I.J. Intelligent Systems and Applications, **2023, 1, 13-21**
Published Online on February 8, 2023 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2023.01.02

A Comparative Analysis of Bat and Genetic Algorithms for Test Case Prioritization in Regression Testing

Anthony Wambua Wambua*
Department of Computer Science, School of Computing & Information Technology, Murang'a University of Technology, Murang'a, Kenya
Department of Computer Science, School of Science & Engineering, Daystar University, Nairobi, Kenya
E-mail: awambua@daystar.ac.ke
*Corresponding author
ORCID iD: https://orcid.org/0000-0001-5110-9071

Geoffrey Mariga Wambugu
Department of Information Technology, School of Computing & Information Technology, Murang'a University of Technology, Murang'a, Kenya
E-mail: gmariga@mut.ac.ke
ORCID iD https://orcid.org/0000-0002-0250-0135

Received: 01 August 2022; Revised: 25 October 2022; Accepted: 11 November 2022; Published: 08 February 2023

Abstract: Regression testing is carried out to ensure that software modifications do not introduce new potential bugs to the existing software. Existing test cases are applied in the testing, such test cases can run into thousands, and there is not much time to execute all of them. Test Case Prioritization (TCP) is a technique to order test cases so that the test cases potentially revealing more faults are performed first. With TCP being deemed an optimization problem, several metaheuristic nature-inspired algorithms such as Bat, Genetic, Ant colony, and Firefly algorithms have been proposed for TCP. These algorithms have been compared theoretically or based on a single metric. This study employed an experimental design to offer an in-depth comparison of bat and genetic algorithms for TCP. Unprioritized test cases and a brute-force approach were used for comparison. Average Percentage Fault Detection (APFD)- a popular metric, execution time and memory usage were used to evaluate the algorithms' performance. The study underscored the importance of test case prioritization and established the superiority of the Genetic algorithm over the bat algorithm for TCP in APFD. No stark differences were recorded regarding memory usage and execution time for the two algorithms. Both algorithms seemed to scale well with the growth of test cases.

Indexing Terms: Test Case Prioritization, Bat Algorithm, Genetic Algorithm, Regression Testing, Nature-inspired Optimization Algorithms.

1. Introduction

Software testing is a critical undertaking in software development. As such, software engineers test software at different stages and levels. Regression testing is done before updates or modifications to existing software are submitted. Such updates could arise from additional software functionality or fixing existing software bugs. The validation is critical in ensuring changes do not introduce new defects to the deployed software. Regression testing can be restated as testing software S on all the test cases, TCs in each test suite TS associated with S. If S does not pass on TS, it should reveal the test cases that fail. Regression testing requires the execution of existing test cases in TS. As software updates become frequent and the software's size and complexity grow, test cases run into thousands, with their execution time running into hours [1]. Executing many test cases is both expensive and time taking.

To ensure efficiency in regression testing, three categories of techniques, namely test case reduction, selection and prioritization, are implemented [1]. Test case reduction and selection techniques focus on selecting a subset of the test cases with the most failing test cases [2]. A significant drawback of these techniques is that they discard some test cases, potentially leaving some software defects unearthed [3].

On the other hand, test case prioritization (TCP) does not discard any test cases [3]. The focus is on ordering test cases to maximize the fault detection rate. As such, test cases capable of revealing more faults at an earlier stage are executed first –are prioritized [4]. This paper will focus on test case prioritization to the periphery of the other techniques.

Several strategies for TCP, ranging from metaheuristic optimization algorithms to machine learning algorithms, have been proposed. Since prioritization is a ranking task, learning and, thus, machine learning algorithms have been applied [5]. Prioritization is also seen as an optimization problem, aiming to identify the most efficient ordering of test cases [6]. To this end, numerous nature-inspired optimization algorithms, including Bat and Genetic algorithms, have been proposed. Studies such as [6, 7] have found the Bat algorithm, modified or adapted as is, to offer great potential in TCP.

On the other hand, the Genetic algorithm is deemed mature Gupta [8], and Bajaj and Sangwan [9] found it to offer even more promising results. Both algorithms are bio-inspired and metaheuristic. When the need to apply either Genetic or Bat algorithms for test case prioritization arises, practitioners and researchers have to decide on the efficiency of the two algorithms. Such a decision might not be based on a single metric, such as execution time but several metrics, such as APFD, execution time, memory utilization and the growth of such metrics as the number of test cases in a test suite increases. Most researchers utilize these algorithms without in-depth and experimental comparison. In some instances, the comparison is either done based on a single metric such as APFD or the algorithms are simply used for baseline comparison with new techniques that the researchers could propose. This paper seeks to fill this void by providing an in-depth and experimental comparison of these two algorithms. The findings can help researchers in the future to make more informed decisions.

This study aims to compare Bat and Genetic metaheuristic algorithms for TCP. APFD, memory usage and execution time are metrics used in the comparison. Further, unprioritized test cases and brute force approaches are used for baseline comparison. The findings will be helpful to researchers wishing to apply or modify either of the algorithms for TCP. The findings will also contribute to research on using nature-inspired metaheuristic algorithms for TCP. The specific research questions in this comparative study are as follows:

- R1: Which between Bat and Genetic algorithms yields the best APFD?
- R2: How do Bat and Genetic algorithms compare in terms of memory and time usage in prioritizing a given set of test cases
- R3: How do the growths of APFD for Bat and Genetic algorithms compare with the growth of test cases?

The rest of this paper is organized as follows. Section 2 gives an overview of related work. Section 3 outlines the research methodology applied in this study, while Section 4 presents the experimental results. Section 5 discusses the results and implications, while Section 6 offers the study's conclusion.

2. Related Works

Test case prioritization has gained traction from researchers owing to the vital role of regression testing in software development. Authors have proposed different approaches that ensure efficiency, realizing that it might not always be possible to execute all test cases given the limited time and resources.

Researchers have explored algorithms primarily applied in machine learning. For example, Gokilavani and Bharathi [10] proposed a technique that used Principal Component Analysis (PCA) and K-means clustering to prioritize test cases. The study used PCA to select only the desired features from a Firefox bugs report dataset. Then, attributes were clustered using the K-means algorithm. A ranking algorithm was then used to rank the clusters and test cases in them. The technique was also evaluated using the know GZIP dataset, a real-world program with actual faults. The APFD metrics recorded promising performance for this technique.

Further, Bagherzadeh et al. [11] applied reinforcement learning to TCP. The study established the potential of machine learning to perform better in TCP than existing methods.

Other studies have focused on the application of metaheuristics nature-inspired algorithms. Vescan et al. [12] applied the Ant Colony algorithm to TCP. The algorithm’s properties allowed TCP’s features, such as cost and faults, to be encoded with the algorithm implemented in C/C++ language. The authors termed the results encouraging. Khatibsyarbini et al. [13] applied the firefly algorithm, a metaheuristic nature-inspired algorithm, to TCP. The study used benchmark programs to evaluate the performance of the algorithm. The metric used was APFD. The firefly algorithm performed slightly better than the known metaheuristic algorithms, such as the genetic algorithm.

The genetic algorithm (GA) is regarded as one of the mature and well-known [14] bio-inspired algorithms applied in optimization. Darwin’s theory of evolution inspires this algorithm. A fitness function is designed based on the goal to be achieved since the algorithm is based on the concept of evolution. Test cases undergo the operations selection, crossover and mutation and thus generate new solutions. At each iteration, the fitness function is evaluated to see if the permutation of test cases produces a better fitness – higher APFD. The study applied GA to test case prioritization. The study improved the original GA by including block coverage information. The study recorded better performance of GA with coverage information than GA without coverage information.

Mukherjee and Patnaik [3] surveyed 90 primary studies to understand the different approaches applied to TCP for studies conducted between 2001 and 2018. The study established three key findings. Firstly, APFD was the most used metric of all the studies carried out in the period under study. Secondly, coverage-aware prioritization methods were more prevalent. Thirdly, most research used C or Java programs for TCP studies.

Data mining concepts have also been explored for TCP. Azizi [15] proposed a tag-based recommender system TCP. The system used information retrieval (IR) to select test cases that were textually similar to the part of the code that had been altered. The study found the techniques effective for TCP. Similarly, Peng et al. [16] applied an information retrieval concept to TCP. The study used large, real-world software data sets with actual bugs for evaluation. It underscored the potential for information retrieval techniques in TCP. Change-aware IR techniques performed better than established coverage-based techniques for TCP.

Banias [17] proposed a memoization dynamic programming approach to TCP. One thousand test cases were used to evaluate the technique. This technique realized great memory utilization – the techniques used between 40 to 400 times less than other techniques.

2.1. Overview of BAT Algorithm

The bat algorithm is a metaheuristic algorithm conceptualized by Yang [18] and modeled around the collective behaviour of bats. The algorithm is based on the echolocation behaviour of Microbats – a species of bats comprising small-sized insectivorous bats. These creatures use echolocation to locate their prey and avoid obstacles as they fly around. These bats prefer optimal locations while looking for prey. Using echoes, bats can determine the distance, orientation of the prey and the speed at which the prey moves. A bat sends an echo and listens back while hunting for its prey. As it nears the prey, it increases the pulse rate and decreases its volume.

2.2. Adapting BAT Algorithm to TCP

Using the prior described bat behaviour, the following parameters have been modeled to help with optimization problems such as TCP. These variables are x_i the location, v_i the velocity, A_i the loudness, r_i the pulse emission rate, and f_i the frequency of the bat. The *i* represents the iteration; since bats keep moving, the algorithm is executed for several defined iterations. The steps in this algorithm are summarized as follows

- Initialize the population of bats with each bat assigned a random location x_i with velocity v_i,
- Compute the frequency f_i
- Determine the pulse rate r_i. and loudness A_i
- Perform *N* iterations by moving the bats. At each iteration, determine the local solution $F(x_i)$by checking if r_i>rand.
- While performing the iterations, determine the global best solution using $F(x_{best}) < F(xi)$ and A_i>rand
- Terminate the algorithm after all iteration
- Order the bats.

In our case, each bat represents a possible ordering of test cases. At each iteration, the APFD will be computed and stored as the local solution if r_i> rand. The local solution will then be compared with the global best $F(x_{best})$ and assigned to be the global best if $F(x_{best}) < F(xi)$ and Ai>rand

2.3. Overview of Genetic Algorithm

Proposed by J. H. Holland in 1992, the Genetic algorithm is a population-based metaheuristic algorithm based on Darwin’s theory of survival for the fittest [14]. Chromosomes are vital elements in the solution space. They iteratively undergo operations such as selection, crossover and mutation based on their fitness to produce the most optimal solution after successive generations.

2.4. Adapting Genetic Algorithm to TCP

The above-described evolution behaviour is modeled to adapt the Genetic algorithm to test case prioritization.
The following is a summary of the steps to be followed

- Initialize the population with N number of chromosomes. Each chromosome represents a possible combination of test cases.
- Define the fitness criteria. In this case, it is the APFD.
- Pick the chromosome (test case combination) with the best APFD
- For all the chromosomes, while all the faults have been covered, perform the crossover and mutation. A single crossover is applied.
- Terminate when the specified number of iterations is reached.

2.5. Evaluation Metrics

To compare the performance of the Bat and Genetic algorithms, this study will use known metrics such as the APFD, memory usage in MB and execution time in milliseconds. APFD is calculated as shown in (1).

$$APFD = 1 - \frac{TC + TC2 \ldots TCN}{N * M} + \frac{1}{2 * N} \tag{1}$$

Where N is the number of test cases, M is the number of faults and TC1+TC2…TCN refers to the sum of the position of the test cases that identify a particular fault. A higher APFD indicates that most faults are identified by test cases that are higher in the prioritization and, thus, excellent performance. A lower APFD value indicates that the position of the test case that identifies the faults is lower and, therefore, not good prioritization. In a systematic literature review involving 69 primary studies by Khatibsyarbini et al. [4], it was established that the most prevalent metric used for TCP studies was APFD, at 51%, followed by Coverage Effectiveness at 10%. The same study identified execution time at 7% as another metric commonly used to measure TCP performance. A similar survey by Lima and Vergilio [1] found that time and APFD were common metrics among the 35 primary studies used.

3. Methodology

The study employed an experimental research design. The dependent variables were the AFPD, execution and memory usage.

3.1. Experiment Context

Experiments for the Bat and Genetic Algorithm were carried out in a controlled environment. Each algorithm program was developed using Java and python programming languages in the IntelliJ IDEA integrated development environment (IDE). *Matplotlib* - a popular library used in python, was integrated into IntelliJ IDEA to plot the APFD graphs. The programs were executed on a computer with an Intel Core i7 processor, 8GB RAM. and 512GB SSD.

3.2. Experiment Variables

The experiment consisted of controlled variables and dependent variables. Controlled variables are the ones the authors can tweak and modify to observe the effect on the dependent variables. In this study, the controlled variables were the size of the test cases and the number of iterations the programs were to be executed. The dependent variables were APFD, execution time and memory usage.

3.3. Experiment Data

In this experiment, two test suits, A and B, were used. The two test suites were randomly generated. Random generation of test cases for experiments is not a new phenomenon amongst TCP researchers. Studies such as [7, 13] applied a similar approach. Test suite A had 10 test cases (T1-T10) and 13 faults (F1-F13), as shown in Table 1. Test suite B, which was double the size of test suite A was used to test the effect of the growth of test cases on the algorithms' APFD, memory utilization and execution time. Test suite B had 20 test cases (T1-T20) and 26 faults (F1-F26), as shown in Table 2. For both Table 1 and Table 2, the presence of ✓ at the intersection of the fault's column and test case's row implied that the specific test case unearthed the specific fault. The presence of × implied that the specific test case could not reveal the specific error.

For program execution during the experiment, Table 1 and Table 2 were converted into a matrix-fault matrix where each ✓ was replaced with a one and each × was replaced with a zero. The fault matrix was then fed into the Java program for processing.

Table 1. Fault Matrix - test Suite A.

T/F	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11**	**F12**	**F13**
T1	√	√	√	√	x	x	x	x	x	x	x	x	x
T2	√	x	√	√	x	x	√	x	√	x	x	x	x
T3	x	√	x	x	x	x	√	√	x	x	x	x	x
T4	√	x	√	x	x	x	√	x	x	√	x	√	√
T5	x	x	√	√	x	x	√	x	√	x	x	x	x
T6	x	x	x	x	x	√	√	√	√	x	x	x	x
T7	√	x	x	x	x	x	√	√	x	x	x	√	√
T8	x	√	x	√	x	√	x	x	x	x	√	x	x
T9	x	x	x	x	√	√	√	√	√	x	x	x	x
T10	x	x	x	√	√	x	√	√	x	x	x	x	√

3.4. Experiment Flow

Each program was executed with test suite A, shown in Table 1, and for 50 iterations. This test suite had 10 test cases and 13 faults. The stochastic nature of metaheuristic algorithms necessitates the programs to be executed in many iterations. In addition to the Bat and Genetic algorithm, a brute force approach was used for baseline comparison. The brute force program applied no optimization strategy but generated all possible combinations of the test cases and evaluated their APFDs. Such permutations would be critical for assessing the performance of the Genetic and Bat algorithms. Lastly, a program that executed all the test cases in unprioritized order was executed. The unprioritized approach would offer a better baseline in comparing Bat, Genetic and the brute force approach. The point is to compare the metric values of the unprioritized cases with those of the test cases as ordered by the Genetic and the Bat algorithms. This would underscore the importance of test case ordering. For each program, the APFD, the execution time and memory usage were recorded for analysis. Further, the resulting test case ordering was noted and are shown in the results section.

To understand the impact of doubling the test cases on the algorithms' APFD, test suite B, shown in Table 2, was used. The test suite had double the number of test cases in Table 1. Test suite B had 20 test cases and 26 faults. The respective fault matrix was coded into the program for each test suite to automate the APFD computation process.

Table 2. Fault Matrix - test Suite B.

T/F	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26
T1	√	√	√	√	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	√	√	√	x	x	√	√
T2	√	x	√	√	x	x	√	x	√	x	x	x	x	√	x	√	√	x	x	√	√	x	x	x	x	√
T3	x	√	x	x	√	x	√	√	x	x	x	x	x	√	√	x	x	x	x	x	√	x	x	x	√	√
T4	√	x	√	x	x	x	√	x	x	√	x	√	√	√	x	x	x	x	√	√	√	√	√	x	√	√
T5	x	x	√	√	x	x	√	x	√	x	x	x	x	√	√	x	√	x	x	√	√	x	x	√	√	√
T6	x	x	x	x	x	√	√	√	√	x	x	x	x	√	√	√	√	x	x	x	x	x	x	x	x	x
T7	√	x	x	x	x	x	√	√	x	x	x	√	√	x	x	x	x	√	√	√	√	√	x	x	x	x
T8	x	√	x	√	x	√	x	x	x	x	√	x	x	√	x	x	x	x	x	√	√	x	x	x	√	√
T9	x	x	x	x	√	√	√	√	√	x	x	x	x	√	√	√	√	x	x	x	x	x	√	x	x	x
T10	x	x	x	√	√	x	√	√	x	√	x	x	√	x	√	x	√	x	√	x	x	x	x	√	x	x
T11	√	x	x	x	x	x	√	√	x	x	x	√	√	√	√	√	√	x	x	x	x	x	x	x	x	x
T12	x	√	x	√	x	√	x	x	x	x	√	x	x	x	√	x	x	x	x	√	√	x	x	x	x	x
T13	√	√	√	√	x	x	x	x	x	x	x	x	x	x	x	x	x	x	√	√	√	√	x	x	x	x
T14	x	√	x	√	x	√	x	x	x	x	√	x	x	x	x	x	x	√	√	√	√	√	x	x	x	x
T15	√	x	x	x	x	x	√	√	x	√	x	√	√	√	x	x	x	x	x	√	√	x	√	x	√	√
T16	√	√	x	√	√	x	√	√	x	x	√	√	√	√	x	x	x	x	x	√	√	x	x	x	√	√
T17	√	x	x	x	x	√	√	√	√	x	x	√	√	√	x	√	x	x	x	√	x	x	√	x	√	√
T18	√	√	x	x	x	x	x	√	x	x	x	√	√	x	√	x	x	x	x	√	√	x	x	x	x	x
T19	√	x	√	√	x	x	√	√	x	√	x	x	√	x	√	x	√	x	√	x	x	x	x	√	x	x
T20	x	x	x	x	x	x	√	√	√	x	x	√	√	√	√	√	√	x	x	x	x	x	x	x	x	x

4. Results

4.1. APFD for BAT and Genetic Algorithms

After 50 iterations, Bat Algorithm returned an APFD of 72.69%. This value was arrived at in its 14th iteration. The algorithm prioritized the test cases in this order: T9, T1, T4, T2, T3, T5, T8, T7, T10, and T6. Fig.1 shows the performance of the algorithm over all 50 iterations.

As for the Genetic algorithm, after the 50 iterations, it returned an APFD rate of 75.76%. This was arrived at in its 20th iteration. The algorithm prioritized the cases in the order T8, T4, T9, T2, T6, T5, T1, T3, T7 and T10.

A brute force algorithm was applied to compare the performance of the two algorithms. A comparison between the three algorithms' APFD values is shown in Fig. 3. Table 3 shows each algorithm's final test case prioritization order.

Table 3. Test Case Ordering.

Algorithm	**Test Case Ordering**
Bat Algorithm	T9 T1 T4 T2 T3 T5 T8 T7 T10 T6
Genetic Algorithm	T8 T4 T9 T2 T6 T5 T1 T3 T7 T10
Brute force Algorithm	T4 T8 T9 T1 T5 T6 T7 T2 T3 T10
Unprioritized	T1 T2 T3 T4 T5 T6 T7 T8 T9 T10

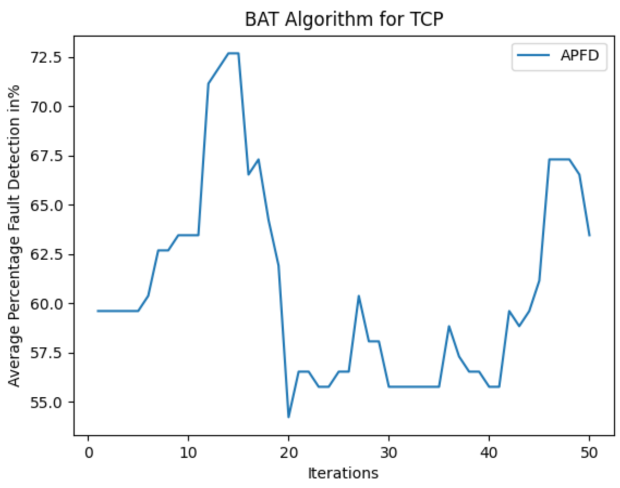

Fig.1. Bat Algorithm' APFD in TCP for 50 Iterations.

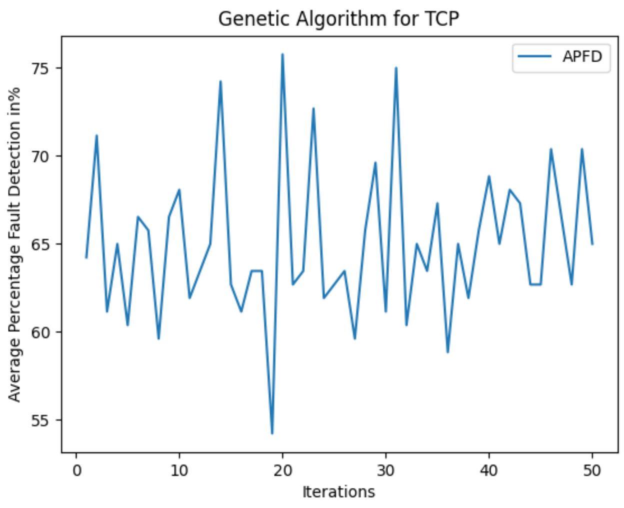

Fig.2. Genetic Algorithm' APFD in TCP for 50 Iterations.

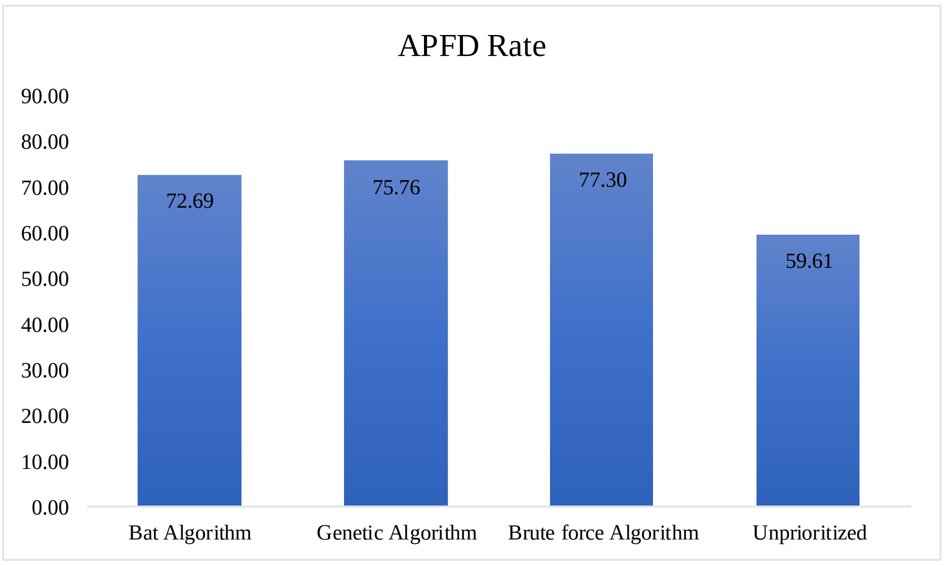

Fig.3. A Comparison of Genetic and Bat Algorithms' APFDs.

4.2. Memory Usage and Execution Time of Bat and Genetic Algorithms

The study sought to establish the memory and time utilization of the two algorithms. The findings are as shown in Table 4.

Table 4. Algorithm Resource Utilization.

Algorithm	**Time in Milli seconds**	**Memory in MB**
Bat Algorithm	187	1.19
Brute force algorithm	94329	177.32
Unprioritized	718	1.99

4.3. APFD, Time and Memory Utilization Growth

We sought to understand the impact of doubling the number of test cases and the faults on the growth of the APFD, execution time and memory utilization for both Genetic and Bat Algorithm. The brute force algorithm and the unprioritized case approach are also presented to provide a better analysis. Table 5 presents our findings in this aspect.

Running the brute force algorithm to explore all possible permutations of the test cases and compute their APFD proved challenging as the program kept running into Java heap space errors.

Table 5. Algorithm Performance with Test Case Growth.

	Test suite A			**Test suite B**		
Algorithm	APFD	Time in Milli seconds	Memory in MB	APFD	Time in Milli seconds	Memory in bytes
Genetic Algorithm	75.76%	109	1.18	87.69%	187	1.20
Bat Algorithm	72.69%	187	1.19	87.5%	171	1.19
Brute force algorithm	77.30%	94329	177.32	Java heap space error	Java heap space error	Java heap space error
Unprioritized	59.61%	718	1.99	83.26%	187	1.18

5. Discussion

A comparison of the Genetic and Bat algorithms shows that the Genetic algorithm reports a higher APFD rate of 75.76%, while the Bat algorithm achieved 72.69%. The difference is not stark for a small data set. However, the difference can be vital when the test cases run into thousands. The brute force approach achieves a stellar performance of an APFD of 77.30%. Comparing the APFD rates of the two algorithms to the APFD of the unprioritized approach that achieved a meager APFD rate of 59.61% underscores the importance of prioritizing test cases. More errors can be unearthed earlier with prioritized test cases as opposed to having the test cases unprioritized.

For the execution time, the Genetic algorithm performs better by using 109 milliseconds than the Bat algorithm, which uses 187 milliseconds. The unprioritized approach uses 718 milliseconds, while the brute force approach uses 94329 milliseconds. Given that our test suite A had a meager number of only 10 test cases and 13 faults, the brute force performance is dismal and unacceptable. This approach would fail for huge test cases. Even though the brute force approach had recorded the highest APFD, it was at the expense of time. It would be practical to forgo an accuracy of 77.30% for either 75.76% or 72.69% and save more time.

As for memory utilization, even though there is no stark difference between the Genetic algorithm and the Bat algorithm, the Genetic algorithm still performed better than the Bat Algorithm by using 1.18MB, while Bat Algorithm used 1.19MB. The baseline approaches reported higher memory usage, with the unprioritized approach using 1.99MB and the brute force approach using 177.32MB. Test case prioritization thus saves time and memory. It improves the APFD rate, allowing the discovery of software faults earlier in the regression testing.

On doubling the test cases and faults, the Genetic algorithm still shows superiority over the Bat algorithm regarding the APFD rate. The Genetic algorithm reported 87.69%, 0.19% higher than the Bat algorithm, which reported 87.5%. Both APFD rates improved since this depends more on the nature fault matrix and not entirely on the number of faults or test cases. Even though the difference is slight, the indication is that with a small or big test suite, the Genetic algorithm could perform better than the Bat algorithm. Interestingly, the Genetic algorithm used more memory and time with the doubling of test cases and faults than the Bat algorithm. While the differences are trivial, it would be interesting for researchers to see how they play out with large and real data sets. This forms part of this study's future undertakings. The brute force approach could not run to the end with the doubled test cases and faults as the program repeatedly reported Java heap space errors and indications that more memory than available was needed. The approach is unscalable as it computes all possible permutations while computing the APFD rate. It is bound to take a long time and use tremendous memory.

The limitation of the study is two-fold. Firstly, the use of randomly generated test cases and fault matrix. Using real-world program test cases and faults would have been ideal. To mitigate this, two sets of test cases were used. The second test case is two times bigger than the first one regarding the number of test cases and faults. Secondly, the failure of the brute force approach to execute to the end using the second test case, which had 20 test cases and 26 faults, denies us the opportunity to compare Bat and Genetic algorithms against the brute force algorithm in terms of the APFD, time utilization and memory usage. The fact that the experiment shows the difference between Bat and Genetic algorithms regarding APFD, memory and time usage mitigates the limitation of the brute force approach failing.

6. Conclusion and Future Work

This study's goal was to compare Bat and Genetic algorithms to establish which of the two is more efficient. Such information would be helpful to test case prioritization researchers and practitioners in deciding which of the two algorithms to apply in TCP or similar problem domains. To meet this goal, the study sought to determine the APFD, memory usage and execution time of both Bat and Genetic algorithms subjected to the same test cases. Two test case sets were used where test case set two had double the number of test cases and faults as those of test case set one. This would help us understand how APFD, memory usage and execution time would vary with growth in test cases. Additionally, two baseline approaches were used for comparison with Bat and Genetic algorithms, namely the brute force and unprioritized test cases. This study established the superiority of the Genetic algorithm over the Bat algorithm for test case prioritization in terms of the average percentage of faults detection. Even though the brute-force approach recorded a stellar performance intern of APFD on smaller test cases, the time utilization and memory utilization are unacceptable. On the other hand, the unprioritized test case approach offers a poor performance in terms of its APFD, memory and time utilization compared to Genetic and Bat algorithms, underscoring the need for test cases to be prioritized during regression testing. Further, the study concludes that there is no considerable difference in memory usage and execution time between the Genetic and Bat algorithms, even though the Genetic algorithm performs better in the two aspects.

Future work will focus on using real-world project test cases and faults. Such as the study will compare the two algorithms with a huge data set and the complexities of real test cases and faults as opposed to randomly generated test cases.

References

[1] J. A. P. Lima and S. R. Vergilio, "Test Case Prioritization in Continuous Integration environments: A systematic mapping study," *Information and Software Technology,* vol. 121, p. 106268, 2020.

[2] E. Cruciani, B. Miranda, R. Verdecchia, and A. Bertolino, "Scalable approaches for test suite reduction," presented at the Proceedings of the 41st International Conference on Software Engineering, Montreal, Quebec, Canada, 2019. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00055.

[3] R. Mukherjee and K. S. Patnaik, "A survey on different approaches for software test case prioritization," *Journal of King Saud University - Computer and Information Sciences,* vol. 33, no. 9, pp. 1041-1054, 2021/11/01/ 2021, doi: https://doi.org/10.1016/j.jksuci.2018.09.005.

[4] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology,* vol. 93, pp. 74-93, 2018/01/01/ 2018, doi: https://doi.org/10.1016/j.infsof.2017.08.014.

[5] A. Bertolino, A. Guerriero, B. Miranda, R. Pietrantuono, and S. Russo, "Learning-to-rank vs ranking-to-learn: strategies for regression testing in continuous integration," presented at the Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, South Korea, 2020. [Online]. Available: https://doi.org/10.1145/3377811.3380369.

[6] S. Chaudhary and R. Singh, "Test case prioritization using modified bat algorithm," *IJCSE,* vol. 6, no. 7, pp. 145-149, 2018.

[7] A. Bajaj and O. P. Sangwan, "Test case prioritization using bat algorithm," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science),* vol. 14, no. 2, pp. 593-598, 2021.

[8] P. K. Gupta, "K-Step Crossover Method based on Genetic Algorithm for Test Suite Prioritization in Regression Testing," *J. Univers. Comput. Sci.,* vol. 27, no. 2, pp. 170-189, 2021.

[9] A. Bajaj and O. P. Sangwan, "A systematic literature review of test case prioritization using genetic algorithms," *IEEE Access,* vol. 7, pp. 126355-126375, 2019.

[10] N. Gokilavani and B. Bharathi, "Test case prioritization to examine software for fault detection using PCA extraction and K-means clustering with ranking," *Soft Computing,* vol. 25, no. 7, pp. 5163-5172, 2021.

[11] M. Bagherzadeh, N. Kahani, and L. Briand, "Reinforcement learning for test case prioritization," *IEEE Transactions on Software Engineering,* 2021.

[12] A. Vescan, C.-M. Pintea, and P. C. Pop, "Test case prioritization—ANT algorithm with faults severity," *Logic Journal of the IGPL,* vol. 30, no. 2, pp. 277-288, 2022.

[13] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, H. N. A. Hamed, and M. D. M. Suffian, "Test case prioritization using firefly algorithm for software testing," *IEEE Access,* vol. 7, pp. 132360-132373, 2019.

[14] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications,* vol. 80, no. 5, pp. 8091-8126, 2021.

[15] M. Azizi, "A tag-based recommender system for regression test case prioritization," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW),* 2021: IEEE, pp. 146-157.

[16] Q. Peng, A. Shi, and L. Zhang, “Empirically revisiting and enhancing IR-based test-case prioritization,” presented at the Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, 2020. [Online]. Available: https://doi.org/10.1145/3395363.3397383.

[17] O. Banias, “Test case selection-prioritization approach based on memoization dynamic programming algorithm,” *Information and Software Technology*, vol. 115, pp. 119-130, 2019.

[18] X.-S. Yang, “A new metaheuristic bat-inspired algorithm,” in *nature inspired cooperative strategies for optimization (NICSO 2010)*: Springer, 2010, pp. 65-74.

Authors’ Profiles

Mr. Anthony Wambua Wambua received his Bachelor of Applied Computer Science from Periyar University, India, in 2009 and his Master of Computer Science from Bharathiar University, India, in 2011. He is pursuing a Ph.D. in Computer Science and has been a lecturer in the Department of Computer Science at Daystar University, Kenya, since 2014. Mr Wambua is a member of IEEE & ACM. His main research focuses on Software Engineering, Metaheuristic Algorithms, and eLearning. He has eight years of teaching experience.

Dr. Geoffrey Mariga Wambugu received his B.Sc. degree in Mathematics and Computer Science from Jomo Kenyatta University of Agriculture and Technology, Kenya, in 2000, the M.Sc. degree in Information Systems from The University of Nairobi, Nairobi, Kenya, in 2012, and the Ph.D. in Information Technology JKUAT, in 2019. He has served for over ten years as head of the department in higher education institutions in Kenya and has been involved in the design, development, review and implementation of Computing Curricula in different universities in Kenya. Currently, he is a Senior Lecturer and Dean of the School of Computing and Information Technology at Murang’a University of Technology. His research interests include Probabilistic Machine Learning, Text Analytics, Natural Language Processing, Data mining, and Big Data Analytics.

How to cite this paper: Anthony Wambua Wambua, Geoffrey Mariga Wambugu, "A Comparative Analysis of Bat and Genetic Algorithms for Test Case Prioritization in Regression Testing", International Journal of Intelligent Systems and Applications(IJISA), Vol.15, No.1, pp.13-21, 2023. DOI:10.5815/ijisa.2023.01.02