Adaptive Genetic Operators in Elimination Genetic Algorithm

Domagoj Jakobovic
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia
domagoj.jakobovic@fer.hr

Abstract: This paper describes a variant of Genetic Algorithm (GA) with adaptive probabilities of crossover and mutation. The application of GA presented here refers to multimodal function optimization and some deceptive problems. Adaptive probabilities do not, however, restrict the applicability in any way. The adaptive method is designed for steady-state GA with elimination selection. The main idea of the method lies in selective choosing of solutions to be crossed and in varying number of mutations. The adaptive GA is analyzed concerning its capability of locating the global optimum in a multimodal landscape. Several test functions with various degrees of complexity, employed earlier for comparative studies, have been optimized. The performance of standard steady-state GA is compared with that of the adaptive GA. A notable improvement can be perceived in the results: the adaptive GA converges faster and gets stuck at a local optimum fewer times. There is also less preparation needed for the user to get the algorithm to work, as it calculates some of its control parameters itself.

Keywords: genetic algorithm, elimination selection, generation gap, local and global optima

1. Introduction

Genetic algorithms [1][3][5] were proposed by John H. Holland in early seventies and in the following twenty years they have found application in a number of practical problems. The strength of GAs lies mainly in their capability to locate the global optimum in a multimodal surrounding. Unfortunately, they can only locate the solution with some probability of success. A considerable attention has been paid to the efforts to increase that probability, and this work has the same purpose. 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 GA directly and tries to increase the efficiency by changing its internal structure. This method is not problem specific and it does not restrict the applicability at all. Moreover, GAs that have been altered in this way can also be transformed into evolutionary programs, so every progress in this area can be reflected to a wide range of applications. The adaptive method presented here is an example of this approach.

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 probabilities of crossover and mutation (hereafter referred to as p_c and p_m) remain on

the user to be determined. The choice of p_c and p_m is known to critically affect the behaviour and the performance of the GA, and a number of guidelines exist in the literature for choosing p_c and p_m. That choice also becomes specific to the problem under consideration. It is for the above reason that the adaptive methods are being developed: to form a self-contained algorithm that will be both efficient and easy to use.

2. Elimination selection and elitism

The adaptive method described in this paper is designed for *steady-state* GA involving an elimination selection mechanism. The first ever genetic algorithm, also called SGA (simple GA), employs a *generational* principle; that is, it replaces the whole population during selection with a number of chosen individuals. The more general steady-state concept replaces only a part of the population per cycle. It eliminates a predefined number of members (a percentage of population) and then creates the same number of new chromosomes, to maintain the population size, using crossover as a making tool. The percentage of population to be eliminated is called the *generation gap*.

What are the differences in these two approaches? A generational GA uses some method to select the full number of solutions to form a new set from the old one. In that process, better solutions can get more than one offspring in the new generation, whilst worse ones can disappear. A steady-state GA with elimination selection works in a different manner. The solutions selected for the next generation *never* get more than one offspring. On the other hand, the crossed solutions are not destroyed (in generational GA crossover replaces chromosomes with its products) because we have to compensate the generation gap - crossover generates new chromosomes for the population. The parameter p_c has no meaning in elimination GA. Generation gap, which is used instead, tells us exactly what number of crossovers we have to perform. The adaptive methods for generational GAs have been developed earlier [6], but because of the above differences the same logic cannot be applied to elimination based algorithm.

Another feature that steady-state GAs posses is implicit elitism. *Elitism* is realized through preservation of the best member in the population. During the elimination selection, the best member always gets an offspring, while that is not certain for any other solution, and that is why are steady-state GAs said to have an implicit elitism. The idea of elitism can easily be implemented in any GA; we have to pick the best member in every generation and preserve it from changing. It is a common phenomenon that without elitism the best member often becomes disrupted, and it is possible that at the end of a GA run we do not have the best-member-ever present in population. That is why both the GA variants tested in this work employ the elitism concept.

3. Adaptive crossover and mutation operators

Two characteristics are held to be essential in GAs for optimizing multimodal functions. The first one is the capability to converge to an optimum, local or global, after locating the region containing the optimum. 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 is determined by the values of p_c and p_m and the type of the operators employed. As we speak in terms of steady-state GA, we will not use the value of p_c, but instead we will affect the way the solutions are picked up for the crossover.

It is commonly understood that crossover plays a main role in converging, by combining the solutions close to an optimum. If we choose solutions with higher fitness values for crossover, we may expect GA to converge faster to the nearby optimum. On the other hand, when population becomes too homogenous it is not clever to favour solutions in that way, because there is a danger of getting stuck to a local optimum.

Mutation is the operator which is mainly responsible for preventing the GA of becoming stuck. If a population converges to a local optimum, it is possible to drive it away with increased mutation probability. But to vary the choice of solutions to be crossed and the mutation rate, it is essential to be able to identify whether the GA is converging to an optimum.

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 [1] and it has proven itself in all experiments accompanying this work. However, without proper scaling mechanism (adjusting the values of fitnesses to have similar relations regardless of the problem under consideration), this value can vary significantly from case to case. That is why another, normalized expression has been used here in determining the degree of population diversity:

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

Before evaluating (1), fitness value of the worst member is subtracted from $f_{\max}$ and $\overline{f}$ in order to make them always greater than zero. The expression assumes values between 0 and 1, usually between 0.05 and 0.8 for most of the applications. If the value is low the population converges to local optimum; 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 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. If it is lower, the adaptive mechanism takes effect.

In GA implementation utilized in this work the solutions are ranked regarding their fitness values. The choice of the solutions to be crossed is done by calculating their indexes in the following manner:

$$i = N \cdot r^{\exp} \tag{2}$$

where N is population size, i index of a member to participate in crossover and r a randomly generated number between 0 and 1. The best member is assigned number 1 and the worst N. If exp equals 1, the chromosomes are picked uniformly; if exp is greater than 1, index i is lower and 'better' solutions are favoured. The parameter exp is evaluated as:

$$\exp = k_1 + k_2 \cdot \left(\frac{p}{corr} \right)^2 \tag{3}$$

where p stands for (1) *in current generation*, and *corr* for the value stored in the static variable. Before calculating (3) the algorithm compares p and *corr* and replaces *corr* with new value if $p>corr$. The experiments have shown the suitable choice of values 0.0 for k_1 and 3.0 for k_2. In the beginning of the run p equals *corr* and exp=3, that is, the stress is on the 'better' solutions. If GA tends to become stuck, exp is lower and the choice gets more uniform. In some cases exp can even get below one - algorithm then crosses 'worse' solutions to prevent premature convergence.

Mutation is adapted in a different way: the method only varies the number of mutations to be done. Number of mutations n is calculated as follows:

$$n = k_3 \cdot N \cdot \frac{corr - p}{corr} \tag{4}$$

where N is population size. The constant k_3 is assigned value of 2.0. If p is lower than *corr*, which corresponds to a more homogenous population, the number of mutations increases linearly.

The general characteristic that steady-state GAs show is somewhat slower convergence then generational GAs, but they also tend less to get stuck at a local optimum. This adaptive strategy rapidly increases the exploitation of good solutions thus speeding up convergence and also prevents the population, in most cases, from getting stuck. Such modified GA will be hereafter called 'EAGA' (Elimination-Adaptive Genetic Algorithm), whereas the standard elimination variant used in comparative tests will be referred to as 'GA' (not the generational one!).

4. Test functions

The choice of suitable functions to verify the performance of GAs is not an easy task. All the functions shown here have already been used in similar purposes and can be found in literature. Their variety gives an amount of credibility to the tests performed. The functions are given in *n*-dimensional form.

Function f6:[6] 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:

$$f6 = 0.5 - \frac{\sin^2\sqrt{\sum x_i^2} - 0.5}{[1.0 + 0.001 \cdot \sum x_i^2]^2}$$

Function f7: [6] This function has the similar properties as *f6*. The variables assume values in range [-100.0, 100.0]:

$$f7 = (\sum x_i^2)^{0.25} \cdot [\sin^2(50(\sum x_i^2)^{0.1}) + 1.0]$$

Function f8: [4] This function is referred to as Griewank's function. It has a number of adjacent mimima with a very small difference to the global optimum in the origin. The variables assume values of [-600.0, 600.0]:

$$f8 = \sum \frac{x_i^2}{4000} - 20 \cdot \prod\left(\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)$$

Deceptive problems: Deceptive functions are being used extensively to evaluate the performance of GAs. Three deceptive problems were utilized in this work. The first one is defined as follows [2]:

$$f_{32767dec}(x) = \begin{cases} x, & 1 \le x \le 32767 \\ 32768, & x = 0 \end{cases}$$

where x is an integer variable that assumes values from 0 to 32767. The other two functions are made by concatenating five and ten 3-order functions. The 3-order bit function [6] is defined in Table 1.

Table 1. 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

5. Experimental results

The performance of the GAs is expressed as a number of times (out of total number of trials) when they got stuck at a local optimum, i.e., they were not able to locate the global optimum within a set number of generations. The GA was considered to reach the global optimum if any

member's fitness value exceeded certain threshold. This threshold is greater than any of the local optimum values, so the algorithm must have located the global optimum. The population sizes were varied to illustrate the effectiveness of the GAs. Tests were made for two- and five-dimensional versions of the functions. The characteristics of the algorithms were as follows:

- binary encoding with fixed point encoding scheme,
- generation gap of 40% (40 percent of the population is eliminated in selection),
- p_m= 0.03 (*this parameter has no effect in EAGA*),
- elimination selection,
- uniform crossover,
- uniform mutation,
- elitism.

For explanation of the above features, reader is advised to [2],[3],[5]. The only difference between GAs was the adaptive method employed in EAGA. The results for the two-dimensional functions are shown in Table 2.

Table 2. Two-dimensional functions

function	pop_size	GA	EAGA	str_len	max_gen	thresh
2dim f6	30	**6** / 50	**2** / 50	44	200	0.9999
2dim f7	30	**2** / 50	**3** / 50	56	200	-0.001
2dim f8	30	**6** / 50	**1** / 50	48	200	80
2dim f6	100	**0** / 30	**0** / 30	44	200	0.9999
2dim f7	100	**0** / 30	**0** / 30	56	200	-0.001
2dim f8	100	**0** / 30	**0** / 30	48	200	80

The numbers in '**n**/m' format show how many times (**n**) GA or EAGA has become stuck at a local optimum out of total number of runs (m). Field **function** denotes a certain test function being optimized, **pop_size** is the population size and **str_len** is the length of the binary string (chromosome). **max_gen** stands for maximum number of generations allowed to algorithm and **thresh** is the fitness value a GA must reach to finish the process successfully.

It can be seen from the results that both algorithms have been equally successful with large-enough population sizes. When the size of population is small, on the other hand, EAGA shows better performance. Population sizes were adjusted in that way that this difference can be noted.

Table 3. Five-dimensional functions

function	pop_size	GA	EAGA	str_len	max_gen	thresh
5dim f6	100	**28** / 30	**6** / 30	110	500	0.9999
5dim f7	150	**7** / 50	**5** / 50	140	500	-0.001
5dim f8	200	**5** / 10	**3** / 10	120	500	640
5dim f6	300	**9** / 10	**0** /10	110	500	0.9999
5dim f7	300	**0** / 10	**0** / 10	140	500	-0.001
5dim f8	300	**0** / 10	**0** / 10	120	500	640

Results for the five-dimension function optimization are shown in Table 3. The results are similar to the ones in the previous table. Only in optimizing five-dimensional *f6* standard GA did not catch up with EAGA. EAGA showed a very good efficiency (relative to GA) in solving deceptive problems, which is displayed in Table 4.

Table 4. Results for deceptive problems

function	pop_size	GA	EAGA	str_len	max_gen	thresh
32767 dec	30	**15** / 50	**0** / 50	15	200	32768
*5*ord 3*	10	**35** / 50	**0** / 50	15	200	150

*10*ord 3*	30	**5** / 50	**1** / 50	30	400	300

The deceptive nature of the problems under consideration has the property to drive the population off the global optimum. However, when population converges to the 'wrong end', increased number of mutations scatters the chromosomes over the solution space and eventually locates the right solution.

6. Conclusion

In the last few years the emergence of new ideas that drastically change functioning and performance of GAs has accompanied their increasing use in many areas of application. GAs have proved themselves to be robust, efficient and easy-to-use with ever growing computing power today at hand. This paper describes an adaptive method for steady-state GA that alters the manner in which genetic operators work. On one hand, it increases the effectiveness of the algorithm by preserving it to get stuck at the local optimum. On the other, it relieves the user of providing another parameter 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. Better solutions get higher chances of undergoing crossover, thus helping the convergence capacities of the GA, while varying mutation rate provides the necessary diversity of the population.

In most of the test cases, adaptive GA outperformed the standard version and achieved a significant improvement. Nevertheless, more testing on a wide range of problems is needed to estimate the GA's potentials, which is mainly done in practice. Future upgrades of EAGA could include automatic choice of generation gap; the experiments point that the choice of this parameter affects the GA's performance in a great measure.

Acknowledgment

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

References

[1] Filho, J.L.R., Treleaven, P.C., Alippi, C. (1994), "Genetic-Algorithm Programming Environments", *IEEE Trans. Computer*, June 1994.

[2] Jakobovic, D. (1996) "The Effect of Adaptive Operator Probabilities on Performance of Genetic Algorithm", (in Croatian) *undergraduate dissertation*, FEEC, University of Zagreb.

[3] Michalewicz, Z. (1992), *Genetic Algorithms + Data Structures = Evolutionary Programs*, Springer-Verlag, Berlin.

[4] Schoenenburg, E., Heinzmann, F., Feddersen, S. (1995) *Genetische Algorithmen und Evolutionsstrategien*, Eddison-Wesley, 1995.

[5] Srinivas, M., Patnaik, L. M. (1994), "Genetic Algorithms: A Survey", *IEEE Trans. Computer*, June 1994.

[6] Srinivas, M., Patnaik, L. M. (1994), "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms", *IEEE Trans. Systems, Man and Cybernetics*, Apr. 1994.