Volume 2, No. 6, Nov-Dec 2011
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

A non-revisiting Genetic Algorithm with adaptive mutation for Function Optimization

Saroj Ratnoo *, Devraj Kamboj
*Associate Professor, 2Student, M Tech (CSE)
Deptt. Of Computer Science & Engg., GJUS&T, Hisar, Haryana.
ratnoo.saroj@gmail.com
devraj.kamboj@gmail.com

Abstract- Genetic Algorithm (GA) is a robust and popular stochastic optimization algorithm for large and complex search spaces. The major disadvantages of Genetic Algorithms are premature convergence and revisits to individual solutions from search space. In other words Genetic algorithm is a revisiting algorithm that leads to duplicate function evaluations which is a clear waste of time and computational resources. In this paper, a non-revisiting genetic algorithm with adaptive mutation is proposed for the domain of function optimization. In this algorithm whenever a revisit occurs, the underlined search point is replaced with a mutated version of the best/random (chosen probabilistically) individual from the GA population. Moreover, the suggested approach is not using any extra memory resources to avoid revisits. To test the power of the method, the proposed non-revisiting algorithm is evaluated using nine benchmarks functions. The performance of the proposed genetic algorithm is superior as compared to simple genetic algorithm as confirmed by the experimental results.

Keywords: Function optimization, Genetic Algorithm, Non-Revisiting, adaptive mutation.

I. INTRODUCTION

Developing new optimization techniques is an active area of research and Genetic Algorithm (GA) is a relatively new stochastic optimization algorithm pioneered by Holland [1]. A GA is capable of finding optimal solutions for complex problems in a wide spectrum of applications due to its global nature. A GA is an iterative procedure that maintains a population of structures that are candidate solutions to the specific problem under consideration. During each temporal increment (called a generation), the structures in the current population are rated for their effectiveness as the problem solutions through a fitness function, and on the basis of these evaluations, a new population of candidate solutions is formed using specific genetic operators such as reproduction, crossover, and mutation[2]. In fact, a GA mimics the natural principle of survival of fittest. A fitness proportionate selection and GA operators ensures the better and better fit solutions to emerge in successive generations. However, GAs is not without limitations. Two such problems are: 1. premature convergence i.e. many a times a GA converges to some local optimal solution. 2. Redundant function evaluations.

A simple genetic algorithm do not memorizes the search points or solutions to the problem that it visits in its life time and it revisits lots of search points generating duplicate solutions which results into redundant fitness computations. Here, a revisit to a search position x is defined as a re-evaluation of a function of x which has been evaluated before. The problem of revisit is all the more severe towards the end of a GA run. In many domains the fitness evaluation is computationally very expensive and lots of time is wasted in revisiting the parts of the search space and duplicate function evaluations. The problem of redundant function evaluations has been addressed by several researchers by providing GA a long term memory i.e. the GA stores all the search points visited and their corresponding fitness into some data structure. In such approaches every time a new search point is produced by GA, before actually computing its fitness, the memory of GA is looked into and if this search point exists, its fitness is not recomputed. If the new solution is not in the memory, its fitness is computed and appended to the memory. The problem with all such approaches is that now GA spends a significant amount of its time in memory look ups and a very large data structure is required as supplemented GA memory. Binary search trees, Binary partition tress and heap structures have been used as GA memory [3]. It is not uncommon for a GA to run for thousands of generations with a population of hundreds of individuals. If we assume a GA with 100 individuals and 5000 generations, we shall need a data structure that can store 250000 problem solutions and that is when we assume half the individuals produced are duplicates.

The GA shall require 500000 look ups to avoid redundant fitness computation. GAs is already considered slow as compared to other optimization techniques and these approaches further slow down GA's performance. Clearly this method is successful only in the domains where fitness computations are significantly larger than the memory look ups and not suitable at all for domain of function optimization where fitness evaluation is relatively less expensive.

In this paper, we propose an improved GA with adaptive mutation operator to avoid revisits and redundant fitness evaluations. This GA has the elitist approach and retains the best individual in every new population. A look up for revisits is made only in the current population along with the population of previous generation. If any individual produced is found duplicate, it is replaced probabilistically with a mutated version of the best individual or of a random individual. The mutation operator is adaptive in the sense that its power of exploration decreases and power of exploitation increases with the number of generations.

The proposed approach demonstrates that the duplicate removal introduces a powerful diversity preservation mechanism which not only results in better

CONFERENCE PAPER
International Conference on Issues & Challenges in Networking, Intelligence & Computing Technologies
Organized by Krishna Institute of Engineering and Technology (KIET) Ghaziabad, India

final-population solutions but also avoids premature convergence. The results are presented for nine benchmark functions and illustrate the effectiveness of duplicate removal through adaptive mutation. The results are directly compared to a simple GA which is not removing duplicate.

Rest of the paper is organized as below. Section II describes the related work. The proposed non revisiting Genetic algorithm which removes duplicate individuals through adaptive mutation is given in section III. Experimental design and results are enlisted in section IV. Section V concludes the papers and points to the future scope of this work.

II. RELATED WORK

Mauldin [4] was among the first ones who enhanced the performance by eliminating duplicate genotypes during a GA run. Mauldin used a uniqueness operator which removes duplicate and similar genotypes in an evolving population. This operator only allowed a new child ***x*** to be inserted into the population if ***x*** was greater than a Hamming-distance threshold from all existing population genotypes. Davis [5] also showed that, by using binary coded GA for a comparable number of child evaluations, that removes duplicates in the population has superior performance.

Eshelman and Schaffer [6] later re-confirmed this observation. Eshelman and Schaffer used new operators and selection-based innovations for their test. They checked the performance by preventing duplicates. They used thirteen mathematical test problems like epistatic problem for their test. Their results showed that the prevention of duplication of individuals reduced the number of evaluations required to find the global optimum. Povinelli and Feng [7] also work on duplicate individuals. They use a small hash table to store all visited individuals. When this table is full, it is thrown away and a larger table is used. Kratica [8] works on visited individuals by using a small fixed size cache which store all visited individuals. When this cache is full, an old entry is thrown away to make place for a new entry using the least-recently-used strategy. They confirmed the improvement to GA by adding the cache, but they do not store all the individuals and thus do not guarantee non-revisiting.

Friedrich *et al.* [9] analyze that an evolutionary algorithm with a population greater than 1 using uniform bit mutation but no crossover has better performance by duplicate removal. He observed that the duplicate removal method changes the time complexity of optimization. Ronald [10] used the Hash table to reduce the number of comparisons. However, he compared a child only with the current population. So it does not guarantee for non-revisiting. Yuen and Chow used a novel binary space partitioning tree to eliminate the duplicate individuals [11].

Saroj et al. (2010) used a heap structure to avoid the redundant fitness evaluations in domain of rule mining. Their approach proved to be effective for large datasets where fitness evaluation was computationally expensive [12].

III. PROPOSED NON-REVISITING GA WITH ADAPTIVE MUTATION

Non-revisiting algorithm is the one which do not visit the search points already visited. The improved GA with non-revisiting algorithm and adaptive mutation has to perform some extra steps than a simple GA. These steps are used to found the duplicate individuals. If any duplicate individual is found then it is mutated and reinserted in the current population. The duplicates are looked with respect to current and the previous generation only. There is a special condition that the best individual is preserved and not mutated. The flow chart and algorithm for the proposed GA is given in Fig. 1 and Fig 2 respectively.

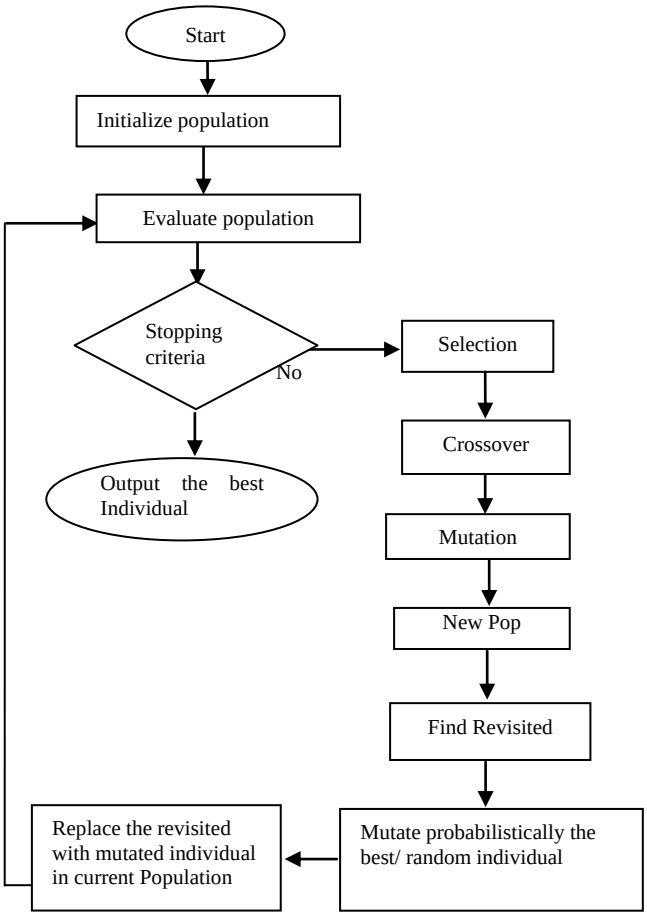

Figure 1: Step by Step Procedure for the Improved GA

The mutation applied is Gaussian adaptive mutation. The Gaussian function generates a random number around 0 mean. The formula for mutation is as follows.

$$mscale = mscale - mshrink * mscale * (current\_generation / total\_generations)$$

$$x(i) = best\_x \pm (gaussian\_rand * mscale)$$

The amount of mutation is controlled by the two parameters mscale and mshrink. Here, mscale represents the variance for mutation for the first generation and mshrink represents the amount of shrink in mutation in successive generations. The mutation scale decreases as the number of generation increase. It is clear from the above formulae that such kind of mutation is exploratory towards initial runs and exploitative towards the final runs

CONFERENCE PAPER
International Conference on Issues & Challenges in Networking, Intelligence & Computing Technologies
Organized by Krishna Institute of Engineering and Technology (KIET) Ghaziabad, India

of the GA. We have kept the mscale as 1.0 and mshrink equals to 0.75.

```
1. Begin
2. Initial condition: Initial population:- old-pop=[], new-pop[], Stopping Flag:- SF=0, Best Fit=0, Visiting Flag:- VF[]=0
3. Initialize the population in old-pop
4. Evaluate the fitness of old-pop and calculate the overall best individual up to the current generation.
5. If(stopping criteria=yes)
 5.1 SF=1 //set stopping flag
 5.2 Output the best individual
 5.3 Stop
 Else

 5.4 Copy the best individual into new-pop

 5.5. Perform a GA step by applying GA operators i.e. selection, crossover and mutation.

 5.6 Maintain the old population and store the newly generated individuals in the new pop.

 5.7 For (i=1 to pop-size)

 check revisits within the new-pop and with respect to old-pop
 5.8 If revisit && not best_individual
 5.9 VF[i]=1
 5.10 For(i=1 to pop-size)
 5.11 If VF[i]==1
        New-pop[i] =mutated (new-Pop[i])

 5.12 ld-pop=new-pop
Go to step 2
```

Figure 2: Algorithm for improved GA with no-revisit

The proposed non-revisiting GA with adaptive mutation has three key strengths.

a. It automatically assures maintenance of diversity and prevents premature convergence. Most of the individuals in a GA population are guaranteed to be different. By nature, it is impossible for a population to consist of one kind of individual only.
b. It doesn't require large data structure to store the individuals for to do a look up for duplicates and only uses the previous and current populations which are anyway available.
c. It probabilistically takes the advantage of the best individuals and converges faster without suffering problem of convergence.

IV. EXPERIMENTAL RESULTS

A. Test function set:

Nine Benchmarks functions in four dimensions used to test the proposed GA are as follows.

a. Rastrigin's function
b. Sphere function
c. Generalized Rosenbrock function
d. Generalized Rastrigin function
e. Griewank's function
f. Ackley function
g. Rotated Griewank's function
h. Rotated Weierstrass's function
i. Branin function

All these function are shown in detail in the Appendix. The first four functions are unimodal functions; the next six are multimodal functions designed with a considerable amount of local minima. The eighth function and nine are rotated multimodal functions. The improved GA is implemented in MATLAB. A comparison is made between a simple GA and the proposed GA on the basis of mean fitness and the best fitness over the generations. The best fitness is the minimum score of the population [13]. Both the GAs stop when there is no improvement in the best score over last fifty generations. The population size is kept at 20 for all the functions and, the crossover and mutation rates are equal to 0.6 and 0.01 respectively.

The normal mutation rate is kept low as adaptive mutation to remove duplicates is also applied. The best individual or a random individual is mutated with equally likely (probability = 0.5) to replace the revisited points in the new population. We have used real encoding, linear scaling, roulette wheel selection, heuristic crossover and Gaussian mutation. The mean fitness and the best fitness of the final populations are shown in Table 1. These results are averaged over 20 runs of the GAs. A graph comparing the performance of the proposed GA and simple GA for Rastrigin's function is shown in Fig 3.

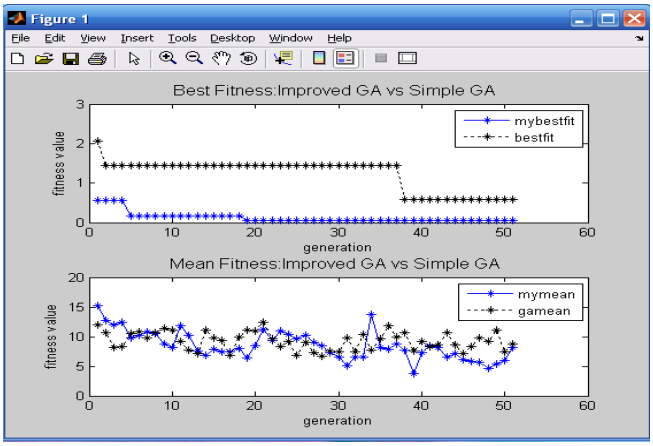

Figure 3: Performance comparison between improved GA and simple GA

Table1: A comparison of mean and best fitnesses for the nine benchmark functions

BENCHMARKS FUNCTION	BEST FITNESS		MEAN FITNESS	
	SGA	Improved GA	SGA	Improved GA
Rastrigin's function	0.589	0.053	6.684	3.817
Sphere function	0.002	0.001	0.279	0.105
Generalized Rastrigin function	0.079	0.002	6.474	5.414
Griewank's function	0.001	0.001	0.077	0.021
Generalized Rosenbrock function	0.031	0.017	28.22	21.34
Ackley function	0.024	0.013	1.426	0.936
Rotated Griewank's function	0.001	0.001	0.041	0.017
Rotated Weierstrass's function	0.212	0.083	1.531	1.321
Branin function	0.146	0.054	1.134	1.011

The accuracy of our approach is better for all the nine benchmark functions. It is quite clear from the results that the performance of the non-revisiting GA with adaptive mutation is better than the simple GA.

V. CONCLUSION

In this paper, a novel non-revisiting GA with adaptive mutation is proposed and tested in the domain of function optimization. Though new improved GA may not completely eliminate the revisits to the same points in the

search space and redundant function evaluation, it guarantees enough diversity to avoid the problem of premature convergence. The envisaged approach achieves better accuracy without much overheads of searching time for duplicates individuals and large data structures to serve as the long term memory for a GA.

The mechanism of a probabilistic adaptive mutation provides the much required balance between exploration and exploitation along with faster convergence to the optimal. It is exploratory in the initial runs of GA and exploitative towards the final runs of GA. More the number of generations of the GA, smaller will be the change in the new individual that replaces the revisited search point. The experimental results are very encouraging and show that the improved GA is clearly superior to its conventional counterpart. The adaptation of the current approach is underway for the domain of rule mining in the field of knowledge discovery.

Appendix

A. Benchmarks Functions:

a. Rastrigin's function [14]:
f1(x) = $10x+\sum_{i=1}^{D}[x^2-10\cos(2\pi x)+10]$
Where x $\in[-5.12, 5.12]^D$
Min f1(x) = f1 ([0, 0….0]) =0

b. Sphere function [14]:
$f2(x) = \sum_{i=1}^{D} x^2$ where x $\in[-5.12, 5.12]^D$
Min f2(x) =f2 ([0, 0….0]) =0

c. Generalized Rosenbrock function [14]:
f3(x) = $\sum_{i=1}^{D-1}[100(x_{i+1}-x_i^2)^2 + (x_i-1)^2]$
Where x$\in$ $[-5.12, 5.12]^D$
Min f3(x) = f3 ([1, 1….1]) =0
\

d. Generalized Rastrigin function [14]:
f4(x) = $\sum_{i=1}^{D}[x^2-10\cos(2\pi x)+10]$
Where x $\in[-5.12, 5.12]^D$
Min f4(x) = f4 ([0, 0… 0]) =0

e. Griewank's function [14]:
f5(x) = $\frac{1}{4000}\sum_{i=1}^{D} x^2-\prod_{i=1}^{D}\cos\frac{xi}{\sqrt{i}}+1$
Where f5(x) $\in$ $[-5.12, 5.12]^D$
Min f5(x) = f5 ([0, 0… 0])

f. Ackley function [14]:
f6(x) = -20 exp $(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2})$-exp $(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i)$ +20+e
where x $\in$ $[-5.12, 5.12]^D$
Min f6(x) = f6 ([0, 0… 0]) = 0

g. Rotated Griewank's function [14]:
f7(x) = $\frac{1}{4000}\sum_{i=1}^{D} z^2-\prod_{i=1}^{D}\cos\frac{zi}{\sqrt{i}}+1$
Where z=xM , f7(x) $\in$ $[-5.12, 5.12]^D$
Min f7(x) = f7 ([0, 0… 0])

h. 8. Rotated Weierstrass's function [14]:
f8(x) = $\begin{bmatrix} \cdots & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix}$ //D2=D^2

i. Branin function [14]:
f9(x) = $(x_2-\frac{5}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6)^2+10(1-\frac{1}{8\pi})\cos x_1+10$
Where x $\in$ [-5.12, 5.12]
Min f9(x) = f9 ([-3.142, 12.275]) =
f9 ([3.142, 2.275])

VI. REFERENCES

[1]. J. H. Holland, "Adaptation in natural and artificial systems". Ann Arbor, MI: Univ. of Michigan Press, 1975.

[2]. K. M. Bakwad, S. S. Pattnaik, B. S. Sohi, S. Devi, B. K. Panigarhi and V, R. S. Gollapudi, "Multimodal Function Optimization using Synchronous Bacterial Foraging Optimization Technique". IETE Journal of Research, vol. 56, pp. 80-87, April 2010.

[3]. Goldberg D.E.: "Genetic Algorithms in Search, Optimization and Machine Learning". New York: Addison-Wesley Publishing Company, Inc. MA, 1989.

[4]. M. L. Mauldin, "Maintaining diversity in genetic search," in Proc. National Conf. Artif. Intell., 1984, pp. 247–250.

[5]. L. Davis, Handbook of genetic algorithms. New York: Van Nostrand Reinhold, 1991 L. Eshelman and J. Schaffer, "Preventing premature convergence io genetic algorithms by preventing incest," in Proceedings of the Fourth International Conference on Genetic Algorithms,R. Belew and L. Booker, Eds. 1991, pp. 115-122**,** Morgan Kaufmann Publishers, San Mateo, CA.

[6]. R. J. Povinelli and X. Feng, "Improving genetic algorithms performance by hashing fitness values," in Proc. Artif. Neural Netw. Eng., 1999, pp. 399–404.

[7]. J. Kratica, "Improving performances of the genetic algorithm by caching," Comput. Artif. Intell., vol. 18, no. 3, pp. 271–283, 1999.

[8]. T. Friedrich, N. Hebbinghaus, and F. Neumann, "Rigorous analyses of simple diversity mechanisms," in Proc. Genetic Evol. Comput. Conf., Jul. 2007, pp. 1219–1225.

[9]. S. Ronald, "Duplicate genotypes in a genetic algorithm," in Proc. IEEE Int. Conf. Evol. Comput., 1998, pp. 793–798.

[10]. S. Y. Yuen and C. K. Chow, "A non-revisiting genetic algorithm," in Proc. IEEE Congress. Evol. Comput., 2007, pp. 4583–4590.

[11]. Saroj, Kapila, Dinesh Kumar, Kanika, A Genetic Algorithm with entropy based probabilistic initialization and memory for automated rule mining, In Natrajan Meghanathan, Brijesh Kumar Kaushik and Dhinaharan Nagamalai, Advances in Computer science and Information technology, CCSIT-2011, Bangalore, India, Jan-2-4, 2011, vol. 131, pp. 604-613.

[12]. S. Y. Yuen and C. K. Chow, "A Genetic Algorithm That Adaptively Mutates and Never Revisits" IEEE Transaction on Evolutionary computation,vol 13, april 2009, pp 454-472.

[13]. X. Yao, Y. Liu, and G. M. Lin, Evolutionary programming made faster," IEEE Trans. Evol. Compute., vol. 3, no. 2, pp. 82–102, Apr.

CONFERENCE PAPER
International Conference on Issues & Challenges in Networking, Intelligence & Computing Technologies
Organized by Krishna Institute of Engineering and Technology (KIET) Ghaziabad, India