Adaptive Genetic Algorithm

Domagoj Jakobović, Marin Golub
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia
domagoj.jakobovic@fer.hr, marin.golub@fer.hr

Abstract: *In this paper we introduce an adaptive, 'self-contained' genetic algorithm (GA) with steady-state selection. This variant of GA utilizes empirically based methods for calculating its control parameters. The adaptive algorithm estimates the percent of the population to be replaced with new individuals (generation gap). It chooses the solutions for crossover and varies the number of mutations, all regarding the current population state. The state of the population is evaluated by observing some of its characteristic values, such as the best and worst individual's cost function (fitness) values, the population average etc. Furthermore, a non-uniform mutation operator is introduced, which increases the algorithm's efficiency. Adaptive method does not, however, restrict the applicability in any way. The described GA is applied to optimization of several multimodal functions with various degrees of complexity, employed earlier for comparative studies. Some deceptive problems were also taken into consideration, and a comparison between the adaptive and standard genetic algorithm has been made.*

Keywords: genetic algorithm, local and global optima, adaptive genetic operators

1. Introduction

Genetic algorithm [3][6][8] is a representative of a class of methods based on heuristic random search technique. It was proposed by John H. Holland in early seventies and has found application in a number of practical problems since. A genetic algorithm may be viewed as an evolutionary process wherein a population of solutions evolves over a sequence of generations. The algorithm maintains a set of solutions called *chromosomes*, which are evaluated by fitness function in each generation. After evaluation, solutions are selected for reproduction based on their fitness. Selection embodies the principle of 'survival of the fittest': 'good' solutions are selected for reproduction and 'bad' ones are eliminated. The selected solutions then undergo recombination under the action of genetic operators, *crossover* and *mutation*. Crossover causes exchange of genetic material between solutions; crossed solutions can produce ones with better (or worse) fitness value. It occurs only with some probability p_c - the crossover probability or crossover rate. Mutation is done by modifying a solution with some probability p_m - the mutation probability. The role of mutation is in restoring lost or unexplored genetic material. After performing genetic operators, a generation cycle is concluded and a test is performed in order to determine whether a termination condition is reached or not. A stopping criterion that is going to be used is in most instances chosen by the user.

The strength of GAs lies mainly in their capability to locate the global optimum in a multimodal surrounding. Unfortunately, no matter how robust and efficient a genetic algorithm may be, the solution it provides always bears a certain measure of unreliability. Genetic algorithm can only locate the global optimum with some *probability of success* and a considerable attention has been paid to the efforts to increase that probability. In achieving this goal, two major approaches can be recognized: the first one is to design a GA for a class of problems that we are dealing with. This includes creating data structures and genetic operators characteristic to a problem at hand, creating an *evolutionary program*. The method

is, however, problem specific and requires a lot of modeling for each purpose. The second approach acts on the algorithm directly and tries to increase the efficiency by changing its internal structure. This method generally does not improve the performance of the algorithm as the first one but it is not problem dependent and it does not restrict the applicability at all. The adaptive method presented here is an example of this approach.
The engineers utilizing GAs in everyday practice in most cases do not need the genetic algorithm to be robust and applicable to a wide range of problems. They need it to solve their specific problem, and for that purpose they usually have to create a specialized algorithm that, in general, will not perform well (or will not work at all) when used for other optimization problems. On the other hand, if the algorithm is adapted using second approach, it can *still* be, in most cases, transformed into an adequate evolutionary program. That is why every progress in internal GA structure can be reflected to a variety of applications.
Another good thing we obtain from adapting the genetic algorithm is that we can bypass the task of defining its parameter values, which is in most cases left to the user. Those values are known to significantly affect the algorithm's performance; poorly chosen parameters can cause the algorithm not to produce any relevant solutions at all. Moreover, the optimal parameter configuration is often problem dependent. This can make an inexperienced user's utilization of genetic algorithm very difficult.

2. Adaptive method - the overview

In this work, the standard steady-state GA with elimination selection is altered using three independent techniques which can also be applied separately: the evaluation of the generation gap (the percentage of population to be replaced), the choice of solutions to participate in crossover and the evaluation of the number of mutations. Furthermore, a non-uniform mutation operator is described, which is added to all the GAs in comparative experiments.
Two characteristics are held to be essential in genetic algorithms for optimizing multimodal functions. The first one is the capability to converge to an optimum, local or global, after locating the region containing it. The second characteristic is the capacity to explore new regions of the solution space in search of the global optimum. The balance between these characteristics can be achieved by affecting the way the genetic operators are performed.
We can get a rather good picture of the state the population is in by observing two of its characteristic values: $f_{\max}$ - fitness value of the best member, and $\overline{f}$ - average fitness of the set of solutions, both assigned to a current generation. The expression $f_{\max} - \overline{f}$ is likely to be less for a population that has converged than for a population scattered in the solution space. The above property has already been recognized earlier in literature [11] and it has proven itself in all experiments accompanying this work. A normalized expression has been used here in determining the degree of population diversity:

$$(f_{\max} - \overline{f}) / (f_{\max} - f_{\min}) \qquad (1)$$

where $f_{\min}$ represents the worst fitness value. If the value (1) is low the population is homogenous; if it gets higher the population is more diversed. However, in optimizing problems with a large solution space (long binary strings) this value tends to be very low in the beginning and to raise slowly over the process. This is due to the functions that have approximately average values in most of the defined search space, whereas the higher function values are located in a considerably smaller area. To effectively exploit (1), a correction technique is performed in each generation. In the beginning of the process the expression is evaluated and its value stored in a static variable. It is calculated in each generation and compared to that stored in the variable. If the new value is *greater* than the old one, the value

of the variable is then *replaced* with the new one. Let us name the value of (1) in current generation with *curr_val* and the static variable with *prev_val*.

3. The adaptive techniques

The selection process is adapted by evaluating the generation gap in every cycle. First, the value of the following expression is calculated and named as *w*:

$$w = \left(\frac{curr\_val}{prev\_val} \right)^2 \tag{2}$$

The logic behind *w* is as follows: if the population becomes more homogenus, what we want to avoid, *curr_val* is smaller than *prev_val* and *w* consequently decreases. The 2nd power is added for increased sensitivity. Before calculating (2), the algorithm compares the variables and replaces *prev_val* with the new value if *prev_val* < *curr_val*. If that is the case, the population has become more diversed, which is desirable, and *w* equals one. The number of solutions to replace is then calculated with:

$$N \cdot (0.35 + r^w \cdot 0.6) \tag{3}$$

where *N* is population size and *r* is a randomly generated number between 0 and 1. The value of *w* only drives (3) to be greater if population converges. The elimination of chromosomes is done by roulette-wheel bad individual selection.

The second technique affects the way the chromosomes are picked up for crossover. For every solution a characteristic value *v* is calculated as follows:

$$v = (f - f_{\min}) \cdot (2w - 1) - (f_{\max} - f_{\min}) \cdot (w - 1) \tag{4}$$

where *f* stands for fitness value of a chromosome. Again a roulette-wheel method is used to select the chromosomes for crossover, but this time regarding their characteristic values. A chromosome can participate in crossover more then once, depending on its fitness value. If *w* equals 1, the characteristic values will equal to the corresponding fitness values of the chromosomes. If *w* is 0.5, all the characteristic values will be the same and if *w* is zero, the worst solution will have the highest characteristic value and for the best solution it will equal zero. The probability for participating in crossover is denoted with chromosomes' characteristic values. This reflects to the selection in a following manner: if a population is scattered in problem space the value of *w* will be higher; according to (4) better solutions will have higher characteristic values and they will get higher chance to mate and produce offspring. If *w* is lower, the selection becomes more uniform, and for *w* < 0.5 the algorithm even favors worse solutions.

Finally, the third technique varies the number of mutations in each generation. The number of mutations is calculated as:

$$n = 2N \cdot \frac{prev\_val - curr\_val}{prev\_val} \tag{5}$$

where *N* is population size. The number of mutations increases linearly with the decrease of (1) in current generation. The minimum number of mutations is zero and the maximum number equals twice the size of the population.

This adaptive strategy increases the exploitation of good solutions thus speeding up the convergence and also prevents the population, in most cases, from getting stuck at a local optimum. Each one of these methods can be applied independently, which further increases

their configurability. The adaptive genetic algorithm (hereafter reffered to as AGA) includes all of the explained techniques, as well as the non-uniform mutation.

4. Non-uniform mutation

This mechanism, that is incorporated in every GA included in test section, is proposed separately from the above ones. The non-uniform mutation operator takes into consideration the fitness value of a solution and selects the scope in which the solution will be changed. This is done in practice by restricting the number of bits which mutation operator can affect in a single chromosome. In binary representation, only a set number of rightmost (i.e. less significant) bits can be mutated. The changeable bits form a rightmost substring of a chromosome, the length of which is defined with:

$$bits = 1 + chrom\_length \cdot \left(1 - \frac{f - \overline{f}}{f_{\max} - \overline{f}}\right),\ f > \overline{f} \tag{6}$$

where *chrom_length* is the total number of bits in a chromosome and f is the fitness value of the selected individual. This restriction is only made for solutions whose fitness value is greater then the population average. The same technique for floating-point representation is easily implemented by defining the greatest difference between the old solution and the new one after mutation. For problems where the euclidean distance between chromosomes cannot be defined, this operator is meaningless.

The non-uniform mutation operator has significantly improved the algorithm's 'fine tuning' capabilities. However, the best results, in overall, are achieved when both types (uniform and non-uniform) of operators are included. We conducted the evaluation of the effect of non-uniform mutation by optimizing three functions, defined later in the article. The instances of the same genetic algorithm employed only uniform, only non-uniform and both types of mutations. The figures in Table 1. show average improvement (positive value) or deterioration (negative values) of the results in percentages, relative to uniform mutation. The improvement of 100% here denotes the global optimum.

Table 1. The effect of non-uniform mutation

	f1	*f2*	*f3*
only uniform	0 %	0 %	0 %
only non-uniform	0 %	24 %	-12 %
both	0 %	94 %	78 %

For function f_1 non-uniform mutation did not have any influence at all. In optimizing f_3 the results with non-uniform mutation are even worse then the original ones, but the improvement is gained when both mutation types are involved. In all of the GA implementations following this work, there is a 50% chance that either of the operators will be applied when a chromosome is mutated.

5. Test functions

The choice of suitable functions to verify the performance of GA is not an easy task. All the functions shown here have already been used for similar purposes and can be found in literature. They are given in n-dimensional form, where the index i next to the variable x stands for the i-th element of n-dimensional vector.

Function f_1 : This is a rapidly varying multimodal function symmetric about the origin. Each variable assumes value in range [-100.0, 100.0], and the global optimum is located in the origin. It is also referred to as 'sine envelope sine wave' function:

$$f_1 = 0.5 + \frac{\sin^2 \sqrt{\sum x_i^2} - 0.5}{[1.0 + 0.001 \cdot \sum x_i^2]^2} \tag{7}$$

Function f_2 : This function has the similar properties as f_1. The variables assume values in range [-100.0, 100.0]:

$$f_2 = 1 + (\sum x_i^2)^{0.25} \cdot [\sin^2(50(\sum x_i^2)^{0.1}) + 1.0] \tag{8}$$

Function f_3 : The third function has a number of adjacent mimima with a very small difference to the global optimum in the origin. The variables assume values of [-100.0, 100.0]:

$$f_3 = \sum \left(x_i^2 - 10\cos(2\pi \cdot x_i) \right) \tag{9}$$

Deceptive problem: Deceptive functions are being used extensively to evaluate the performance of GAs. The functions utilized in this work are made by concatenating several 3-order bit functions. A 3-order bit function is defined in Table 2.

Table 2. A three-order bit deceptive function

Binary Code	000	001	010	011	100	101	110	111
Function Value	28	26	22	0	14	0	0	30

6. Experimental results

The performance of the algorithms is measured regarding the fitness value of the best member at the end of a GA run. The experiments were undertaken for 5 and 10 dimensional instances of test functions and for 10 concatenated order-3 bit functions.

Table 3. GA parameter settings

	GA-rw	**GA-eli**	**AGA**
population size	200	200	200
crossover rate	0.4	-	-
generation gap	-	0.8	-
mutation rate	0.01	0.01	-
selection method	roulette wheel	elimination	elimination

Twenty trials have been executed for every configuration, and the performance of a standard roulette-wheel genetic algorithm (*GA-rw*), of an elimination GA (*GA-eli*) and adaptive genetic algorithm (*AGA*, featuring all the adaptive techniques described above) is compared. The algorithms' parameter settings are shown in Table 3. The common features of all the algorithms are binary encoding, precision of 1e-5, uniform crossover operator and both uniform and non-uniform mutation. The dashes in the table denote that the algorithm does not use the specific parameter or it uses adaptive operators as in case of AGA. The optimization results are listed in Table 4.

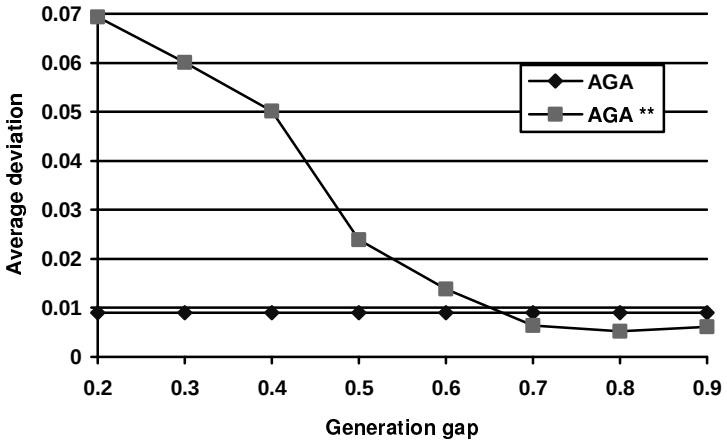
Table 4. Optimization results

	GA-rw		GA-eli		AGA			
	av_gen	**av_dev**	**av_gen**	**av_dev**	**av_gen**	**av_dev**	**gen**	**thresh**
5dim f1	**9695**	**0.00793**	10000	0.01239	10000	0.00971	10000	0.001
10dim f1	**20000**	**0.01246**	20000	0.28930	20000	0.03172	20000	0.001
5dim f2	10000	0.07668	10000	0.02041	**10000**	**0.00902**	10000	0.001
10dim f2	20000	0.45994	20000	0.79667	**20000**	**0.07962**	20000	0.001
5dim f3	8084	0.25335	973	0.10679	**1248**	**0.00084**	10000	0.001
10dim f3	20000	14.5267	**3743**	**0.00812**	20000	1.97158	20000	0.01
*10*ord3*	3294	0.4	5000	8.4	**562**	**0**	5000	0

The *av_gen* column denotes the average generation number whereas *gen* column shows the maximum number of allowed generations per run. Field *av_dev* contains the average deviation from the global minimum of the best population member (smaller value indicates better performance). If the algorithm reaches the *thresh* value (i.e. if the difference between the optimum and the best member's fitness value is less then *thresh*), the run is terminated. This is important in *10*ord3* optimization, where only the exact maximum value means that the global optimum is reached. The notable improvement can be perceived in optimizing f_2 and deceptive order-3 bit function, whereas for f_1 AGA did not outperform its competitors. The elimination genetic algorithm managed to get the best results in optimizing ten-dimensional f_3, while AGA was the most succesfull one in five-dimensional variant.

It must be noted that all of the adaptive techniques implemented in AGA did not improve its performance in the same measure. The least successful of them was, as observed in the implementation process, the generation gap evaluating algorithm. We will illustrate that fact in Figure 1.

The figure shows optimization results for function f_2. The algorithm marked with AGA** presents the version in which we have a constant generation gap value. It can be seen from the



picture that we can obtain the best results when all the adaptive properties except the generation gap evaluation are included and when the user manually selects that parameter.

7. Conclusion

This paper describes an adaptive technique that alters the functioning of a steady-state GA. It increases the overall effectiveness of the algorithm on one hand and relieves the eventually

inexperienced user of providing parameter values whose choice significantly affects the algorithm's performance. Rather, it uses certain population-specific values to determine the need for each operator and applies it.
In most of the test cases, the adaptive genetic algorithm outperformed the standard versions, but there were also occasions when it performed relatively worse then it would be expected. It is our opinion that the generation gap evaluation technique is still not general and robust enough to be applied for a variety of the optimization problems. Nevertheless, the adaptation made it possible for the AGA to act relatively good over a number of functions. The adaptive techniques can be implemented independently by a more experienced user, which can, combined with specific problem dependent knowledge, lead to a significant performance enhancement. The human factor remains the most responsible element for successful GA utilization in real-life applications.

Acknowledgment

This work was carried out within the research project "*Problem-Solving Environments in Engineering*", supported by the Ministry of Science and Technology of the Republic of Croatia.

References

[1] Alippi, C., Filho, J.L.R., Treleaven, P.C., (1994), "Genetic-Algorithm Programming Environments", *IEEE Trans. Computer*, June 1994.
[2] L. Budin, M. Golub, D. Jakobović, Parallel Adaptive Genetic Algorithm, *International ICSC/IFAC Symposium on Neural Computation NC'98*, Vienna, 1998, pp. 157-163.
[3] Davis, L. (1991) *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York
[4] Hinterding, R., Michalewicz, Z., and Eiben, A.E. (1997), "Adaptation in Evolutioary Computation: A Survey", *Proc. 4th IEEE Int. Conference on Evolutionary Computation, 1997.*
[5] Hinterding, R., Michalewicz, Z., and Peachey, T.C. (1996), "Self-Adaptive Genetic Algorithm for Numeric Functions", *Proc. of the Fourth Int. Conference on Parallel Problem Solving from Nature, Berlin, Sept 1996.*
[6] Jakobović, D. (1996) "Utjecaj prilagodljivih genetskih operatora na učinkovitost genetskog algoritma", (in Croatian) *undergraduate dissertation*, FEEC, University of Zagreb.
[7] Jakobović, D. (1997) "Adaptive Genetic Operators in Elimination Genetic Algorithm", *Proc. 19th Int. Conf. ITI'97,* Pula, 17-20 June 1997, pp.351-356
[8] Michalewicz, Z. (1992), *Genetic Algorithms + Data Structures = Evolutionary Programs*, Springer-Verlag, Berlin.
[9] Schoenenburg, E., Heinzmann, F., Feddersen, S. (1995) *Genetische Algorithmen und Evolutionsstrategien*, Addison-Wesley, 1995.
[10] Srinivas, M., Patnaik, L. M. (1994), "Genetic Algorithms: A Survey", *IEEE Trans. Computer*, June 1994.
[11] Srinivas, M., Patnaik, L. M. (1994), "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms", *IEEE Trans. Systems, Man and Cybernetics*, Apr. 1994.