International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:1, 2007

Restartings: A Technique to Improve Classic Genetic Algorithms' Performance

Grigorios N. Beligiannis, Georgios A. Tsirogiannis and Panayotis E. Pintelas

Abstract**—**In this contribution, a way to enhance the performance of the classic Genetic Algorithm is proposed. The idea of restarting a Genetic Algorithm is applied in order to obtain better knowledge of the solution space of the problem. A new operator of "insertion" is introduced so as to exploit (utilize) the information that has already been collected before the restarting procedure. Finally, numerical experiments comparing the performance of the classic Genetic Algorithm and the Genetic Algorithm with restartings, for some well known test functions, are given.

Keywords**—**Genetic Algorithms, Restartings, Search space exploration, Search space exploitation.

I. Introduction

GENETIC Algorithms (GAs) are known to be one of the best methods for searching and optimization [1]–[3]. By applying genetic operators (reproduction, crossover and mutation) in a population of individuals (sets of unknown parameters properly coded), they achieve the optimum value of the fitness function, which corresponds to the most suitable solution. As a result, they converge to the (near) optimal solution by evolving the best individuals in each generation. The main advantage of the GAs is that they use the parameters' values instead of the parameters themselves. In this way they search the whole parameter space. However, GAs encounter some serious problems (concerning the convergence speed and the finding of the exact value of the global optimum) when they have to deal with functions that contain too many local optima.

The idea of restarting the classic GA, so as to increase the performance, derives from the well known idea of restarting the Arnoldi's method for finding the eigenvalues [4]–[7]. In this restarting technique [4]–[7], we are not willing to throw away the useful information, concerning the Krylov subspace, that has been captured before restarting. This is achieved by using a vector that constitutes of a mix of Ritz values [4], as a starting vector for the next step of Arnoldi's method.

In order to apply this strategy to the classic GA the following technique is established. A fixed number of genomes – from the current population before the restarting –is selected and included into the new population. Hopefully these genomes encapsulate all the useful information gathered about the solution space till that generation. The role of the "vector that is a mix of Ritz values" is played by the set of genomes that is passed to the new generation.

The paper is organized as follows. In section II the proposed technique is described and analyzed. In section III experimental results are presented in order to prove the significance and the efficiency of the proposed technique. Finally, section IV summarizes the conclusions and suggests future applications and extensions of the method.

II. GAs' Restartings

Experimental results have shown that GAs, when used for optimizing a function, are able to reach a relative good score (compared to the global optimum) in a quite small number of generations. In the sequel generations, they just refine the solution space trying to identify the exact optimal solution of the function. As known, classic GAs make use of three basic genetic operators (selection, crossover and mutation) in order to evolve the population of possible solutions to fit to the conditions and the characteristics of each specific problem.

Generally speaking, one can interpret the initial generations of a classic GA as a global search mechanism and all the remaining ones as a refining procedure towards the true optimal value of each specific optimization problem. At this point, it should be noticed that the application of the crossover operator is a clever way to escape from local optima (at the early stages of the evolution procedure it assists the effective exploration of the whole search space). So, in a typical classic GA, trying to avoid premature convergence, the evolution procedure can be described as in Fig.1.

It is clear that this procedure wastes a large amount of evaluations of the objective function in order to refine a local optimum solution which is often abandoned in the next generations due to the application of the mutation operator. This is because the use of the mutation operator can often lead to much better solutions compared to the ones found so far, directing the evolution of the algorithm to another area of the search space in which a local optimum with higher objective

G. N. Beligiannis is with the Department of Computer Engineering and Informatics, University of Patras, 26500 Rio, Patras, Greece and with the Research Academic Computer Technology Institute (R.A.C.T.I.), Riga Feraiou 61, 26221 Patras, Greece (corresponding author, phone: +30-2610-997755; fax: +30-2610-997706; e-mail: beligian@ceid.upatras.gr).

G. A. Tsirogiannis is with the Department of Engineering Sciences, University of Patras, Rio, Patras, Greece (e-mail: g.tsirogiannis@des.upatras.gr).

P. E. Pintelas is with the Department of Mathematics, University of Patras, Rio, Patras, Greece, (e-mail: pintelas@math.upatras.gr).

This contribution was financially supported by the Greek Ministry Of Education and Religious Affairs under grant "PYTHAGORAS: Supporting Research Groups in Universities", b.365.018 (EPEAEK II).

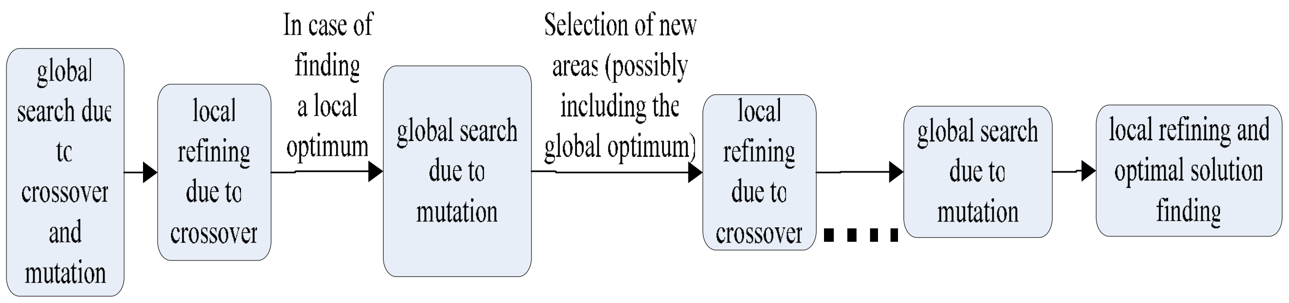
global search due to crossover and mutation
local refining due to crossover
In case of finding a local optimum
global search due to mutation
Selection of new areas (possibly including the global optimum)
local refining due to crossover
global search due to mutation
local refining and optimal solution finding

FIGURE 1
THE EVOLUTION PROCEDURE OF A CLASSIC GA

value lies. In general, the application of the mutation operator usually leads to a "from the beginning" global search for "good" areas (areas of the search space that will hopefully include the global optimum). The use of restartings manages to succeed better exploration of the search space by avoiding premature convergence and entrapment in local optima while at the same time it saves most of the evaluations included in the local refining procedure (fewer generations and evaluations of the fitness function).

In this contribution, a restarting procedure for the classic GA is proposed so as to achieve a better global exploration of the solution space while executing the minimum possible number of generations (function evaluations). In order to achieve this goal, we use the standard global exploration mechanism used by classic GAs (selection, crossover, mutation) but when the GA reaches the local refining phase, we restart the GA so as to preserve the global search procedure. This technique alleviates the enormous computational burden introduced by the local refining procedure, which is quite often useless in finding the optimal solution. The proposed technique is described in Fig. 2. Of course, the new starting of the GA procedure should include all the valuable information gathered from the previous global search. Thus, we propose a new operation called "insertion" to be included in the classic GAs' evolution procedure. The insertion operator works as follows. It chooses randomly a constant percentage of the genomes of the population of the last generation (before the restarting procedure takes effect) and inserts them into the new initial population of the GA as shown in Fig.3.

The main difficulty of all restarting techniques is to decide when to apply the restartings. If they are applied too early (in a rather early phase of the evolution procedure), the global search procedure completed till that point will be able to reveal only a small part of the useful information included in the solution space of the specific problem. In other words, the GA should be let to run for a minimum number of generations (before the application of the restarting procedure) in order to manage to search effectively the solution space and gather useful information. On the other hand, if restartings are applied too late (in a rather late phase of the evolution procedure) most of the information carried by the genomes of the last population (the population before the restarting procedure) will be concentrated on a local solution, probably not the optimal one. Of course, this leads to loss of useful information and premature convergence.

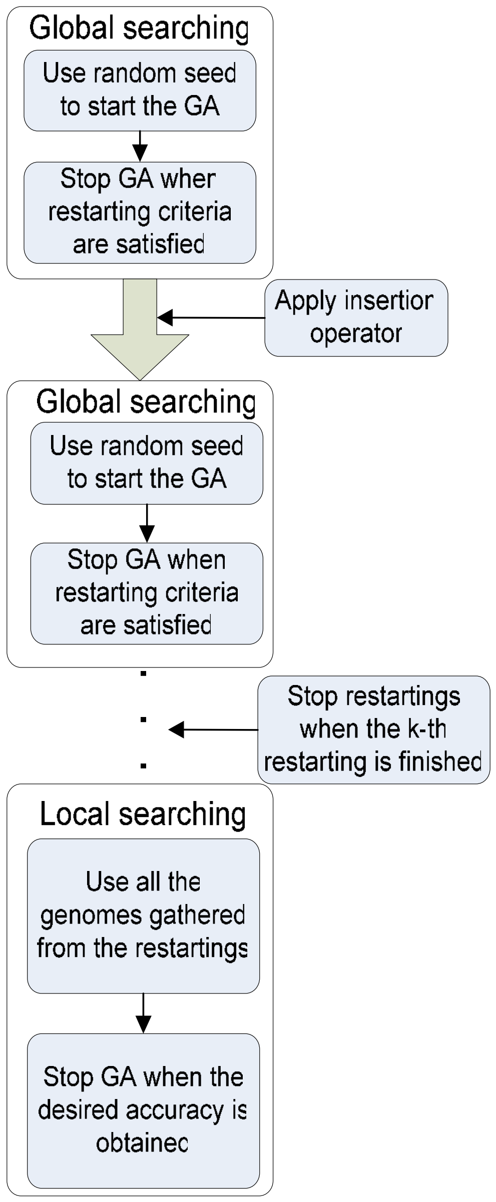
Global searching
Use random seed to start the GA
Stop GA when restarting criteria are satisfied
Apply insertion operator
Global searching
Use random seed to start the GA
Stop GA when restarting criteria are satisfied
Stop restartings when the k-th restarting is finished
Local searching
Use all the genomes gathered from the restartings
Stop GA when the desired accuracy is obtained

FIGURE 2
THE EVOLUTION PROCEDURE OF THE PROPOSED GA WITH RESTARTINGS

In this contribution, three different criteria for deciding when to apply restartings are proposed:

- Fitness function value (that is the restarting procedure is applied whenever the value of the fitness function exceeds a predefined threshold)
- Number of generations (that is the restarting procedure is applied whenever the number of generations executed exceeds a predefined threshold)
- Mean fitness function value of population (that is the restarting procedure is applied whenever the mean value of the fitness function of the whole population exceeds a predefined threshold)

Another important aspect of the restarting method is to decide its termination criterion, that is, to decide when the application of the restarting procedure used to refine the solution should stop. A rule of thumb is the following: the more complex the solution space is, the more times the restarting procedure should be applied. Following this rule, an integer constant, whose value is totally depended on the complexity of the solution space, is proposed in each specific application in order to specify the total number of restartings.

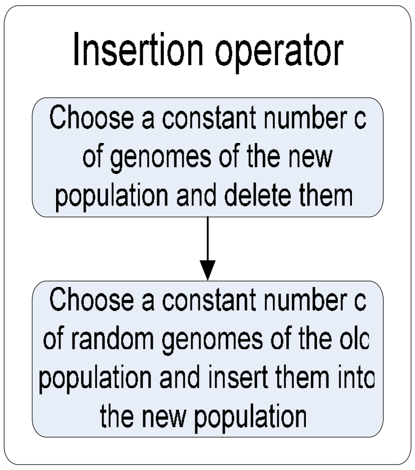
Insertion operator
Choose a constant number c of genomes of the new population and delete them
Choose a constant number c of random genomes of the old population and insert them into the new population

FIGURE 3
THE INSERTION OPERATOR

III. EXPERIMENTAL RESULTS

In order to demonstrate the efficiency and performance of the proposed technique, several simulation experiments were carried out. All the experiments were carried out 100 times (100 Monte Carlo runs). In this section, we present the results of the application of both the classic GA and the proposed GA with restartings to four well known optimization problems. The functions selected to be optimized are the first three functions of the De Jong test suite [8] and the Himmelblau function [9]. These functions are quite popular in GAs' literature, so it is possible to make direct comparisons.

The first De Jong test function is the sphere model:

$$f_1(x_1, x_2, x_3) = \sum_{i=1}^{3} x_i^2 \;, \; -5.12 \le x_i \le 5.12 \qquad (1)$$

It is smooth, unimodal and symmetric. The goal is to find the global minimum $\min(f_1) = f_1(0,0,0) = 0$.

The second De Jong test function is the Rosenbrock's valley:

$$f_2(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2 \;, \; -2.048 \le x_i \le 2.048 \qquad (2)$$

It has a very narrow ridge. The tip of the ridge is very sharp and it runs around a parabola. The goal is to find the global minimum $\min(f_2) = f_2(1,1) = 0$.

The third De Jong test function is the Step function:

$$f_3(x_1, x_2, x_3, x_4, x_5) = 5 \cdot i \sum_{i=1}^{5} \lfloor x_i \rfloor, \; -5.12 \le x_i \le 5.12 \qquad (3)$$

It is discontinuous and representative of the problem of flat surfaces. The goal is to find the global minimum $\min(f_3) = f_3([-5.12,-5),\ldots,[-5.12,-5)) = 0$.

The fourth test function is the Himmelblau function:

$$f_4(x_1, x_2) = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2 \;, \; -6 \le x_i \le 6 \qquad (4)$$

It is a multimodal function with four distinct minima. The goal is to find the global minimum $\min(f_4) = f_4(3,2) = 0$.

For both the classic GA and the proposed GA with restartings the same set of GA's operators and parameters were used in order to have a fair comparison of their efficiency and performance. The representation used for the genomes of the genetic population is the classic binary string. As far as the reproduction operator is concerned, the classic biased roulette wheel selection was used. The crossover operator used is uniform crossover (with crossover probability equal to 0.9), while the mutation operator is the flip mutator (with mutation probability equal to 0.001. The size of the population both for the classic GA and the proposed GA with restartings was set to 50, while the percentage of the genomes passed to the next initial population by each restarting procedure equals 20% (in our case 10 genomes, i.e. c=10). Except for that, both GAs used linear scaling and elitism.

Both GAs were implemented using the C++ Library of Genetic Algorithms GAlib [10] and especially the GASimpleGA class for the implementation of the GAs (non-overlapping populations) and the GABin2DecGenome class for the binary string genomes (an implementation of the traditional method for converting binary strings to decimal values). All the experiments were carried out on a Intel Pentium IV 2.7GHz PC with 256 MB RAM.

The comparison of the algorithms is based on two criteria. For each one of the four test functions two specific quantities are taken into consideration. The first one is the value achieved by the fitness function of each algorithm. We measure the number of fitness function evaluations made by each algorithm in order the value of the fitness function to overcome a predefined threshold. The second quantity is the number of fitness function evaluations. We measure the best value of the fitness function achieved by each algorithm for a specific number of fitness function evaluations.

In the following table the performance and efficiency of both the classic GA and the proposed GA with restartings is shown for the first De Jong function.

TABLE I
EXPERIMENTAL RESULTS FOR THE FIRST DE JONG FUNCTION

Performance Criterion	Classic GA	GA with restartings: Every fifty generations	Every sixty generations	Every seventy generations	Every eighty generations	Every ninety generations
Fitness function value		Number of evaluations				
<1.0e-10	20768	21245	20524	18822	20608	21085
<1.0e-16	35290	36044	34253	32017	34260	35415
Number of evaluations		Fitness function value				
10000	1.69e-06	1.06e-05	2.67e-06	1.01e-06	8.52e-06	1.26e-06
20000	8.99e-10	4.33e-09	7.32e-10	8.88e-11	5.91e-10	2.97e-09

In the following table the performance and efficiency of both the classic GA and the proposed GA with restartings is shown for the Second De Jong function.

TABLE II
EXPERIMENTAL RESULTS FOR THE SECOND DE JONG FUNCTION

Performance Criterion	Classic GA	GA with restartings: Every ten generations	Every twenty generations	Every thirty generations	Every forty generaions	Every fifty generations
Fitness function value		Number of evaluations				
<1.0e-4	1238271	77218	34748	106398	88680	189813
<1.0e-8	Not able after 4000000	353265	156853	167083	225154	218781
Number of evaluations		Fitness function value				
50000	3.51e-02	2.36e-04	4.19e-05	4.45e-04	3.74e-04	1.51e-03
100000	2.09e-02	9.93e-05	8.86e-06	1.53e-04	9.05e-05	5.63e-04
200000	1.55e-02	1.69e-05	7.73e-09	9.69e-09	7.09e-08	6.45e-08

In the following table the performance and efficiency of both the classic GA and the proposed GA with restartings is shown for the third De Jong function.

TABLE III
EXPERIMENTAL RESULTS FOR THE THIRD DE JONG FUNCTION

Performance Criterion	Classic GA	GA with restartings: Every ten generations	Every twenty generations	Every thirty generations	Every forty generaions	Every fifty generations
Fitness function value		Number of evaluations				
<= 1	8692	7061	8116	8433	6517	6466
= 0	15339	13236	14297	13107	14431	14215
Number of evaluations		Fitness function value				
6000	2.2	1.7	1.4	1.2	1.6	1.6
12000	1.1	0.6	1.1	0.9	1.1	0.8

In the following table the performance and efficiency of both the classic GA and the proposed GA with restartings is shown for the Himmelblau function.

TABLE IV
EXPERIMENTAL RESULTS FOR THE HIMMELBLAU FUNCTION

Performance Criterion	Classic GA	GA with restartings: Every ten generations	Every twenty generations	Every thirty generations	Every forty generaions	Every fifty generations
Fitness function value		Number of evaluations				
<1.0e-4	114064	7779	8612	11401	16944	15925
<1.0e-8	178481	30576	31712	37852	22950	25547
<1.0e-12	Not able after 4000000	2225732	118509	141107	96773	91626
Number of evaluations		Fitness function value				
10000	2.11e-02	1.37e-05	2.43e-05	3.91e-04	3.02e-03	1.57e-03
50000	2.02e-02	5.29e-09	6.67e-09	2.09e-09	1.19e-09	2.62e-09
100000	9.71e-03	1.34e-09	1.64e-10	2.24e-10	7.12e-13	7.11e-13

From the above tables, one can easily come to the conclusion that the proposed technique enhances significantly the performance of the classic GA. The restarting procedure manages to achieve a better global exploration of the solution space while executing fewer fitness function evaluations. This is more obvious especially when the function to be optimized has many local optima like the second De Jong function and the Himmelblau function.

IV. CONCLUSIONS AND FUTURE WORK

As experimental results show, the proposed technique manages to significantly enhance the performance of the classic GA, especially in optimizing "hard" functions with many local optima. It would be very interesting to check the efficiency and performance of the proposed GA with restartings to other difficult test functions and NP-Hard problems like the TSP problem. These issues will be the main scope of our future work.

REFERENCES

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Mass.: Addison-Wesley, 1989.
[2] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., N. Y.: Springer-Verlag, 1996.
[3] M. Mitchell, *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*, Cambridge, Massachusetts, London, England: A Bradford Book, The MIT Press, 1998.
[4] H. A.. van der Vorst, "Computational Methods for large Eigenvalue Problems", in *Handbook of Numerical Analysis*, vol. 8, P. G. Ciarlet and J. L. Lions, Eds. Amsterdam: North-Holland (Elsevier), pp. 3-179, 2002.
[5] Y. Saad, *Numerical methods for large eigenvalue problems*, Manchester, UK: Manchester University Press, 1992.
[6] S. G. Petition, "Parallel subspace method for non-Hrmitian eigen-problems on the connection machine (CM2)", *Applied Numerical Mathematics*, vol.10, pp. 19-36, 1992.
[7] D. C. Sorensen, "Implicit application of polynomial filters in a k-step Arnoldi method", *SIAM J. Matrix Anal. Applic.,* vol. 13(1), pp. 357-385, 1992.
[8] K. De Jong, "*A*n analysis of the behaviour of a class of genetic adaptive systems", PhD thesis, University of Michigan, 1975.
[9] D. M. HimmelBlau, *Applied Linear Programming*, McGraw-Hill, 1972.
[10] GAlib - A C++ Library of Genetic Algorithm Components, Matthew Wall, Massachusetts Institute of Technology (MIT). Available: http://lancet.mit.edu/ga/