Design for Scientific RENAISSANCE

JACSTR

# A Practical Method for the Analysis of Genetic Algorithms

Imad Fakhri Al Shaikhli
Department of computer science KICT- IIUM,
Kuala Lumpur, Malaysia.
imadf@iium.edu.my

Article Info

Received: 10 July 2011
Accepted: 05 August 2011
Published online: 01 September 2011



**ABSTRACT**

In this research, a practical method was proposed for the analysis of genetic algorithms for the sake of proving their efficiency, showing the weak points, and checking the performance by answering most of the important questions related. The experimental result done on some of the known genetic algorithms show the usefulness of the proposed method in improving the quality of genetic algorithms.

***Keywords***: **Genetic Algorithms, Soft Computing, Evolutionary Computations, Knapsack Ciphers, Practical Analysis.**

## 1. Introduction

Genetic Algorithms still lack the strong theoretical base which can support them, except from the first work by Holland (Hoolland, 1992) and some few researches as it is stated in (Michalewicz, 1996). Although some research has been done to prove this hypothesis (Building Block Hypothesis) (Bethke, 1980) for most nontrivial applications we rely mostly on empirical results. Li et al. (2007) claimed that the theoretical basis of genetic algorithm is weak, especially on algorithm convergence.

In the absence of theory, there was a need to develop a method for examining the workability of any GA and at the same time to analyze it and to answer some questions like:

a. Are we sure that the GA is working the way it was planned?
b. Is it working better than a random search?
c. What is the effect of each of the GA operators used and why they were used?
d. What is the effect of any new technique (if any) used in the GA?
e. What is the ideal set of parameters to be used in the GA?

Review of Genetic Algorithms Genetic Algorithms (GAs) was first suggested by John Holland in the early seventies (Hoolland, 1992). Over the last 30 years they have been used to solve a wide range of search, optimization and machine learning problems. As the name indicates, these algorithms attempt to solve problems by modeling the way in which biological genetic process seems to operate.

A good treatment of the nature and the use of Gas can be found in (Michalewicz, 1996; Haupt, 2004; Affenzeller, 2009). A GA begins with a randomly selected population of chromosomes represented by strings. The GA uses the current population of strings to create a new population such that the strings in the new population are on average better than those in current population (the selection depends on their fitness value).

Three processes which have a parallel in biological genetics are used to make the transition from one population to the next, (selection, mating, and mutation) as shown in Fig.1.

The selection process determines which strings in the current population will be used to create the next generation.

This is done by using a biased random selection methodology in a way that the best strings in the population have the greatest chance of being selected.

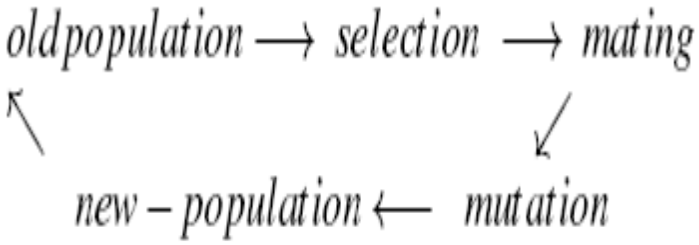


Fig.1. the basic genetic algorithm cycle

The mating process determines the actual form of the strings in the next generation. Here two of the selected parents are paired, if the length of each string is r, then a random number is selected between 2 and r, call it s, then we will combine bits 1; : : : ; (s-1) of the first parent with bits s; : : : ; r of the second parent to produce one new string(offspring),and vice-versa for a second new string.

A fixed, small mutation probability is set at the start of the algorithm, then bits in all the new strings are subjected to change based on this probability.

## 2. Proposed method

The method has the following steps:

a. Creation of a test-bed of sample problems for the GA.
b. Comparison with random search
   - Disable all GA operators of the GA.
   - Disable the selection mechanism.
   - In each generation we do the following:
     – Generate a random population.
     – Apply the new techniques used in the GA or just check the solution.

- Compare the results with those of the GA to check the contribution of the GA.

c. Checking the contribution of the operators. During this phase disable the genetic operators one by one and run the algorithm, then compare both results. This will show the role of the operators used and how much each contributes to the success of the GA.
d. Checking the contribution of new techniques. In this phase disable the new techniques that are used in the GA, one by one and apply the GA in each time on the same test-bed and then disable all the new techniques together and see the results. This phase will show the role and effect of each of the new techniques and whether it contributes to the GA or not.
e. Parametric study After choosing the final version of the GA, do a parametric study to choose the best set of parameters for it (population size, number of generations, probability of crossover and mutation) by running the GA with different sets of parameters until finding the best of them.

## 3. Experimental Results

For testing the validity and robustness of the proposed method, a number of published papers were chosen to be tested using the proposed method. The results of the first and the second algorithm represent the average number of generations needed for the GA to find the solution, this number is the average of applying the GA on 10 cipher, twice for each cipher, while for the third algorithm, it represent the same with the deviation from the optimal schedule.

### 3.1 Analyzing a GA for solving the multiplicative knapsack cipher

The cycle of the algorithm to be analyzed contains the following processes (Imad and Sahasrabuddhe, 1998);

a. Selection.
b. One-point crossover.
c. Multi-point crossover.
d. Mutation.
e. Target expansion techniques.
f. Injection of random chromosomes
g. Evaluation

The analysis is done on knapsack of size 25 and population of size 200.

a. The best result of this GA before the analysis was 244 generations.
b. Random Search: The random search was not able to find the solution even after 8000 generations for any of the ciphers.
c. Checking the operators: Results after disabling the operators one by one were:
   - Disabling the one point crossover: 320 Generations.
   - Disabling the multi-point crossover: 239.5 generations.
   - Disabling both (one point and multipoint crossover: 557 generations.

- Disabling mutation: no solution even after 1000 generations.

d. Checking new techniques
   - The GA was executed with only repair (by disabling checkpairs and findsol) and the result was: 470 generations.
   - When the GA was executed with only findsol (by disabling repair and checkpairs) and the result was: 674 generations.
   - Executing GA with repair and checkpairs (by disabling findsol) and the result was: 351.45 generations.

Parametric study was done by running the GA with different sets of parameters and we found that the following set is the best:

a. Probability of crossover = 0.85.
b. Probability of mutation: K2 = 0.2 and K4 = 0.15.

One more thing needed was about findsol, what is the percentage of its expansion to the target set and on which type of chromosomes (Which set of HD) it works better? For answering these questions the following was done: We ran a program that generates all members (chromosomes) of the search space and applied findsol on them and wrote down those of them that can be solved by findsol with their HD from the target. We did this on knapsack of size 15 and 25 and the results are shown in the following Tables:

Table 1: applying findsol on the search space of knapsack of size 15

| *Hamming Distance* | *total no. of* chromosomes | *No. of solvable* chromosomes |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 15 | 0 |
| 2 | 105 | 0 |
| 3 | 455 | 8 |
| 4 | 1365 | 38 |
| 5 | 3003 | 84 |
| 6 | 5005 | 142 |
| 7 | 6434 | 157 |
| 8 | 6434 | 89 |
| 9 | 5005 | 25 |
| 10 | 3003 | 2 |

Table 2: applying findsol on the search space of knapsack of size 25

| *Hamming Distance* | *total no. of* chromosomes | *No. of solvable* chromosomes |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 25 | 13 |
| 2 | 350 | 23 |
| 3 | 2300 | 34 |
| 4 | 12650 | 40 |
| 5 | 53130 | 53 |
| 6 | 177100 | 81 |
| 7 | 480700 | 97 |
| 8 | 1081575 | 104 |
| 9 | 2042975 | 71 |
| 10 | 3268760 | 31 |
| 11 | 4457400 | 6 |

### 3.2 Discussion on the results of the analysis

a. The multi-point crossover operator was not necessary, because when we disabled it, we got almost the same result, which means it is not contributing to the GA.
b. The one point crossover was necessary for the information exchange between chromosomes (building blocks) and that's why when we disabled it, the average number of generations needed to find the solution reached to 557 generations.
c. The mutation is an important operator, it prevent the population from being dominated by some chromosomes, by introducing new points (chromosomes) from the search space and that's why when we disabled it, the algorithm couldn't find the solution for all the cases even after 1000 generations for each case .
d. The new techniques we used (repair, checkpairs and findsol) were important to reach this level of performance (244 generations).
e. We found a suiTable set of parameters for our GA.
f. It has been confirmed that repair and checkpairs works better (with the selection mechanism and the fitness function) than findsol, although the expansion of findsol is bigger than that of repair. That is because the selection mechanism promote chromosomes which are closer to the target(as an integer value), and most of them will belong to the set of chromosomes of H.D1(repair) or H.D2 (checkpairs), while findsol works on chromosomes scattered in the search space.

### 3.3 Analysis of a GA for solving the Chor-Rivest knapsack cipher

a. The analysis was done on knapsack of size 25 with 12 ones in each vector, on 10 ciphers.

b. The analysis was done in two phases, the first one started with the first version of this GA reported in (Imad et al., 1999), the cycle of the algorithm to be analyzed contains the following processes:
   - Selection.
   - Swap operator.
   - Invert operator.
   - Target expansion techniques.
   - Evaluation

The average number of generations needed for this GA to find the solution was 636.35. The random search with only HD2 needed an average of 2444.9 generations to find the solution. Then, by comparing the two results (636.35 and 2444.9) and analyzing the results in details, we found that:

a. The result is not very far from the random search because it is one fourth of it.
b. The algorithm consumed a high number of generations to find the solution for specific chromosomes.

From the above, it's found that we need to use a crossover operator in our GA to give more support to the building blocks hypothesis. So HDxover crossover operator was presented.

By applying the new version of the GA (with HDxover) on the same test-bed, the average number of generations needed was 281.5 to find the solution, which is better than the previous one but still the algorithm consumed a high number of generations to find the solution for two of the 10 ciphers. This indicates that HDxover has contributed well to the algorithm because the average dropped from 636.35 to 281.5. By taking a close look to the two ciphers (where the algorithm failed to find the solution in less number of generations), we can see that it contains blocks of compact schemas (schema with consecutive number of the same bit).

On the other hand if we look at the way the HDxover works, we can see that it do, contribute to the building blocks hypothesis by exchanging information between chromosomes, but at the same time, it might destroy some of the compact schemata because of its way of working. That's why the improved 1-point crossover was introduced.

By adding the new crossover operator and applying the new version of our GA on the same test-bed, the average was 149.9 generations to find the solution and that for all the ciphers without any exception and without consuming high number of generations for certain chromosomes. The second phase of analysis was done on the new version of the GA, the analysis was as follows:

a. Random search: The random search was not able to find the solution before 2444.95 generations.
b. Checking operators
   - HDxover: By disabling the HDxover and running the GA, the solution was found in only five cases out of 20, although the algorithm worked for 1000 generation each time.

- reparable crossover: By disabling this operator, the solution was found in only 2 cases out of 20.
- Both HDxover and the reparable crossover: By disabling both crossover operators, the average number of generations was 1692.5.
- Swap: By disabling Swap operator but not injecting the population with random chromosomes in each generation, the average number of generations was 234.2. But by disabling both, the swap operator as well as the injection of random chromosomes, the algorithm was not able to find the solution in 18 cases out of 20.

c. Parametric study: By doing the parametric study, the following set of parameters was seen to give better results:
   - Probability of HDxover = 0.5
   - Probability of reparable crossover = 0.5
   - Probability of Swap 0.05
   - POPSIZE = 200

d. discussion on the results of the analysis
   - The analysis method was useful to give an insight about this algorithm and hence improving it.
   - The fact that the proposed GA (Imad et al., 1999) is doing a good job in analyzing the Chor-Rivest knapsack scheme by examining a very small fraction of the search space, was confirmed.
   - The analysis found that both crossover operators used in (Imad et al., 1999) were important and the reason behind it is that each of them is working in building a different kind of blocks (schemata).
   - One-point crossover use to build the compact schemata, while HDxover builds the scattered schemata.
   - Confirm that the swap operator (which is a kind of mutation) is an important operator for introducing new points (chromosomes) in the search space and for preventing the population from being dominated by certain chromosomes.
   - Found that the invert operator was not necessary in (Imad et al., 1999) and that the injection mechanism was better.
   - A suiTable set of parameters for this GA were found.

### 3.4 Analyzing the Multiprocessor Scheduling Problem (MSP) GAs

The analysis is done on the following algorithms:

a. BEL (Bauer et al., 1995)
b. AD (Ahmad and Dhodhi, 1996)

On Directed Acyclic Graphs (DAGs) of size 32 with number of processors, 2,3,4,6,8,10 and 12 and with different number of edges for each number of processors, the total number of these DAGs were 13.

c. Random search: The random search deviation was as in Table 3:

Table 3: Results of BEL, and AD using random search

| *algorithm* | *relative deviation* |
|---|---|
| BEL | 0.574 |
| AD | 0.047 |

d. Checking the operators

- Crossover: By disabling the crossover operator from each of the algorithms and let the algorithm run with the remaining parts, the results are shown in Table 4:

Table 4: Results of BEL, and AD after disabling crossover

| *algorithm* | *# of generation* | *relative deviation* |
|---|---|---|
| BEL | 44.53 | 0.689 |
| AD | 36.61 | 0.0603 |

- Mutation: Disabling mutation from the algorithms showed the results as in Table 5.

Table 5: Results of BEL, and AD after disabling mutation

| *algorithm* | *# of generation* | *relative deviation* |
|---|---|---|
| BEL | 17.76 | 0.292 |
| AD | 5.6 | 0.0085 |

## 4. Conclusion

The proposed practical analysis method proved to be a useful tool for analyzing genetic algorithm and helped in providing an insight by answering the questions related to its performance, like the superiority of the GA to the heuristic algorithms or random search, the effect and contribution of each of the operators, and some other questions, which will help researchers to develop a more robust, efficient, and realistic genetic algorithms.

## References

Affenzeller, M., (2009). Genetic Algorithms and genetic programming: Modern concepts and practical applications.Boca Raton: CRC Press.

Ahmad, I. and Dhodhi, M.K., (1996). Multiprocessor scheduling in a genetic paradigm. Parallel Computing, pages 395–406.

Bauer, T., Ejlerse, O. and Holm, N. G. (1995). A genetic algorithm for multiprocessor scheduling. Master's thesis, Institute of Electronic systems, AALBORG University, Denmark.

Bethke, A.D. (1991). GAs as function optimizers, PhD thesis, University of Michigan, 1980. edited by Lawrence Davis. The Handbook of Genetic Algorithms. 1991.

David, E. Goldberg.(1989). Genetic Algorithms in Search, Optimization, and Machine learning. Addison-Wesley Publisheing, INC, Massachusetts.

Fuzeng Z. R., Li, Yongsheng Zhao and Lihua Song. Rough (2007). Sets in hybrid soft computing systems. Lecture Notes in Computer Science, 4632:35–44.

Haupt, R. L. (2004). Practical Genetic Algorithms. Hoboken, NJ: Willey.

Hoolland. J. (1992). Adaptation in Natural and Artificial Systems, MIT press, Massachusetts.

Imad, F.T.Yaseen and Sahasrabuddhe. H.V., (1998). Breaking multiplicative knapsack ciphers using genetic algorithms. In KBCS98, India.

Imad, F.T.Yaseen and.Sahasrabuddhe. H.V. (1999). A ga for the cryptanalysis of chor-rivest pkc. In ICCIMA99, India.

Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin Heidelberg,