BULETINUL INSTITUTULUI POLITEHNIC DIN IAŞI
Publicat de
Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Volumul 62 (66), Numărul 4, 2016
Secția
ELECTROTEHNICĂ. ENERGETICĂ. ELECTRONICĂ

MATLAB-BASED TEST BENCH FOR GENETIC ALGORITHM PARAMETER TUNING

BY

ADRIANA SÎRBU[*] and IOLANDA-ELENA ALECSANDRESCU

Technical University “Gheorghe Asachi” of Iaşi,
Faculty of Electronics, Telecommunications and Information Technology

Received: November 16, 2016
Accepted for publication: December 5, 2016

Abstract. The proliferation of evolutionary computation used to solve various technical problems requires increased performances, especially when real-time implementation is necessary. The paper proposes a MATLAB based test bench providing the essential tools for fine tuning the parameters of genetic algorithms. The capabilities of the designed platform are tested using an NP-hard problem: clustering algorithms for Wireless Sensor Networks, which uses genetic algorithms. The experimental results point out the importance of choosing the right parameters for a given application. Using the designed test bench, we can so report improvements of the algorithm behavior both from the obtained optimum result point of view and the convergence speed.

Key words: genetic algorithm; parameter tuning; algorithm performance; experimental methodology.

1. Introduction

Genetic algorithms (GAs) represent a robust and flexible heuristic optimization technique inspired by natural evolution. They are particularly

[*]Corresponding author: *e-mail*: asirbu@etti.tuiasi.ro

suited to problems where traditional optimization techniques break down, either due to the particular structure of the search space (for example, absence of gradient information) or because the search operation is computationally infeasible.

Recently the field of GAs as a means to find good solutions to problems that were otherwise computationally difficult has drawn increasing attention. Their possible application to a wide range of practical problems in science, engineering and industry raised several practical issues not anticipated by earlier theory (McCall, 2005).

It has been noticed though that, in spite of the large amount of research in the field (Doughabadi *et al.*, 2011; Eiben *et al.*, 2011; Eiben *et al*, 2012; Moraes Barbosa *et al.*, 2015) fine tuning of the parameters of a GA is a matter of testing as each application has a different behavior and is sensitive to particular parameters.

This is the reason why a test bench for off-line fine tuning of the parameters of a GA dedicated to a special problem is absolutely necessary (Gunawan *et al.*, 2011; Akbaripour *et al.*, 2013). In the present paper we propose a MATLAB-based (MathWorks, 2017) solution for this subject.

We organize our paper as follows. In section 2, we present an overview of the genetic algorithms and the problematic of parameter tuning. In section 3, we describe the MATLAB implementation of the proposed test bench platform. In section 4, we illustrate the advantages of using adequate GA parameter tuning with a case study envisaging minimizing the communication distance in wireless sensor networks (WSN) (Al-Obaidy et al., 2015). Finally we will conclude or paper in Section 5, underlining further research directions.

2. Genetic Algorithms and Parameter Tuning

Genetic algorithms were devised to mimic the evolutionary behavior of biological systems. The approach was based on the remark that the genetic code of an individual defines his ability to survive and breed, usually referred as its *fitness*, and, this way, to transmit his code to succeeding generations. The objective of the species can be viewed in this perspective as a search for an optimum code for a specific environment. Modifications of the code appear during reproduction or due to mutation. All these demand setting up appropriate genetic operators (crossover, mutation etc) to drive a set of solutions to an optimum.

While a GA evolves, it progresses through a series of populations, which succeed as so called generations, *g*. Each population P*g* builds upon the previous population, P*g*-1 in order to obtain progressively more fit individuals. This is accomplished by selecting individuals from P*g*-1 and replacing them according to some criteria which take into consideration the fitness of individuals. This process is repeated while some termination criteria are not yet

achieved (such as a specific number of generations having passed or an individual with fitness above a certain threshold being found).

The pseudocode description for a typical GA is given in Fig. 1 (Kramer, 2017).

```
begin
        initialize g to 0
        initialize members of P0 to random values
        evaluate fitness of members of P0
        while (termination condition not reached) loop
                increment g
                select parents from Pg-1 to compose Pg
                perform crossover on parents of Pg
                perform mutation on population Pg
                evaluate fitness of members of Pg
        end loop
end
```

Fig. 1 − Genetic algorithm structure.

Parameter tuning can be considered as a special case of algorithm design. As shown in Eiben *et al.*, (2012), in recent years, an increasing number of publications were devoted to the case studies on parameter tuning. The outcome of tuning varies from increasing algorithm performance to decreasing the computational effort.

One of the main challenges for GA designers is that the parameter values influence to a large extend the performance of the algorithm. A good parameter values choice can improve by significant orders of magnitude the behavior of the algorithm.

Literature (Eiben *et al.*, 2012) mentions two relevant performance measures for evolutionary algorithms in general: one regarding solution quality and one regarding algorithm speed or search effort (referring to any measure of computational effort such as the number of fitness evaluations or CPU time).

There are different combinations of fitness and time that can be used to define algorithm performance in one single run. For instance (Eiben *et al.*, 2012):

a) Given a maximum execution time (computational effort), algorithm performance is defined as the best fitness at termination.

b) Given a minimum/maximum fitness level, algorithm performance is defined as the execution time (computational effort) needed to reach it.

c) Given a maximum execution time (computational effort) and a minimum/maximum fitness level, algorithm performance is defined through the

Boolean notion of success: a run succeeds if the given fitness is reached within the given time; otherwise it fails.

The parameters which influence the performances of a GA can be split into two categories (Eiben *et al.*, 2007; Eiben *et al.*, 2011; Eiben *et al.*, 2012): qualitative parameters and quantitative parameters. The qualitative parameters envisage the representation mode (binary-coded, real-coded etc.), the type of crossover (one-point, two-point, uniform, arithmetic, heuristic etc.), the type of mutation (bit-flip boundary, non-uniform, uniform, Gaussian etc.), the parent selection method (tournament, Roulette Wheel Selection, Stochastic Universal Sampling, Tournament Selection etc.). Among the quantitative parameters one can enumerate: mutation rate (*pm*), crossover rate (*pc*), population size.

For example the problem of how to choose an adequate population size deals with two antagonistic facts (Sun, 2011): if the population is too small, it is not likely that the GA will find a good solution for the problem at hand. On the contrary if the population is too large, the GA usually takes a long time in processing solutions. There are articles (Haupt, 2000) reporting that the choice of population size and mutation rate can cause the run time of the GA to vary by several orders of magnitude. The results of this investigation show that a small population size and relatively large mutation rate can be far superior to the combination large population sizes and low mutation rates that is used by most of the papers. In (Sastry *et al*., 2000) the author affirms that the convergence time was found to be independent of the population size provided that it was above some threshold value.

Even though many studies have reported indications of how to choose the adequate parameters for GA (Haupt, 2000; Ridge *et al*., 2006; Witt, 2008; Sun, 2011; Rajakumara *et al.*, 2013; Moraes Barbosa *et al*., 2015), there is no general recipe valid for any kind of application using GA. In fact, one of the common conclusion of these studies can be resumed as follows: "Do tune your evolutionary algorithm with a tuner algorithm" (Eiben & Smit, 2012).

3. MATLAB Implementation of GA Test Bench

In order to increase the efficiency of the GA parameter tuning, we have devised a MATLAB-based platform devoted to perform as a test bench for the set-up evaluation.

The Genetic Algorithm Toolbox (MathWorks, 2017) is a collection of routines, written mostly as m-files, which provides a set of versatile tools for implementing a wide range of GA methods and operators.

The main components of this platform are:

a) Population representation and initialization – the population individuals are represented as binary strings and they can be initialised with random values from {0,1} *n*-space.

b) Population evaluation – a fitness value can be calculated as a measure of the quality of each individual. The fitness scaling adjusts the fitness

values before the selection of the parents for the next generation. This scaling affects the diversity of the population.

c) Selection – better solutions have higher chance to be selected as parents for the next generation solutions. There are different selection methods:

c_1) Uniform – the individuals are randomly selected from a uniform distribution and they are used for parents selection.

c_2) Roulette – this method simulate a roulette wheel technique: assign each individual part of the wheel and then spin wheel (use a random number) to select the individual.

c_3) Tournament – select randomly several individuals from the population and then the best one of them is selected to be a parent.

d) Reproduction – controls how the next generation is created. The amount of elitism and the fraction of the next generation are specified:

d_1) Elite count specifies the number of individuals in the current generation that are guaranteed to survive to the next generation.

d_2) Crossover fraction specifies the fraction of individuals in the next generation that are created by crossover, except the elite children. This value is between 0 and 1. A crossover fraction of 0 means that all the children are mutation children. A crossover fraction of 1 means that all the children except the elite individuals are crossover children. The default value for crossover fraction is 0.8.

d_3) Mutation – decides if an individual should be mutated or not, based on the mutation rate. For bit string population type, the mutation function is uniform, flipping bits randomly with a uniform distribution along the string.

e) Crossover – recombine pairs of individuals with given probability to produce offspring. The following methods are implemented in MATLAB:

e_1) Single point: a crossover point is randomly set. The binary string from the beginning to the crossover point is copied from the first parent in the first offspring and then the others are copied from the second parent. The second offspring is created in a similar way, starting from the second parent.

e_2) Two point: in this case two crossover points are randomly set. The binary string from the beginning to the first crossover point is copied in the first offspring from the first parent, the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent again.

e_3) Scattered: creates a random binary vector and the selects from the first parent the bits where the vector is a 1, and from the second parent selects the bits where the vector is a 0. Then combine the bits to form the first child.

f) Stopping criteria – determines what causes the algorithm to terminate. It could be related to the number of generations, to the time limit, to the best fitness value or to other characteristics of the population, *e.g.* diversity.

The developed MATLAB-based platform allow us to perform a variety of tests using the command line to control the parameters as: population size, crossover and mutation probabilities, crossover and mutation operators, stopping condition.

A display routine has been implemented in order to plot the results in each generation and for different values of the controlled parameters given as input in the command line. In this way, the experimental results can show how the fitness value can be improved by using a various set of solutions and how the selected parameters influence the performance of the system.

Users can set their own batch tests varying simultaneously both qualitative (sequentially choosing different genetic operators) and quantitative parameters (stepwise modifying values for mutation rate, crossover probability and so on).

4. Case Study: Tuning GA Parameters for WSN Communication Distance Optimization

In order to demonstrate the effectiveness of the designed test bench we have chosen a NP-hard problem envisaging the clustering of WSNs (Ahmed, 2015). It has been investigated and proved (Shrestha *et al.*, 2007) that grouping sensors into clusters and designating a so called cluster-head to take over the transmission of data from its assigned sensor nodes to the sink increases the life-time of the network by saving energy.

Given an arbitrary sensor node distribution, we use a GA to optimize the number of clusters and sensor connections (clusters structure). The output of the optimization process will be the number of clusters and the structure of each cluster. Previous approaches for clustering have considered the number of clusters as *a priori* known.

For the selected case study, an individual is encoded as an *n*-bit string, where *n* represents the number of sensors. A bit of 1 denotes a cluster-head, while a bit of zero denotes a regular sensor. Each regular node is connected to a cluster-head.

The initial population consists of randomly generated individuals.

In order to generate a cluster, for each regular node, a deterministic method based on finding its nearest cluster-head is used. It was proved in (Al-Obaidy *et al.*, 2015) that the energy consumption in a WSN is minimized if the communication distance in minimized; the shorter the transmission distance, or the lower the number of cluster-heads, the higher the fitness value of an individual is.

In the context of the GA approach, we will use the term *Fitness* to measure the adequacy of a certain sensor distribution. So:

$$Fitness = w * (D - distance_i) + (1 - w) * (n - H_i), \quad (1)$$

where: D is the total distance of all nodes to the sink, $distance_i$ – the sum of the distances from regular nodes to their respective cluster-heads plus the sum of the distances from all cluster-heads to the sink, while n is the total number of nodes. In order to evaluate the contribution of each factor, distances or number of cluster-heads, to the energy consumption, a predefined weight factor, w, is introduced.

For the illustration of the advantages offered by the proposed test bench, we have considered a random distribution of 100 sensor nodes and the associated sink deployed on an area of 7,000 × 7,000 units.

We evaluate the total communication distance defined as the sum of distances between each node and the sink, for the case of non-clustered networks and, respectively, as the sum of distances between each cluster head and the sink plus the sum of distances between a cluster head and each of its assigned sensor nodes.

The overall behavior of the GA-based clustering algorithm for a WSN as compared with the unclustered behavior is presented in Fig. 2.

We present the results obtained for a population of 500 individuals, crossover probability 0.8, mutation probability, scattered crossover and Stochastic Uniform selection.

In Fig. 2 *a* we present the network structure without clustering (Total_communication_distance = 4.2595e + 05 units) and in Fig. 2 *b* the clusters structure obtained using a GA (Total_communication_distance_Cl = = 1.0932e + 05 units). One can easily notice the drastic reduction of the communication distance when using clusterization.

For the sensor deployment in Fig. 2 we have performed different tuning experiments using the designed test bench.

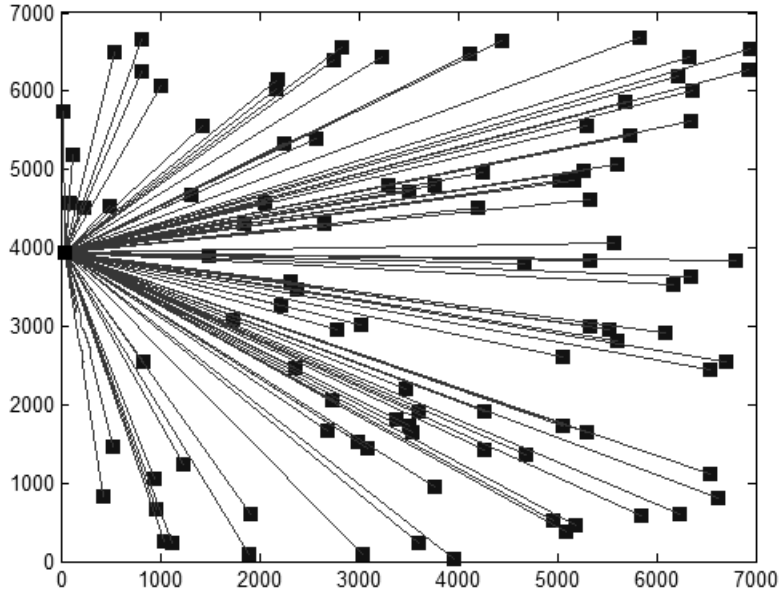
In this study, one-point, two-point and scattered crossovers are tested as crossover operators. It is to be noticed that, as expected, two-point crossover and scattered crossover represent an enhancement over one-point crossover in terms of supplying more diversity in the population.

In Table 1 we present the influence of the population size in conjunction with different crossover operators. We have used Stochastic Uniform selection method, mutation probability 0.01 and crossover probability 0.8.

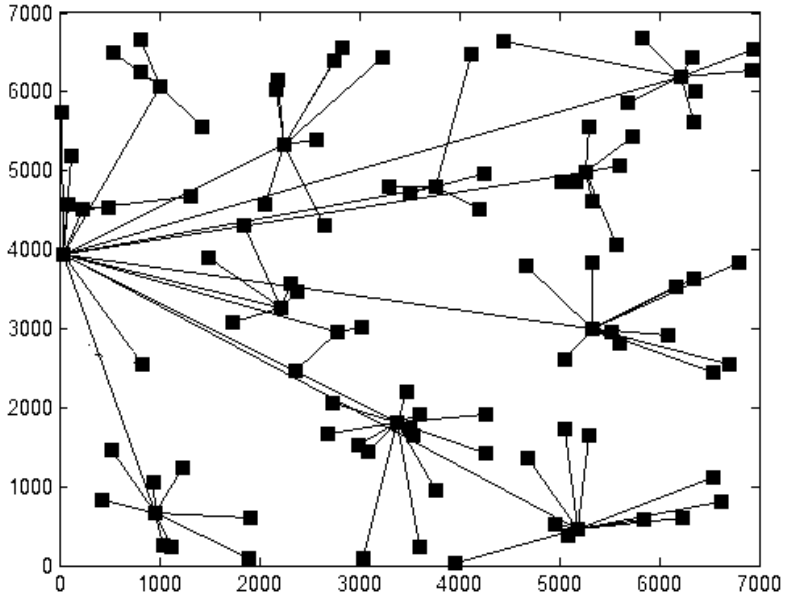
Table 1

Influence of the Population Size for Different Crossover Operators

Population_size	Crossover	Total_comm_distance_Cl	No_of_gen
100	One-point	1.6201 e+05	70
100	Two-point	1.5473 e+05	66
100	Scattered	1.4264 e+05	59
500	One-point	1.2097 e+05	60
500	Two-point	1.1097 e+05	59
500	Scattered	1.0932 e+05	55
1,000	One-point	1.3401 e+05	61
1,000	Two-point	1.2373 e+05	60
1,000	Scattered	1.1164 e+05	55



a



b

Fig. 2 – *a* – WSN without clustering; *b* – WSN with clustering.

It is to be noticed that for the small values of the population size (100 represents the number of sensor nodes), the algorithm has medium performances as compared with the 500 case. Moreover, as predicted, there is a threshold over which increasing the population size does not improve the algorithm behaviour, as illustrated when the value 1,000 is used. That observation has major impact when real time implementation is envisaged – limiting the search space speeds up the runtime duration.

Another interesting aspect is that increasing the mutation probability, and so the diversity, we can obtain good results even from small population sizes. We illustrate this behaviour in Table 2, where the crossover probability is considered constant, 0.8, and scattered crossover operator and Stochastic Uniform were used as selection method.

An important qualitative parameter is also the selection method. In Table 3 we present the result obtained for a population of 500 individuals, crossover probability 0.8, mutation probability and Stochastic Uniform, respectively Tournament selection.

Table 2
Influence of the Population Size and Probability of Mutation for Different Crossover Operators

Population_size	Crossover	Mutation Probability	Total_comm_distance_ Cl	No_of_ gen
100	One-point	0.05	1.5711 e+05	75
100	Two-point	0.05	1.3343 e+05	69
100	Scattered	0.05	1.2374 e+05	65
500	One-point	0.01	1.2097 e+05	60
500	Two-point	0.01	1.1097 e+05	59
500	Scattered	0.01	1.0932 e+05	55

Table 3
Influence of the Selection Method for Different Crossover Operators

Selection method	Crossover	Total_comm_distance_Cl	No_of_gen
Stochastic Uniform	Single-point	1.2097 e+05	60
Stochastic Uniform	Two-point	1.1097 e+05	59
Stochastic Uniform	Scattered	1.0932 e+05	55
Tournament	Single-point	1.3401 e+05	70
Tournament	Two-point	1.2373 e+05	66
Tournament	Scattered	1.1164 e+05	59

Using the proposed fine-tuning platform and so being able to select the right values for the GA we have succeeded to outperform the results reported in (Al-Obaidy *et al.*, 2015). For example while, for a given sensor configuration, their algorithm converged in 100 generations, while we have obtained the same results in only 55 generations.

5. Conclusions

The MATLAB-based test bench for fine tuning of the parameters of GAs proposed in this paper proved to be efficient, leading to notable improvements in both the resulted optimum value and the convergence speed.

In the context of increasing the performances of GAs, another research direction to have in view is that envisaging their parameter control. In case of parameter control, the parameter values are changing during the run, so one needs initial parameter values and suitable control strategies. The control approach can be deterministic, adaptive, or self-adaptive.

REFERENCES

Ahmed Z. H., *An Improved Genetic Algorithm using Adaptive Mutation Operator for the Quadratic Assignment Problem*, Proc. of the 38th Internat. Conf. on Telecommunications and Signal Processing (TSP), July 9-11, 2015, Prague, Czech Republic, 1324-1330.

Akbaripour H., Masehian E., *Efficient and Robust Parameter Tuning for Heuristic Algorithms*, Internat. J. of Industrial Engng. & Production Res., **24**, *2*, 143-150 (2013).

Al-Obaidy M., Ayesh A., *Energy Efficient Algorithm for Swarmed Sensors Networks*, Sustainable Computing: Informatics and Systems, **5**, 54-63 (2015).

Doughabadi M. H., Bahrami H., Kolahan F., *Evaluating the Effects of Parameters Setting on the Performance of Genetic Algorithm Using Regression Modeling and Statistical Analysis*, J. of Industrial Engng., Univ. of Tehran, Special Issue, 61-68 (2011).

Eiben A.E, Smit S. K., *Evolutionary Algorithm Parameters and Methods to Tune Them*, in Hamadi Y., Monfroy E., Saubion F. (Eds.) Autonomous Search, Springer, 15-36, 2012.

Eiben A.E., Michalewicz Z., Schoenauer M., Smith J., *Parameter Control in Evolutionary Algorithms*, Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence **54**, Springer Verlag, 19-46 (2007).

Eiben A.E., Smit S.K., *Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms*, Swarm and Evolutionary Computation, **1**, 19–31 (2011).

Gunawan A., Lau H.C., Lindawati L., *Fine-Tuning Algorithm Parameters Using the Design of Experiments Approach*, Proc. of the 5th Internat. Conf. Learning and Intelligent Optimization, LION 5, Rome, Italy, January 17-21, 2011, 278-292, 2011.

Haupt R., *Optimum Population Size and Mutation Rate for a Simple Real Genetic Algorithm that Optimizes Array Factors*, Proc. of IEEE AP-S Internat. Symp., July 2000.

Kramer O., *Genetic Algorithm Essentials*, Springer Internat. Publ., AG 2017.
McCall J., *Genetic Algorithms for Modelling and Optimisation*, J. of Computational and Applied Mathematics, **184**, 205-222 (2005).
Moraes Barbosa E.B., França Senne E.L., Silva M.B., *Improving the Performance of Metaheuristics: An Approach Combining Response Surface Methodology and Racing Algorithms*, Internat. J. of Engng. Mathematics, Article ID 167031, 2015.
Petrovski A., Brownlee A., McCall J., *Statistical Optimisation and Tuning of GA Factors*, The 2005 IEEE Congress on Evolutionary Computation, 2005.
Rajakumara B.R., Georgeb A., *APOGA: An Adaptive Population Pool Size Based Genetic Algorithm*, AASRI Procedia, **4**, 288-296 (2013).
Reed P., Minsker B., Goldberg D.E., *Designing a Competent Simple Genetic Algorithm for Search and Optimization*, Water Resources Research, **36**, *12*, 3757-3761 (2000).
Ridge E., Kudenko D., *Sequential Experiment Designs for Screening and Tuning Parameters of Stochastic Heuristics*, Workshop on Empirical Methods for the Analysis of Algorithms, Reykjavik, Iceland, 2006.
Sastry K., Goldberg D.E., *On Extended Compact Genetic Algorithm*, IlliGAL Report No. *2000026*, April, 2000.
Shrestha A., Xing L., *A Performance Comparison of Different Topologies for Wireless Sensor Networks*, IEEE Conf. on Technol. for Homeland Security, 280-285, 2007.
Sun W., *Population Size Modeling for GA in Time-Critical Task Scheduling*, Internat. J. of Foundations of Computer Sci., **22**, *3*, 603-620 (2011).
Witt C., *Population Size Versus Runtime of a Simple Evolutionary Algorithm*, Theoretical Computer Sci., **403**, 104-120 (2008).
* * * *Global Optimization Toolbox User's Guide*, The MathWorks, Inc., 2017.

PLATFORMA DE TEST BAZATĂ PE MATLAB UTILIZATĂ PENTRU ALEGEREA ADECVATĂ A PARAMETRILOR ALGORITMILOR GENETICI

(Rezumat)

Proliferarea calculului evolutiv utilizat în rezolvarea diferitelor probleme tehnice a impus creşterea performanţelor acestora, în mod special dacă implementarea lor în timp real este nesesară. Lucrarea de faţă propune o platformă de test bazată pe MATLAB ce oferă instrumentele de test nesesare pentru alegerea adecvată a parametrilor algoritmilor genetici. Capabilităţile platformei proiectate au fost testate pe o problemă NP-hard şi anume un algoritm de clusterizare a reţelelor de senzori wireless, ce utilizează algoritmi genetici. Rezultatele experimentale au demonstrat importanţa alegerii corespunzătoare a parametrilor pentru o aplicaţie dată. Astfel, folosind platforma de test proiectată, putem raporta creşerea performanţelor algoritmului atât din punctual de vedere al otimului obţinut, cât şi din cel al vitezei de convergenţă.