non-deterministic search may lead to different results under the same situation. While some of the stochastic heuristic schemes (e.g. simulated annealing) store only one solution in each iteration of the algorithm, the other strategies are based on a population of candidate solutions. The population evolves as the algorithm proceeds toward more promising zones. In evolutionary algorithms, the search in the space of solutions is carried out by means of a population of individuals, i.e. possible solutions to the problem. Evolutionary algorithms require that appropriate crossover and mutation operators be designed in order to generate new individuals for the next population. Aside

from the selection of the operators themselves, several parameters such as the crossover and mutation rate or the population size affect the behavior of the stated algorithms. The manner in which individuals are represented also is also important because depending on this and on the mentioned operators, the algorithm will consider the interrelation between the different pieces of information used to represent the individuals implicitly. One attempt to design evolutionary algorithms that removes the need to define problem-specific crossover and mutation operators, while being able to consider the interrelations between the variables representing the individuals, is called Estimation of Distribution Algorithms (EDA). These algorithms explicitly model the good solutions found so far and use the constructed model to generate new population [Baluja 1994, Mühlenbein, Paaß 1996, Pelikan, Goldberg, and Cantú-Paz 1999, Larrañaga, Etxeberria, Lozano, and Peña 2000].

This paper is intended to introduce an algorithm that relies on GMDH neural network to model high-quality solutions found so far, and to apply the extracted rules from constructed network to generate new candidate solutions. The rest of the paper is organized as follows: In section II, the related work and the motivation of the proposed algorithm is presented. Section III lays out a review on the basics of GMDH neural networks followed by an introduction to RGEA. In Section IV the results of the experiments are demonstrated. The paper ends with conclusions.

2 MOTIVATION AND RELATED WORK

Genetic algorithms are 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) [Goldberg 1989]. The genetic algorithm implicitly manipulates a large number of building blocks through the mechanisms of selection and recombination. General, fixed, and problem independent recombination operators often break the

building block 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. The problem of building block disruption through recombination operators is referred to as the linkage problem [Harik and Goldberg 1996]. Several attempts have been made to prevent the disruption of important building blocks. The first class of methods is based on changing the representation of solutions in the algorithm or evolving the recombination among individual solutions. The goal of these methods is to make the interacting components of partial solutions less likely to be broken by recombination operators [Pelikan 2000]. The second class of methods is based on extracting some information from the entire set of promising solutions in order to generate new results. These methods change the principles of recombination to cope with the disruption of partial solutions (linkage problem). In these methods, a model of good solutions is constructed and the reproduction and the mixture of the building blocks are performed based on the resultant model. This line of search led to the proposition of a class of evolutionary algorithms that probabilistically model promising solutions to guide the further exploration of the search space rather than using crossover and mutation as in simple GA. A general framework of the algorithm based on this principle is called the Probabilistic Model-building Genetic Algorithm (PMBGA) or Estimation of Distribution Algorithm (EDA) [Pelikan, Goldberg, and Lobo 2000].

The EDA, therefore, functions in a similar way to the simple GA except that it replaces genetic recombination and mutation operators by the following two steps:

- 1) A model is constructed that is an estimate of the distribution of selected promising solutions.
- 2) New solutions are generated according to the constructed model.

The algorithm proposed in this paper is motivated by the EDA framework.

3 GMDH NEURAL NETWORK OVERVIEW

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

The group method of data handling (GMDH) was introduced by Ivakhnenko in 1966 as an inductive learning process for complex systems modeling [Ivakhnenko 1996]. The GMDH model has a forward multi-layer neural network structure. Each layer consists of one or more units with two inputs and one output. Every unit follows Ivakhnenko polynomial forms (1):

$$z = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 \quad (1)$$

$$z = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 + a_4x_1^2 + a_5x_2^2$$

It consists of two input variables x_1 and x_2 , an output variable z , and the coefficients a_i 's. Figure 1 illustrates a typical multi-layer GMDH network with 4 inputs, 3 layers and 7 units.

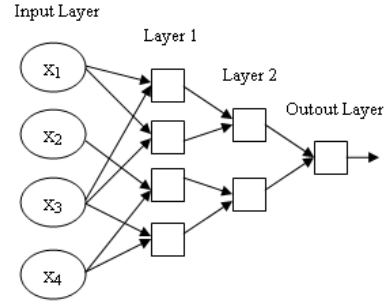


Figure 1: A typical GMDH network with 4 inputs, 3 layers and 7 units

Basically, the GMDH learning algorithm is a self-organizing method which consists of the following steps [Fujimoto and Nakabayashi 2003]:

- 1) Given a learning data sample including a dependent variable y and independent variables x_1, x_2, \dots, x_n ; split the sample randomly into a training set and a test set.
- 2) Feed the input data of n variables and generate $\binom{n}{2}$ combination units from every two variable pairs at the first layer¹.
- 3) Estimate the coefficients of all units in formula (1) using the training set [Howland, Voss 2003].
- 4) Compute the square error between y and the prediction of each unit using test set. For example, the error value for the i^{th} unit can be computed using the following formula:

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

In (2), $y(k)$ is the output of the k^{th} test data; while $\hat{y}_i(k)$ is the predicted output of the i^{th} unit using k^{th} test data. Also c is the number of test data in test set.

- 5) Sort out the units by error and eliminate the

¹The number of combinations of a set of n objects taken r at a time is given by $\binom{n}{r} = \frac{n!}{r!(n-r)!}$

Figure 2: RGEA general schematic

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.

GMDH presents a self-organizing learning algorithm, i.e. the network structure does not need to be determined in advance. This method is also capable of determining the effective variables of the system. Finally, the relation between the inputs and the output of the system is computed in the polynomial form as depicted in (3):

$$y = a_0 + \sum_{i=1}^M a_i x_i + \sum_{i=1}^M \sum_{j=1}^M a_{ij} x_i x_j + \sum_{i=1}^M \sum_{j=1}^M \sum_{k=1}^M a_{ijk} x_i x_j x_k \dots \quad (3)$$

$$X = (x_1, x_2, \dots, x_M) \quad , \quad A = (a_1, a_2, \dots, a_M)$$

The capitalized 'X' is the effective variables vector; likewise, A is the coefficients vector.

The GMDH learning algorithm will be represented by an example in section 4.2 (see figure 3).

4 THE RULE-BASED GMDH EVOLUTIONARY ALGORITHM

In this section, the new Rule-based GMDH Evolutionary Algorithm, RGEA, is described. The algorithm explicitly models the selected individuals of the population. Here, each individual is a binary string with a fitness value in its possession. Figure 2 illustrates a general schematic of the operation of the algorithm.

In RGEA, the initial population is generated randomly. In each iteration, the next population is created according to

Figure 3: GMDH model construction steps

the extracted rules from the selected individual solutions. The RGEA algorithm as depicted in Figure 2 is composed of four major components:

- 1) Selection
- 2) Modeling
- 3) Rule-Extraction
- 4) Next Population Generation

These components are further described in the following subsections.

4-1 Selection

The selection phase of the algorithm is intended to choose a set of promising strings (high fitness individuals). To serve this purpose, any selection algorithm can be adopted. During this phase, a subset of high-fitness individuals is chosen as the input data for the modeling algorithm (see figure 2).

4-2 Modeling Algorithm

The modeling phase aims at constructing a model from selected individuals and finding their effective variables (bits). RGEA uses GMDH neural network for its modeling process. Selected individuals and their associated fitness values serving as the required input data to construct the network are split randomly into two sets: a training set and a test set (see figure 2). The polynomial used in the process has the following form:

$$z = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2$$

Figure 3 illustrates how a GMDH network is constructed step by step from a 5-dimensional data set. Figure 4 illustrates the final GMDH model of the 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_1 , x_2 and x_4 prove to be the effective variables of the selected individuals.

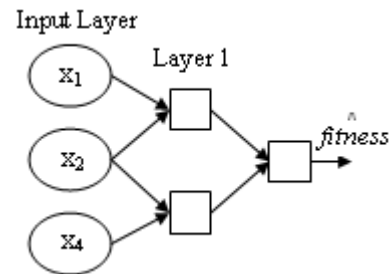


Figure 4: Sample final GMDH network

4-3 Rule Extraction

In this phase, rules are extracted from 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 is employed [Fujimoto and Nakabayashi 2003].

To give an insight of the procedure, consider the network illustrated in Figure 4. As shown in the network, the vector $X = (x_1, x_2, x_4)$ with length $L=3$ contains the effective variables. All combinations of vector X are indexed as follows.

$$X_1 = (0, 0, 0), X_2 = (0, 0, 1), \dots, X_2^L = (1, 1, 1)$$

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 $\hat{fitness}$. The result is represented in Table 1.

Table 1: Rule extraction table

X_1	$\hat{fitness}_1$	G_1
X_2	$\hat{fitness}_2$	G_2
\vdots	\vdots	\vdots
X_k	$\hat{fitness}_k$	$G_k \xleftarrow{max}$
\vdots	\vdots	\vdots
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:

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

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

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

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

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

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, RGEA uses the fuzzy karnaugh map which groups X_i 's ($i=1..k$) with respect to their corresponding $\hat{fitness}$ values. To do so, the $\hat{fitness}$ values are normalized in $[0,1]$ as depicted in formula (7). Hereafter, the normalized values are called *Belief*.

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

Each X_i ($i=1..k$) in Table 1, is associated with a degree of belief. As shown in Figure 4, the fuzzy membership function for high fitness is defined over X_i 's.

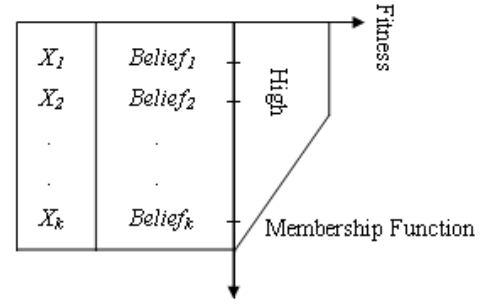


Figure 5: Fuzzy view of Table 1

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 group². Figure 6 represents an example of the fuzzy karnaugh map in operation.

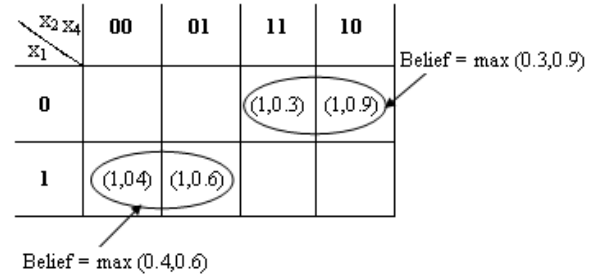


Figure 6: An example of fuzzy karnaugh map with *max* operator

The extracted rules of the sample illustrated in Figure 6 are as the following:

Rule 1: IF $\sim x_1 x_2$ THEN Fitness is High (Belief = 0.9)

Rule 2: IF $x_1 \sim x_2$ THEN Fitness is High (Belief = 0.6)

These rules identify two building blocks of the problem with their associated degree of belief (also see figure 2). The two extracted rules reflect a kind of XOR relation between x_1 and x_2 .

4-4 Generating New Strings

This section outlines the final stage in the RGEA algorithm: the generation of new strings based on the extracted rules. First, new strings are initialized at random. Next, the extracted rules are applied to them. For this purpose, the rules are sorted according to their degree of belief. The higher the degree of belief, the higher the chance for being applied later. This way, whenever there is a conflict between the extracted rules, only the ones

² Other S-norm operators may be used.

with higher degree of belief may cancel out the effect of those with a lower belief level. For instance, the extracted rules in the previous section are applied to a new string as shown below:

Rule 1:

IF $x_1 \sim x_2$ THEN Fitness is High $\rightarrow \text{Prob}(x_1 x_2 = \mathbf{10}) < 0.6$

Rule 2:

IF $\sim x_1 x_2$ THEN Fitness is High $\rightarrow \text{Prob}(x_1 x_2 = \mathbf{01}) < 0.9$

Consequently, this algorithm may insert identified building blocks separately into a new string, mix them or leave the new string intact. This procedure sets up a balance between the exploration and the exploitation of the search space. The algorithm realizing the production of new strings can be summarized as below:

- 1) Initialize new strings randomly.
- 2) For each new strings perform steps (3) and (4):
- 3) Using a selection method, identify an order to apply the extracted rules so that the rules with higher degree of belief get more chance for a later application.
- 4) Insert each of extracted building blocks into the new strings according to its associated degree of belief.

New Individuals are added to the current population, replacing some of the old ones. In this step, 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 good individuals (see fig. 2).

5 EXPERIMENTAL RESULTS

This section demonstrates the simulation results and compares RGEA with simple GA, both in terms of the number of generations and the time required to get optimum solutions.

RGEA was applied to a variety of different problems with varying degree of complexity. A few representative results are presented here. The problems chosen are [Digalakism and Margaritis 2003]:

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

The characteristics of these benchmarks are listed in

Table 2. Also, the length of the solution associated with each function is also shown in this table.

Table 2: 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.

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 3 summarizes the results obtained by applying RGEA to each function. All results are averaged over 40 runs.

Table 3: Results obtained by the algorithm to the functions

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

Figure 7 illustrates the effect of population size on the number of generations required for DeJong Function 4. 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, RGEA 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. RGEA 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.

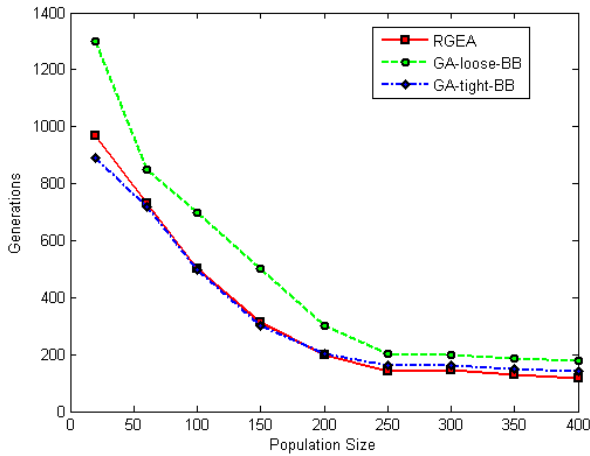


Figure 7. Average number of generations required for optimizing DeJong Function 4

Figure 8, illustrates the effect of population size on the time required for optimizing DeJong Function 4. As depicted in this figure, RGEA needs more time to find optimum solutions, it can be attributed to the fact that the GMDH model construction and rule-extraction are time consuming while genetic operators are very fast.

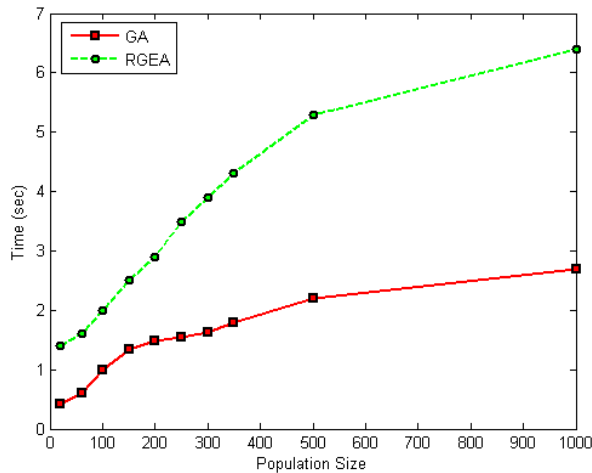


Figure 8. Average time (sec) required for optimizing DeJong Func. 4

6 CONCLUSION

This paper presented the RGEA, an evolutionary algorithm that finds the building blocks of the problem by extracting rules of the selected individuals of the population. In this paper the results are compared with simple genetic algorithm. The problems used here were static function optimization. They 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.

REFERENCES

- P. Larrañaga, R. Etxeberria, J.A. Lozano, and J.M. Peña (2000), "Combinatorial optimization by learning and simulation of Bayesian networks", In Proceedings of Sixteenth Conference on Uncertainty in Artificial Intelligence, Stanford, pp. 343-352
- D. E. Goldberg (1989), "Genetic Algorithms in Search, Optimization and machine Learning", Addison Wesley: Reading, MA
- M. Pelikan, D.E. Goldberg, and E. Cantú-Paz (1999), "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
- G.R. Harik, D.E. Goldberg (1996), "Learning Linkage", Foundation of Genetic Algorithms, Vol. 4, pp. 247-262
- H. Mühlenbein, G. Paaß (1996), "From recombination of genes to the estimation of distributions. binary parameters", In Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature, PPSN IV, pp. 178-187, Springer
- S. Baluja (1994), "Population Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning", Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, Pennsylvania
- M. Pelikan, D.E. Goldberg, and F. Lobo (2000), "A Survey of Optimization by Building and Using Probabilistic Models", University of Illinois
- A.G. Ivakhnenko (1996), "The Group Method of Data Handling - A Rival of the Method of Stochastic Approximation", Soviet Automatic Control, Vol. 13, No.3, pp. 43-55
- K. Fujimoto, S. Nakabayashi (2003), "Applying GMDH Algorithm to Extract Rules from Examples", Systems Analysis Modeling Simulation, Vol. 43, No. 10, pp. 1311-1319
- J.C. Howland, M.S. Voss (2003), "Natural Gas Prediction Using The Group Method of Data Handling", proceedings of 7th IASTED International Conference on Artificial Intelligence and Soft Computing, Banff, Alberta, Canada
- J.G. Digalakis., K.G. Margaritis (2002), "An Experimental Study of Benchmarking Functions for Genetic Algorithms", International Journal of Computer Mathematics, Vol. 7, pp. 403-416