Statistical Analysis And Performance Evaluation Of Genetic Algorithms

Bhargav Joshi
Department of Computer Science and Software Engineering
Auburn University
Auburn, USA
bhargav.csse@gmail.com

Armin Khayyer
Department of Computer Science and Software Engineering
Auburn University
Auburn, USA
azk0100@auburn.edu

Ye Wang
Department of Computer Science and Software Engineering
Auburn University
Auburn, USA
yewang@auburn.edu

***Abstract*—This paper contains a statistical analysis of the genetic algorithm with different variants of evolutionary computation techniques and crossover methods. Pairing various crossover methods with different evolutionary computation techniques result in different variants of the genetic algorithm. In the experiment, the genetic algorithm was subjected to solve the Schaffer F6 function with less than 4000 function evaluations. The paper shows what variants of the genetic algorithm have statistical equivalence through statistical techniques such as t-test and f-test that were performed on the number of function evaluations it took to solve the Schaffer F6 function. The average number of function evaluations is used to evaluate the performance of each variant to find the best performing variant of the genetic algorithm.**

Keywords—Genetic Algorithm, Class Equivalence, crossover

I. Introduction

Genetic Algorithm (GA) is an optimization technique to tackle mathematical problems. It is based on the concept of genetic reproduction and evolution observed in the nature. GA is inspired by the mechanism of natural selection, a biological process in which stronger individuals are likely be the winners in a competing environment [1]. GA is not considered as a mathematically guided algorithm. It uses the biological concept of evolution to evolve the obtained optima from generation to generation rather than using stringent mathematical formulation where the obtained optima is an end product containing the best elements of previous generations where the attributes of a stronger individual tend to be carried forward into the following generation [1].

Briefly put, GA follows the similar process of reproducing offspring from their parents' genes. Each offspring individual is expressed with its genotype and evaluated using its phenotype [2]. The process starts with (1) selection where the parents are selected, then (2) crossover where parents' genes are crossed over to create offspring's gene, then (3) mutation where the genes are slightly modified externally, and (4) computing fitness where the fitness of newly created offspring is calculated. After going through the whole process, GA creates a new generation of chromosomes where some or all parents are replaced by some or all offspring. The replacement is determined through different evolutionary computational strategies among which the following were used in the experiment.

- An elitist generational GA, (μ, μ) - EC
- A steady-state GA, (μ + 1) - EC
- A steady generational GA, (??? + 1) - EC
- A (μ + μ) – EC

Here, μ is defined as total population.

II. Methodology

A. Schaffer F6 function

The experiment was performed to solve Schaffer F6 function. Schaffer's F6 function has many local optima around the global optimum in the solution space [−100, 100]. Once getting into the local optimum, the particles hardly get out of local optima in the solution space [3]. The formula of Schaffer's F6 function is mathematically described as shown in figure 1.

$$\mathrm{F6}(\mathrm{x},\mathrm{y}) = -(0.5 + \frac{(sin\sqrt{x^2 + y^2})^2 - 0.5}{(1 + 0.001(x^2 + y^2))^2})$$

Figure 1: Schaffer's F6 function

B. Parent Selection Methods

Some of popular selection methods are:

- Proportional selection
- Linear Rank selection
- Tournament Selection

For tournament selection, k individuals are randomly selected from the population and the best out of k individuals is returned as the first parent. The process is repeated again for the second parent. Selection pressure increases with the increase in the value of k, and it decreases with the decrease in the value of k. The experiment had the binary tournament selection. Tournament selection is called binary in literature when k = 2 [4].

C. Genetic Algorithm Stratagies

To create evolved generations to reach the optima, the GA uses strategies to perform the selection of new population. Each strategy creates a variant of the genetic algorithm. The strategies implemented in the project are briefly described below.

D. An elitist generational GA

Generational GA uses (μ, μ) replacement strategy in which offspring completely replace parents [4]. In elitist generational GA, the fittest parent or a few best-fitted parents are chosen as elites that do not get replaced with offspring ensuring that the GA does not waste time re-discovering previously discarded partial solutions. Only one elite was chosen from each generation in the experiment.

E. A steady-state GA

In steady-state GA, two parents are chosen using any selection techniques and an offspring is created. The newly created offspring replaces the worst fit individual from the population. The steady-state genetic algorithm works in such a way that one or two members of the population are changed at a time [4]. In the experiment, a modification to steady-state GA was implemented with (μ + 1) strategy to ensure that offspring does not replace the worst fit individual in case if the offspring is worse than the worst fit individual.

F. *A steady generational GA*

A steady generational GA is a combination of steady-state GA and Elitist generational GA. In this strategy, an offspring is created from two parents chosen using any selection technique. The elite individuals are chosen. The offspring randomly replaces an individual from the population that is not an elite individual.

G. *A (μ + μ) – EC*

In this replacement strategy μ offspring are created and added to the population. After that the worst μ individuals from total population are removed.

H. *Crossover*

Crossover in Genetic Algorithms is an operator that takes 2 or more parents crosses them over and creates one or more offspring. In the experiment, three different crossover operators for all strategies were used. The implemented crossovers are explained below.

I. *Single point crossover (SPX)*

Since in the experiment, chromosome had length two therefore the SPX had only one option for splitting the chromosome. The SPX produces an offspring chromosome by taking the first gene of the first parent and the second gene of the second parent as shown in figure 2.

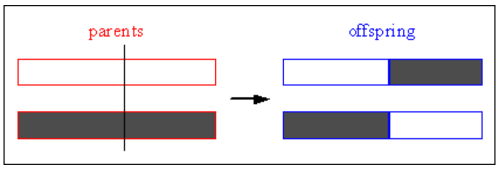

Figure 2: Single point crossover

J. *Mid-point crossover (MIDX)*

The second crossover used in this experiment is a mid-point crossover (MidX) with two probability values 1 and 0.5. When MidX probability is 1, each gene of a kid chromosome is produced by averaging the associated genes of both the parents. On the other hand, when MidX probability is 0.5, for producing each genome of a kid chromosome, we flip a coin if it’s tail we take average of the both parent’s genes, if not we just take the first parent’s gene for the first kid and second [4]parent’s gene for the second kid.

K. *Blend crossover – α (BLX- α), α = 0.0*

Blend Crossover with α = 0.0, also known as Flat crossover, assigns a random value between the genome of the two selected parents to each genome of an offspring. For example: Given two parents where X and Y represent a floating-point number: (1) Parent 1: X (2) Parent 2:Y (3) Offspring: random(X,Y).

III. Experiment

To compare the performance of different above-mentioned combinations of GA strategies namely Elitist Generational GA or (μ,μ), Steady State GA or (μ+1), Steady Generational GA, and (μ+μ) replacement strategy, each algorithm combination was evaluated at most 4000 times and run 30 times with 3 different types of crossover operators namely Single point crossover, mid-point crossover, and blend crossover with α = 0.0. These GAs had two main parameters to adjust, first one was the initial population size, and the second one was the mutation amount. A set of parameters that was discovered during the experiment which had the population size of 25 and the mutation rate of 0.01, showed promising results for all 12 variants of GA. It allowed us to compare the performance of each algorithm variant with exact same settings. The algorithms were implemented using python3. Microsoft Excel was also used to run the ANOVA test and the t-test on the results obtained from all algorithms to see if there were significant differences between performances of algorithms. The following section provides detailed results of the algorithms and their associated equivalent classes achieved using ANOVA test.

IV. Results

Table 1 to 4 show the mean and the standard deviation of the number of function evaluations until the desired output fitness of 0.9974 in F6 function is achieved. It also shows the class equivalence of each algorithm achieved by the ANOVA test.

Crossovers	SPX	MIDX	BLX-0.0
mean	219	116	146
Standard-deviation	198.91	76.02	77.67
Class equivalence	class one	class two	class two

Table 1: Elitist Generational GA

Crossovers	SPX	MIDX	BLX-0.0
mean	1919	1439	1815
Standard-deviation	932.13	976.18	1105.51
Class equivalence	class three	class four	class four

Table 2: (μ+1)

Crossovers	SPX	MIDX	BLX-0.0
mean	2055	2052	2054
Standard-deviation	937.46	911.09	983.24
Class equivalence	class four	class four	class four

Table 3: Steady generational GA

Crossovers	SPX	MIDX	BLX-0.0
mean	114	72	82
Standard-deviation	60.69	30.89	48.19
Class equivalence	class two	class five	class five

Table 4: (μ + μ) replacement

In order to find the equivalence classes, first an ANOVA test was ran with all 12 variants. The achieved P-value was quite smaller than the α < 0.05 value, therefore the null hypothesis (($\mu_{1,}\mu_2$ …, μ_n), where μ_i is the mean of the number of function evaluations for each algorithm) was rejected. It showed that the means were significantly different. To know which algorithms are different, an organized approach was taken to find the algorithms that vary from the rest. Another ANOVA test was run with 11 algorithms by discarding one algorithm and so on. An achieved p-value greater than α concludes that the algorithms' mean is the same, and they share a same equivalence class. On the other hand, if the achieved p-value is smaller than α, then this procedure needs to be run until two algorithms are remained. Once there were only two algorithms left to compare, The T test needs to be performed to find if they were statistically equivalent or not. In the t-test the t-stat was compared with the critical t value (in this case 1.7). For the t values greater than 1.7, H_0 is rejected and it is concluded that the mean of the number of function evaluations for the two algorithms are not the same therefore the two compared algorithms do not share the same equivalence class. We implemented the above-mentioned approach and found that there were five equivalent classes, we found that the second strategy and the third one share the same class, simply because of their mean and variance are closer in value. Also, the achieved p-value from the associated F-test was larger than 0.05. The detailed results, the codes, and the results of ANOVA and T tests are attached in the project file directory.

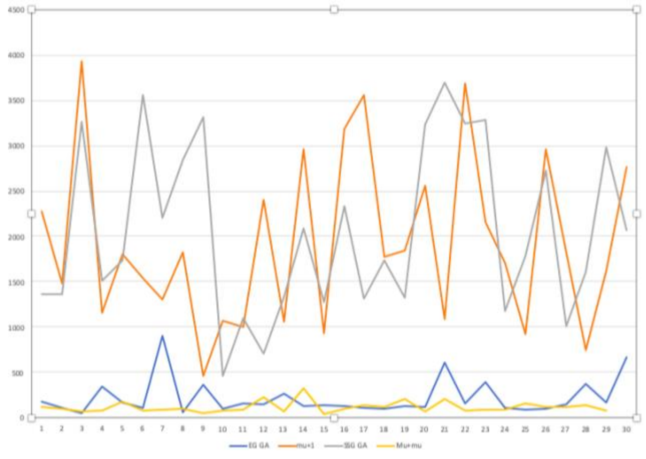

Figure 3: number of function evaluations over 30 runs for each strategy using SPX

Figure 3 shows the different function evaluation values for the 30 replication of each algorithm using the SPX crossover. As it is shown in the graph, the μ + μ algorithm works the best with this crossover, it can achieve the desired output in fewer number of function evaluations. However, we should consider the fact that in the (μ + μ) algorithm at each iteration we produce μ kids while in the (μ + 1), or SSG GA we only produce one offspring per iteration. Therefore, the running time for each function evaluation of (μ + 1) or SSG GA is smaller than that of the (μ + μ) or (μ, μ).

V. CONCLUSION

In this paper, four different GA strategy were compared namely Generational GA (μ, μ), Steady State GA or (μ + 1), Steady Generational GA, and finally (μ + μ). Also, each GA algorithm were implemented using three different crossover operators. The performance of the algorithms was evaluated over the Schaffer F6 problem, which is a challenging problem for EC to solve since it has many local optima around the global optima. (μ + μ) found to have better performance among the other algorithms, as it was expected, simply because it always takes the best μ individuals out of all the parents and offspring. However, one should keep in mind that this strategy produces μ offspring at each iteration, therefore the running time for each iteration is larger compared to the other strategies. Elitist Generational GA also works well, due to the fact that it exploits the best individual over and over again. Also, it is found that Mid-point crossover outperforms the remaining crossover operators in all the different strategies. And it might be due to the fact that in real coded representation it can take advantage of the parent's genotypes and interpolate a better solution.

VI. BREAKDOWN OF WORK

Each author coded one strategy. However, since we are in group of three, Armin Khayyer coded the different crossover operators as well as the last algorithm. Bhargav Joshi put all the different strategies together and created an interactive GUI that asks for the strategy and crossover operator as inputs and will output the number of function evaluations as well as the best individual. There is another capability in our code, if you make the GUI variable zero then it will run the code for the given strategy and for all different crossovers and outputs a csv file containing the best individual and number of function evaluations for 30 runs. For writing the paper and implementing the statistical analysis, all the group members contributed equally.

REFERENCES

[1] K. F. Man, K. S. Tang, and S. Kwong, "Genetic algorithms: concepts and applications [in engineering design]," *IEEE Trans. Ind. Electron.*, vol. 43, no. 5, pp. 519–534, 1996.

[2] G. Dozier, "'Advarsarial Machine Learning.'" 2019.

[3] J. Liu, P. Li, Xiaoning Ma, and Jiaxun Xie, "Modeling, analysis and simulation on searching for global optimum region of particle swarm optimization," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2017, pp. 813–821.

[4] G. Dozier, A. Homaifar, E. Tunstel, and D. Battle. "An Introduction to Evolutionary Computation" (Chapter 17), *Intelligent Control Systems Using Soft Computing Methodologies.* A. Zilouchian & M. Jamshidi (Eds.), pp. 365-380, CRC press.