A COMPENSATORY GENETIC ALGORITHM

Yuri R. Tsoy, Vladimir G. Spitsyn
Department of Computer Engineering,
Tomsk Polytechnic University,
84, Sovetskaya street, Tomsk, 634034, Russia,
Tel: +7 3822 418912,
Fax: +7 3822 419149,
neuroevolution@mail.ru

Abstract

A new subclass of genetic algorithms (GA) is introduced. The primary idea of its creation is to extend searching abilities and prevent premature convergence. Some unexpected results such as inverse dependence of algorithm performance from population size and possible solution of some of the direct encoding problems are discussed.

Introduction

Process of the challenging task solution is often similar to evolution of ideas. At the beginning, on the foundation of given facts, a hypothesizes are formulated, then they are verified, elaborated, also their combinations and variations can appear. Success of variants and their further destiny depends on how well they fit for the task. In the long run either an answer is found or we have to admit that our solution search is stuck and we need to start from the very beginning.

Genetic algorithms (GA) are known as quite common and robust optimization concept. As they use evolutionary mechanisms to evolve good solution from population of potential ones, they have similar to mentioned before problem. In evolutionary computations it is called premature convergence and its feature is degeneration of population when evolution of solutions is merely stopped.

One of the main causes of this phenomenon is allele loss that means that there are no differences for certain loci in chromosomes all over the population. As generations pass the number of such loci can increase and it can cause premature convergence. One of the causes of allele loss is prevail of certain schema in the population. If this schema is highly fitted then we can expect with certain probability that in several generations most of population will correspond to this schema. From one hand it helps to lead evolutionary process but from another hand it involves loss of diversity. It was noticed that increase of population size and mutation probability weakens allele loss [1]. The notion is to have highly fitted competing schemas in every generation. We are going to show further that the same effect can be achieved with use of radically different technique.

Algorithm description

In general suggested algorithm doesn't differ much from canonical GA (CGA) described in [2]. The same proportional selection strategy and one-point crossover operator are used. Distinctions are following – no mutation process, use of elitism technique and compensatory strategy. The last distinction is the key feature of the introduced algorithm.

Compensatory mechanism starts working after selection of strings to mate is done. Lets denote N' – number of strings selected, a'_{ik} – value of loci k in the string i, then k-th compensatory string element at generation t is defined according to the following formula:

$$c_k(t) = \begin{cases} 0, \sum_{i=1}^{N'} a'_{ik}(t) > \frac{N'}{2}, \\ 1, \sum_{i=1}^{N'} a'_{ik}(t) < \frac{N'}{2}, \\ RB, \sum_{i=1}^{N'} a'_{ik}(t) = \frac{N'}{2}, \end{cases} \qquad (1)$$

here RB – random boolean variable. Thus compensatory string is an inversion of the prevalent loci string. Example of such transformation is shown in the figure 1. Upper string houses prevalent loci values, for example digit with index 5 is "1", and it means that number of "1" bits is more than that of "0" bits in the corresponding position all over the population.

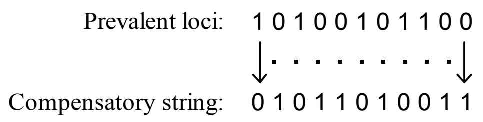
Prevalent loci: 1 0 1 0 0 1 0 1 1 0 0
Compensatory string: 0 1 0 1 1 0 1 0 0 1 1

Figure 1. Compensatory string construction

After compensatory string is constructed we then mate it with randomly chosen string selected for reproduction to produce offspring. Next generation is formed from this posterity, one best string from previous generation and a compensatory string. Use of such strategy can guarantee GA against premature convergence as we artificially compensate prevalence of "0" or "1" in any position in population. This is the reason to call introduced algorithm a "Compensatory Genetic Algorithm" (CoGA).

Experiments and discussion

To test performance of CoGA a set of functions was taken. Following functions included: ONEMAX, Sphere function (n=30) and Rastrigin's function (n=50). ONEMAX task requires a string of "1" to be assembled during evolution. 16 and 32 bit strings were used. Sphere function is a simple unimodal function whereas Rastrigin's function used in experiments has 10^{50}-1 local optima. We will compare results of CoGA with that of CoGA without elitism and canonical genetic algorithm (CGA). All algorithms were limited to 51200 target function evaluations so that the bigger the population size, the smaller number of generations population evolves. For example population of 512 strings has 100 generations to search the solution. For each task 5 runs were made and best results picked.

Results of experiments are shown in figures 2-5. Vertical axis holds target function values.

On simple ONEMAX problems with populations of 256 strings and more CGA shows better or equal performance by comparison with CoGA.

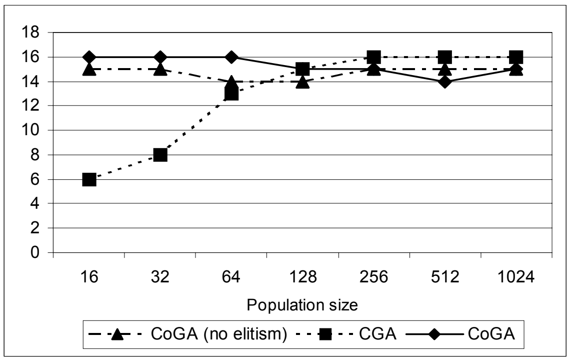

Figure 2. ONEMAX 16 task

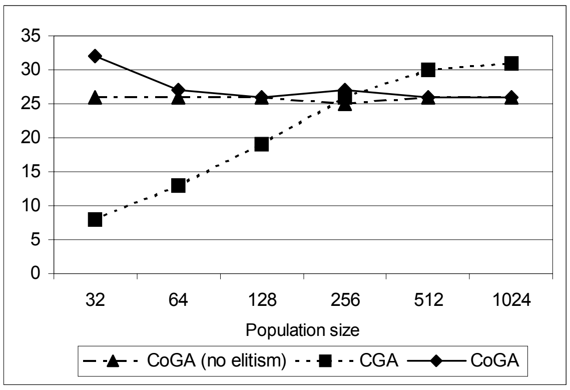

Figure 3. ONEMAX 32 task

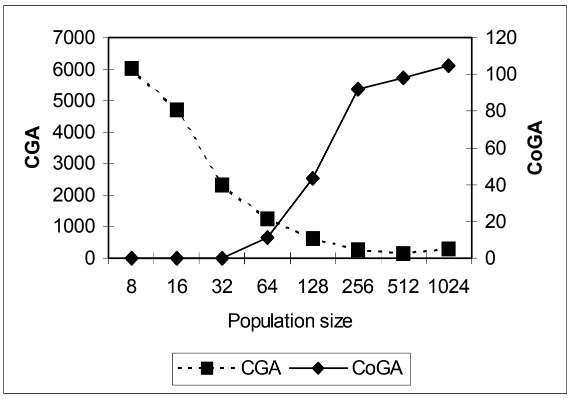

Figure 4. Sphere function

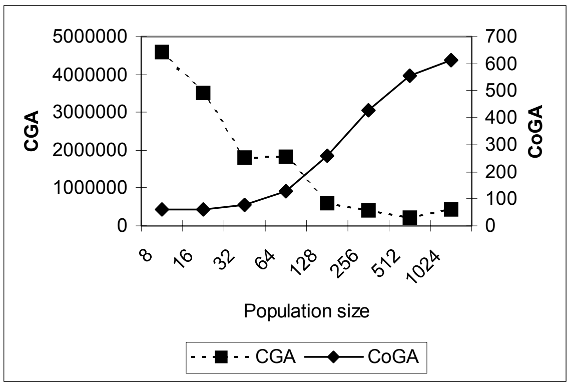

Figure 5. Rastrigin's function

In case of more complex Sphere and Rastrigin's functions CGA shows worse results in all runs. CoGA without elitism performs almost identically in all cases for ONEMAX problems.

As it can be seen CoGA's performance decreases as population size growths. To demonstrate it best results for Sphere function task achieved at 25-th, 50-th, 100-th and 200-th generation for populations of different sizes are given in table 1.

Table 1. Population size dependence

Popul. size	Generation			
	25	50	100	200
8	97.601	68.9143	44.9022	19.5031
16	52.764	44.264	21.1777	12.3515
32	83.3845	43.7421	32.5175	16.1648
64	77.3846	45.1136	39.9596	32.4642
128	76.8295	55.1895	54.5362	44.4661
256	106.738	106.738	98.1128	91.8086
512	121.178	107.441	97.9799	-
1024	112.807	104.868	-	-

Populations with small number of strings evolve much faster and their results are better then results of large populations. To explain this effect a suggestion can be made that in large populations there is a less significant bias of prevalent loci but surely this requires a much more thorough investigation.

As a side effect a problem with direct encoding when numbers 2^n and (2^n-1) differ much in binary representation can be possibly solved. When we construct compensatory string we "automatically" overcome this problem. If we have prevalent genes of 2^n type then compensatory string will likely house (2^n-1) type number and vice versa.

At first sight it seems unnatural for such algorithm to work because GAs considered to evolve highly fitted schemas and we seem to impede this process. It is surely so but there is no guarantee that prevailing schema is best of all possible ones so we can try to search for other schemas and compensatory strategy gives good opportunity to do this. Besides current good schema is not disrupted because of elitism.

Conclusion

Introduced compensatory strategy brings nice searching abilities and effective against premature convergence. Implementation of this strategy involves some interesting results when population size doesn't matter and the main thing, at least for used tasks, is the number of generations a population should evolve. In large populations speed of evolution is slow and probably because of insignificant prevalent loci bias.

In general performance of CoGA is better then that of canonical GA. Future work to be done to improve performance possibly by implementation of other selection strategy and some modifications in compensatory strategy.

References

1. Kenneth A. De Jong Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan, 1975. (University Microfilms No. 76-9381).
2. Whitley, D., A genetic algorithm tutorial, 1994, Statistics and Computing, No. 4, 65-85.