Soft Computing
https://doi.org/10.1007/s00500-022-07121-9

APPLICATION OF SOFT COMPUTING

Improved novel bat algorithm for test case prioritization and minimization

Anu Bajaj[1,2] · Om Prakash Sangwan[2] · Ajith Abraham[1]

Accepted: 31 January 2022

Abstract
Regression testing is essential for continuous integration and continuous development. It is needed to ensure that the modifications have not produced any errors or faults, thereby maintaining the quality and reliability of the software. The testers usually avoid exhaustive retesting because it requires lots of effort and time. The test case prioritization and minimization solve the issue by scheduling the critical test cases and removing redundant ones. Optimization techniques help by improving the efficiency of these techniques while utilizing limited resources. This paper proposed an enhanced discrete novel bat algorithm for the test case prioritization. The algorithm is modified in two ways. First, we have proposed a fix-up mechanism for the discrete combinatorial problem, which conducts the perturbation in the population using the asexual reproduction algorithm. Second, the novel bat algorithm is improved, where the bats hunt in different habitats with quantum behavior using Gaussian distribution and search in the limited habitat with Doppler effect. In addition, we have embedded the test case minimization procedure in the algorithm for redundancy reduction. The experimental results are empirically analyzed using different testing criteria, i.e., fault and statement coverage on three subject programs from the software infrastructure repository. Consequently, test selection percentage, coverage loss, fault detection loss, and cost reduction percentages are deduced for the test case minimization at program and version levels. Empirical results and statistical comparisons with the random search, bat algorithm, novel bat algorithm, birds swarm algorithm, whale optimization algorithm, and genetic algorithm show the outperformance of the proposed algorithm.

Keywords Bat algorithm · Nature-inspired algorithms · Regression testing · Search-based software testing · Test case prioritization · Test case minimization

1 Introduction

To survive in the competitive era is the most challenging task of software companies as it is the need of the hour to update the software according to the customers' requirements. The main target is to update the software without compromising the software quality. It requires retesting to check that the modifications have not induced any side effects. This retesting is known as regression testing (Yoo and Harman 2012). As the software evolves rapidly, it becomes a tedious job to perform testing of each modified version.

On the other hand, the complexity of the software increases with frequent updates and leads to an increase in time, effort, and budget required for regression testing (Öztürk 2018). Therefore, software testing at the evolution and maintenance stage is more important to assure the quality of the software. According to Rothermel et al. (1999) exhaustive retesting took around seven days to retest 20 K lines of code software completely. Moreover, complete retesting accounts for approximately 80% of maintenance costs (Bajaj and Sangwan 2021b). Regression testing is divided into three techniques, i.e., test case prioritization (TCP), test case selection (TCS), and test case minimization (TCM), to overcome these challenging issues.

✉ Anu Bajaj
anu.bajaj@mirlabs.org

[1] Machine Intelligence Research Labs (MIR Labs), Auburn, Washington, USA

[2] Department of Computer Science and Engineering, Guru Jambheshwar University of Science and Technology, Hisar, Haryana, India

Published online: 06 May 2022

TCP ranks the test cases using pre-defined goals, and TCS targets the critical test cases affected by the modified area. In comparison, TCM reduces the test suite by removing redundant information (Yoo and Harman 2012). TCP gained more attention as it does not eliminate test cases from the test suite (Bajaj and Sangwan 2018). Moreover, it helps to minimize the time needed for retesting if the process is interrupted (Bajaj and Sangwan 2021a). It motivates us to include the TCM procedure at the end of the TCP algorithm, which removes the redundant test cases from the prioritized test suite in case of limited time and budget constraints.

Furthermore, TCP uses different test adequacy criteria to schedule the test cases, e.g., maximizing requirements coverage (Bajaj and Sangwan 2019a). The testing criteria play an important role in determining the effectiveness of the technique. These metrics can be used in two ways 1) to prioritize the test cases based on these criteria, e.g., code coverage-based TCP, fault coverage-based TCP, and cost-cognizant TCP, and 2) to evaluate the TCP efficacy through cost and coverage optimization. These factors bring several challenges to assess their relevance, interdependence, and priority concerning one another. Another problem is deciding which sort of these metrics to use, such as code coverage, cost-aware, and fault detection based. In this paper, we identified the effect of these metrics on performance improvement in the TCP and their implications on TCM followed by TCP.

On the other hand, processing a vast test suite makes the regression testing an NP-Hard problem (Yoo and Harman 2012), which can be efficiently solved with optimization algorithms. In other words, the cost-effectiveness of regression testing can be further improved with optimization approaches (Bajaj and Sangwan 2018). Nature-inspired algorithms attracted researchers due to their simple structure and ease of solving the problem (Bajaj and Sangwan 2021d). The algorithms are mathematically formulated by imitating natural phenomena. Various nature-inspired algorithms are proposed in the literature. These are broadly classified into three categories, i.e., biology-inspired, Physics/chemistry, and social phenomena-inspired algorithms (Fister et al. 2013). Among these algorithms, most commonly used are evolutionary algorithms and swarm intelligence-based algorithms from the biology-inspired class. They mimic the living being's evolution mechanism and the flocking behavior of birds or insects.

The algorithms are becoming popular as they are problem independent and follow a derivative-free mechanism. It is evident from the broad application domain like vehicle routing problem (Zhou et al. 2016), economic dispatch problem (Mahdi et al. 2019), supply chain management (Chouhan et al. 2020), and other applications (Osaba et al. 2019). These algorithms have also proved their effectiveness for regression testing (Bajaj and Sangwan 2018). The genetic algorithms (Bajaj and Sangwan 2019b a), particle swarm optimization (Mann et al. 2018), cuckoo search (Kaur and Agrawal 2017) are some of the examples of these algorithms used in regression testing. Few researchers have used relatively new algorithms like the gravitational search algorithm (Bajaj and Sangwan 2021a), firefly algorithm (Khatibsyarbini et al. 2019), and bat algorithm (Öztürk 2018). For example, bat algorithms use the concept of frequency tuning, loudness, and echolocation properties for searching the prey. It was used to solve the TCP problem by mapping the test adequacy criteria (fault coverage) with the loudness properties of the bats (Bajaj and Sangwan 2021c).

On the other hand, the original bat algorithm suffered from several problems like premature convergence and stuck into local optima (Mahdi et al. 2019). These problems can be solved by improvement and hybridization with other algorithms. One such improvement is the novel bat algorithm, which uses the advanced properties of bats. i.e., adaptation in different habitat conditions (Meng et al. 2015). The algorithm shows promising results in various applications like the economic dispatch problem (Gautham and Rajamohan 2016). However, it has not been explored in the TCP domain, and it may be because it was initially designed for continuous problems. So, in this paper, we have proposed an improved discrete novel bat algorithm by using a fix-up mechanism based on an asexual genetic algorithm to convert the infeasible solutions into feasible ones (Bajaj and Sangwan 2021a). Moreover, to accelerate the performance in the last iterations, the uniform distribution of quantum-behaved bats is replaced with the Gaussian distribution. Therefore, the main contributions of this paper are:

- To propose a discrete novel bat algorithm (iBAT) for solving the combinatorial TCP problem using an asexual genetic reproduction algorithm. Furthermore, to improve the algorithm by using random numbers with Gaussian distribution for the quantum-behaved bats.
- To embed the process of the TCM in the TCP for redundancy reduction.
- To analyze the results on different testing criteria, i.e., fault coverage and code coverage, and evaluate the impact of TCM on the cost and coverage reduction.
- To statistically compare the proposed algorithm with the random search (RS), genetic algorithm (GA), bat algorithm (BAT), novel bat algorithm (nBAT), and relatively new algorithms: bird swarm algorithm (BSA) and whale optimization algorithm (WA).

The subsequent section briefs the existing literature on the bat algorithm and its applications in regression testing.

Section 3 describes the working mechanism of the basic bat and novel bat algorithms. The proposed work is presented in Sect. 4. Section 5 explains the experimental settings followed by results and discussions in Sect. 6. The conclusion and future work are presented in Sect. 7.

2 Related work

This Section summarizes the nature-inspired algorithms used in regression testing with a focus on the bat algorithms. It is followed by the ongoing research in the bat algorithms for solving various real-world issues. This retrieved literature information helped us in the proposal of the improved discrete novel bat algorithm for solving the TCP problem as follows:

2.1 Test case prioritization and minimization using nature-inspired algorithms

Nature-inspired algorithms have been used widely by the researchers to improve the cost-effectiveness of TCP and TCM due to these algorithms' effectiveness and efficiency. In the literature, several nature-inspired algorithm-based TCP and TCM strategies have been suggested (Bajaj and Sangwan 2018, 2020). For example, Li et al. (2007) compared the search-based algorithms' performance with the traditional algorithms for TCP. It was found that conventional methods performed better than the search-based algorithms. The other observation stated that the search space could be better explored with the help of GA. It led to further research to exploit the potential of nature-inspired algorithms. Furthermore, Bajaj and Sangwan (2019a) discovered that parameter choices and operators have a significant impact on the efficiency of algorithms. Mohapatra and Prasad (2015) implemented ant colony optimization (ACO) for reducing the test suite size and complexity. The comparative results showed the outperformance of the proposed algorithms over the traditional methods. Cuckoo search algorithm was proposed for configuration-aware software testing to minimize the test suite size (Ahmed 2016).

Table 1 briefs different type of nature-inspired algorithms used in the TCP and TCM. It is observed that biology-inspired algorithms are the most-targeted ones among the nature-inspired algorithms. However, relatively new algorithms are being used more often these days. Sugave et al. (2018) used the bat algorithm for TCM by proposing a new fitness function to improve the diversity in the search process. They posed constraints on cost minimization and complete coverage of the requirements. The comparison with other existing algorithms on different software infrastructure repository programs proved the outperformance of their proposed algorithm. Öztürk (2018), proposed the bat algorithm for TCP and used the execution time and number of faults as the distance and loudness of the bats. The proposed algorithm worked better than the existing algorithms for APFD and resistance against the code complexity. Another preliminary study used the loudness of the bats for the fault coverage to prioritize the test cases (Bajaj and Sangwan 2020).

Hashim and Dawood (2018) proposed firefly algorithm for TCM using UML state charts to maximize the coverage. Furthermore, a hybrid PSO with firefly algorithm was proposed to reduce the test suite size (Bharathi 2022). The hybrid algorithm performed better than CSA and GA for fault detection rate, test reduction rate and computational costs. On the other hand, Hybrid of Dragonfly with PSO produced promising results as compared to GA, DA, BA, and PSO (Bajaj and Abraham 2021). Still, the area is open for research like novel bat algorithm (Meng et al. 2015), whale optimization algorithm (Mirjalili and Lewis 2016), birds swarm optimization (Meng et al. 2016), emperor penguin optimization (Dhiman and Kumar 2018), and chicken swarm optimization (Meng et al. 2014) which are some of the examples which are yet to be exploited for TCP and TCM. A handful of researchers have used these relatively new algorithms for TCP. It may be because of the mapping of the problem because TCP is a combinatorial problem and needs permutation encoding. The above algorithms mapped the TCP problem with the properties of the bats. However, it may lead to erroneous results due to improper mapping of the techniques.

2.2 Bat algorithm and its modifications

Bat algorithms belong to the swarm intelligence-based category of the biology-inspired algorithms that mimic the echolocation behavior of the bats for searching the food (solution). It was developed by Xin-She Yang in (2010) and proved its effectiveness in various engineering optimization problems since its inception (Yang and He 2013). Several researchers have improved and hybridized the bat algorithm with other optimization algorithms to enhance efficiency. For example, Gandomi and Yang (2014) empirically evaluated the effect of different chaotic maps at different stages of the bat algorithm. They noticed that the sinusoidal map used for pulse emission worked better than other variants. Nawi et al. (2016) also used the chaotic map instead of the random number β for frequency update. Besides, they updated the solution with Gaussian distribution random walk to avoid the longer jumps and suboptimal solutions. The results proved that the proposed algorithm worked better than the existing algorithms for big dimensional problems.

Table 1 Summary of nature-inspired algorithms used in TCP and TCM

Author (Year)	Regression Testing Technique	Nature-Inspired Algorithms	Objective Functions
Li et al. (2007)	TCP	GA	Maximize fault coverage
Bharathi (2022)	TCM	PSO-FFA	Maximize fault detection and minimize test suite
Bajaj and Sangwan (2019b)	TCP	GA	Maximize fault coverage
Mohapatra and Prasad (2015)	TCM	ACO	Minimize test suite size
Ahmed (2016)	TCM	CSA	Minimize test suite size
Bajaj and Sangwan (2021a)	TCP and TCM	GSA	Maximize fault coverage
Hashim and Dawood (2018)	TCM	FFA	Maximize code coverage
Bajaj and Abraham (2021)	TCP and TCM	DA	Maximize fault coverage
Sugave et al. (2018)	TCM	BAT	Maximize requirement coverage and minimize cost
Öztürk (2018)	TCP	BAT	Maximize fault coverage
Bajaj and Sangwan (2020)	TCP	BAT	Maximize fault coverage

Meng et al. (2015) proposed the novel bat algorithm and updated the position using both quantum and mechanical behavior of the bats. In this, the bat can move anywhere for a defined probability, and the frequency was updated with the self-adaptive compensative Doppler effect. Gautham and Rajamohan (2016) used this novel bat algorithm for economic dispatch problem. Zhao and He (2016) also considered the Doppler effect and applied the chaotic map to the velocity update. They kept the elite solutions in the elitist set and solved the analog circuit tolerance problem. Mahdi et al. (2019) proposed the bat algorithm with quantum behavior to solve the many-objective problem. To avoid premature convergence, Huang et al. (2019) replaced the random number of quantum behaved bat position update equation with the Gaussian distributions. Though their algorithm outperformed most numerical optimization problems, it still has the disadvantage of requiring plenty of parameters to be tuned.

The bat algorithm was proposed for solving the continuous optimization problem. Several researchers have developed the discrete version of the algorithm by altering the solution representation schemes. For example, Zhou et al. (2016) hybridized the bat algorithm with the path relinking mechanism for the vehicle routing problem. They divided the routes into sub-routes and then combined them to form a valid route. Saji and Riffi (2016) randomly selected pair of solutions and followed a 2-exchange crossover heuristic that sets the velocities pair within which the crossover takes place to solve the travelling salesman problem. Riffi et al. (2017) solved the quadratic assignment problem using the modified uniform crossover operator and updated the solution using the multiple pairwise swapping of the numbers. Osaba et al. (2016) used the 2-opt and 3-opt operators to exploit and explore the travelling salesman problem. Osaba et al. (2019) also used the hamming distance between the solutions to update the velocity and correspondingly update the positions using the insertion and exchange function. They successfully tested the approach on the medical goods distribution problem. Tang et al. (2018) used the self- and collective learning method to update the solutions to identify the k influential nodes. Table 2 summarizes the modifications done in the bat algorithms over time and the used application area to solve a continuous or discrete problem.

The promising results of bat algorithms in various applications as shown in Table 2 motivated us to propose the discrete version of the novel bat algorithm for solving the combinatorial TCP problem. Our work is different from the above works in the following ways. The novel bat algorithm has an advantage over other bat algorithms because the bats hunt in different habitats with quantum behavior and search in the limited habitat with Doppler effect. However, it also has some disadvantages like premature convergence. So, we have improved the algorithm by replacing the position update equation with the Gaussian distribution function to avoid premature convergence, influenced by Huang et al. (2019). Additionally, the novel bat algorithm is initially intended for a continuous problem. TCP is a combinatorial problem, so we modified it using the permutation encoding scheme and performed the perturbation in the population through a new fix-up mechanism.

We have also verified the algorithm's robustness for different testing criteria, fault coverage, code coverage, and

Table 2 Summary of Bat algorithm and its modifications

Author (Year)	Modifications in Bat Algorithm	Application Type	Application area
Gandomi and Yang (2014)	Different chaotic maps	Continuous	Global optimization
Nawi et al. (2016)	Chaotic map for frequency update	Continuous	Big dimensional problem
Meng et al. (2015)	Quantum behaved bats with Doppler effect	Continuous	Engineering designs
Zhao and He (2016)	Doppler effect and chaotic map for velocity update	Continuous	Analog fault diagnosis
Gautham and Rajamohan (2016)	Quantum behaved bats with Doppler effect	Continuous	Economic dispatch problem
Mahdi et al. (2019)	Quantum behaved bats	Continuous	Many-objective problem
Huang et al. (2019)	Quantum behaved bats position update with Gaussian distribution	Continuous	Numerical optimization problems
Zhou et al. (2016)	Hybridized with path relinking mechanism	Discrete	Vehicle routing problem
Saji and Riffi (2016)	2-exchange crossover heuristic	Discrete	Travelling salesman problem
Riffi et al. (2017)	Modified uniform crossover operator	Discrete	Quadratic assignment problem
Osaba et al. (2016)	2-opt and 3-opt operators	Discrete	Travelling salesman problem
Osaba et al. (2019)	Position update using the insertion and exchange function	Discrete	Medical goods distribution problem
Tang et al. (2018)	Self and collective learning method	Discrete	Influence maximization

their combination with execution cost. Besides we have extended the algorithm for minimizing the redundancy as well. In other words, we have investigated the impact of test case prioritization on reducing redundancy, i.e., whether the algorithms prioritize the test cases based on defined testing criteria. The proposed improved novel bat algorithm results have been compared with the baseline approach: random search (RS), state-of-the-art algorithms: GA, BAT, original novel bat algorithm (nBAT), and relatively new algorithms: BSA and WA. The findings revealed that the proposed algorithm performed better than the other nature-inspired algorithms and RS for TCP with different testing criteria. On the other side, in some cases, the difference between the performance of the proposed iBAT algorithm and WA was not statistically significant for TCM. However, the mean values of the proposed iBAT algorithm are better comparatively.

3 Preliminaries

In this section, we summarized the working of the bat and novel bat algorithms.

3.1 Bat Algorithm

The bat algorithm mimics the echolocation behavior, i.e., vary the pulse emission rate, frequency, and loudness for searching the prey, thereby controlling the exploration and exploitation. It sends a very high-frequency ultrasound wave, and with the echoes, it can find:

(1) How distant is the prey from it
(2) Type and orientation of the prey with the loudness of sound
(3) The moving speed of the prey

The bat algorithm is designed using three characteristics of bats, i.e., echolocation, frequency tuning, loudness. In other words, the bats move randomly in the search space at position x_i with velocity v_i and varying frequency in the range of minimum and maximum frequency '$f_{\min}$' and '$f_{\max.}$' Also, they increase their pulse emission rate (r_i) and decrease the loudness (A_i) when they are in the proximity of food. At each generation, the position and velocities are updated with different frequencies and formulated as

$$f_i = f_{\min} + (f_{\max} - f_{\min}) * rand\,(0,1) \quad (1)$$

$$v_i^{t+1} = v_i + \left(x_i^t - x_{\text{best}}\right) * f_i \quad (2)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (3)$$

where t = 1,2,3,…m (number of generations), i = 1,2,3,…..n (number of bats), x_{best} = global best location, rand(0,1) = generates random number between 0 & 1. The local search is performed by improving the current best position using random walk:

$$\text{if } r_i^t > \text{rand } x_i^{t+1} = x_{\text{best}} + \varepsilon * \text{mean}(A) \quad (4)$$

where ε = [− 1, 1]. Further, exploration and exploitation

are controlled by decreasing the loudness and increasing the emission rate using the following equations:

$$A_i^{t+1} = \alpha * A_i^t \tag{5}$$

$$r_i^{t+1} = r0 * \left[1 - e^{(-\gamma t)}\right] \tag{6}$$

where α [0, 1] and $\gamma > 0$ are constants. The pseudocode of bat algorithm is shown in Algorithm 1.

region. Hence, the bats can explore an extensive range of habitats. Two rules are included in the original Bat:

(1) The foraging of the bats in different habitats is decided with the help of stochastic selection.
(2) Bats can adaptively regulate their compensation rate for Doppler effect in echoes following the proximity of their targets.

Algorithm 1. Bat Algorithm

1. **Begin**
2. Generate initial population x_i randomly
3. Define the pulse emission rates r_i and loudness A_i
4. Define frequencies f_i between f_{min} and f_{max}
5. For t=1:Max_generations
6. Generate new solutions by adjusting frequencies using (1)
7. Update locations and velocities using (2) and (3)
8. If (rand>r_i)
   Select a solution from the best ones
   Generate local best solutions from the selected best solution x_{best} using (4)
9. End
10. Generate new solutions by flying randomly
11. If rand<A_i and fit(x_i)>fit(x_{best})
    Accept the new solutions
    decrease A_i and increase r_i according to (5) and (6)
12. End if
13. Sort the bats and find the current best x_{best}
14. End for
15. Store the final results
16. **End**

3.2 Novel bat algorithm

Meng et al. (2015) developed the novel bat algorithm. The algorithm adopted the advanced echolocation capability of the bats because some bats use the constant-frequency sound pulse or some use frequency-modulated signals for echolocation. It depends on their hunting strategies like finding food, avoiding hurdles, and tracing the cracks in trees to settle down in the dark. This capability can also be correlated with their self-adaptive compensation for Doppler effect in echoes. Different species behave differently, i.e., they may exhibit partial or full compensation. Also, some species hunt in different habitats while others search in a single habitat like water or forest habitats. They can regulate their echolocation behavior if they forage under different habitats. These characteristics of the bats can be adapted using quantum mechanics in which a particle with defined probability can move at any location of the search

The quantum-behaved particle's path can be obtained by assuming that M particles having δ potential and specified energy are well-centered in each dimension of N-dimensional Hilbert search space. Therefore, the jth component of the particle's location at tth iteration is obtained through the Monte Carlo method as follows:

$$x_{ij}^{t+1} = a_{ij}^t \pm \frac{L_{ij}^t}{2} \ln\left(\frac{1}{u_{ij}^t}\right) \tag{7}$$

$$L_{ij}^t = 2\Theta \left| \text{mean}_j^t - x_{ij}^t \right| \text{ and } \text{mean}_j^t = \frac{1}{N} * \sum_i^N x_{ij}^t \tag{8}$$

where "u_{ij}" is a uniformly distributed number in the interval (0, 1), "a" is a local attractor/prey for each individual, "*Theta*" is contraction–expansion coefficient. The global best solution can act as an attractor, i.e., if a bat finds the food location, others will also forage there as soon as they reach the food site. So the above equations can be replaced with the below equations

$$x_{ij}^{t+1} = \begin{cases} x_{\text{best}} + \Theta * |\text{mean}_j^t - x_{ij}^t| * \ln\left(\frac{1}{u_{ij}^t}\right), & rand_j(0,1) < 0.5, \\ x_{\text{best}} - \Theta * |\text{mean}_j^t - x_{ij}^t| * \ln\left(\frac{1}{u_{ij}^t}\right), & \text{otherwise} \end{cases} \tag{9}$$

Some random phenomena govern the selection of the bats' habitat. Mathematically it can use the probabilistic decision p, i.e., if $p \in [0, 1] > rand\ [0, 1]$ as the threshold of selection for quantum behavior of bats to hunt in an extensive choice of habitats.

Otherwise, they search in defined habitats with mechanical behavior. The frequencies of the bats are not only updated with the random assignment between f_{min} and f_{max} but also with the help of the Doppler effect and their compensation rates for the Doppler effect. In other words, the bats may distinguish targets by the preys' interference, e.g., preys' wing flutter rates change the Doppler effect. The Doppler effect finds the variation in the frequency of the periodic event by the relative motion of the source and receiver. It is higher when they are approaching each other, identical when passing by, and lower if they are farther away from each other. This phenomenon can be mathematically formulated as:

$$f_r = \frac{v \pm vr}{v \mp vs} f_s \tag{10}$$

where source and receiver are moving in the medium (speed v), having the speeds of v_r, and v_s with frequencies f_r and f_s, respectively. The – and + signs denote source and receiver are moving away or toward each other, i.e., it depends on the direction of the velocity. Similarly, the bats try to approach the target, while the targets attempt their best to move farther from the bats. So, in the algorithm, the bats try to approach the prey, i.e., global best solution, so, '' + '' sign is used, and the frequency is updated as:

$$f_i^{t+1} = \frac{c - v_i^t}{c + v_{\text{best}}} f_{ij}^t \tag{11}$$

Furthermore, the frequencies considerably affect the bats catching up with the prey because it depends on the velocity increment, which is the product of $\lambda_i f_i$. So, the bats themselves regulate the compensation rates for the Doppler effect in echoes. The bats try to fly forwards to catch up with the prey if they lag behind the prey. In other words, the bats compensate positively for the Doppler effect in echoes, i.e., ${x_{ij}}^t$ approaches x_{best}, if it is smaller than x_{best}. On the other side, they slow down their search and negatively compensate for the Doppler effect in echoes if the value of ${x_{ij}}^t$ is higher than $x_{best.}$ Each bat has its own compensation rate C_i, which lies between *0* and *1*, i.e., no and full compensation, therefore, the frequency equation is updated as:

$$f_{ij}^{t+1} = \frac{c - v_{ij}^t}{c + v_{\text{best}}} f_{ij}^t \left(1 + Ci \frac{x_{\text{best}} - x_{ij}^t}{x_{\text{best}} - x_{ij}^t + \epsilon}\right) \tag{12}$$

where c is the speed of sound velocity in the air (340 m/s), ε is a small random number to avoid dividing by zero error, and v_{best} is the speed of the global best position x_{best}. The velocities of bats are updated with the inertia weight w that regulates the rate of inheritance from the previous velocity of the bat.

$$v_{ij}^{t+1} = w * v_{ij}^t + \left(x_{\text{best}} - x_{ij}^t\right) * f_{ij}^t \tag{13}$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1} \tag{14}$$

The local search is performed using the following equations

$$x_{ij}^{t+1} = x_{\text{best}} * \left[1 + \{randn(0, \sigma^2)\}\right] \tag{15}$$

$$\sigma^2 = \left|A_i^t - \text{mean}\ (A^t)\right| + \epsilon \tag{16}$$

where *randn (0, σ^2)* is the Gaussian distribution having a mean and standard deviation of *0* and σ^2, ε is added for positive standard deviation. It updates the loudness and pulse rate similar to the original bat algorithm. However, to escape the local optima, i.e., if the bats are unable to find a better solution for the previous δ attempts then it reinitializes the loudness and sets a high pulse emission rate so that bats can explore in another region.

4 Proposed Work

This section deals with the detailed description of the proposed improved novel bat algorithm (iBAT) for TCP and TCM. The improvements done in the novel bat algorithm:

1. The test case prioritization problem is mapped to the bat algorithm using asexual reproduction fix up mechanism.
2. Gaussian probability is used to avoid premature convergence.
3. The test suite minimization procedure is embedded into the prioritization algorithm to reduce the test suite size as described below:

4.1 Population Update

The bat and novel bat algorithms were initially projected for continuous problems that update the real-valued solutions. We cannot use the original algorithm directly for a

discrete combinatorial problem, and it needs to be modified. In the bat algorithm, we used the loudness, i.e., the target's proximity, as the amount of fault covered by test cases, and the pulse emission rates are updated through the frequency tuning. The position of the bats is then sorted according to the loudness and the pulse emission rate. Though the results are promising on the small experimental study, more case studies still need to be validated. The algorithm's simulation results on the considered real subject programs contrasted with the preliminary study (see Sect. 5). In other words, the algorithm performs worse than the genetic algorithm and is even close to the random

Algorithm 2: Test Case Minimization Procedure

```
1.  Begin
2.    Define test fault matrix TF, Prioritized Suite T and Faults position array F
3.    Initialize minimized suite array of test cases indices MS indices [F]=0
4.    % Find the faults positions which are covered by first test case in T
5.    Index=find(TF(1,F)=1)
6.    % Fill the MS indices with the faults positions covered by the first test case in T
7.    MS indices (Index)=1
8.    Find the other test cases required for full coverage
9.    For i=1: size(T)
10.       For j=1: size(F)
11.          If (TF(T(i),F(j))=1 and MS indices(j)=0)
12.                MS indices (j)=i;
13.          End If
14.       End For
15.   End For
16.   % Provides the list of test cases from their indices
17.   Minimized Suite=T(MS indices)
18. End
```

search, so the area is open to discover other options. Therefore, here we have used the permutation encoding for solution representation because the proper mapping of the problem increases the algorithm's efficiency and effectiveness (Bajaj and Sangwan 2021a). As the original algorithm works on continuous values, we have updated the continuous values to permutation sequences. So, we built a new adaptation strategy for converting the infeasible solutions to the feasible ones with the asexual reproduction algorithm (Farasat et al. 2010 and Mansouri et al. 2011). It builds a relationship between real numbers and test case series by inheriting the parent solution's characteristics shown in Fig. 1 (Mansouri et al. 2011). While generating the offspring (*bud*) from the parent, it keeps the offspring's feasible values (*larva*). In other words, the algorithm updates the solutions and rounds them off to natural numbers. The out of bound and the duplicate values are replaced with don't care conditions (*). These infeasible values are replaced by the values from the previous solution to obtain a correct solution. For example, x = [*4, 5, 1, 6, 2, 3*] is updated to y = [*5.1, 6.7, 7.2, 5.5, 3.2, 1*], y is corrected to [*5, 6, 1, 5, 3, 1*] resulting into [*5, 6, 1, *, 3, **]. So, with the help of the previous solution, the remaining values are acquired to generate a correct solution as [*5, 6, 1, 4, 3, 2*].

4.2 Gaussian Probability

To avoid premature convergence, the uniform random number u of the Eq. (9) is replaced by G, the absolute value of Gaussian distribution abs (*randn*) or *N (0, 1)* with mean zero and standard deviation one. Its one-dimensional probability density function is given by Huang et al. (2019):

$$G(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}},\ x \geq 0 \tag{17}$$

So the equation is updated as

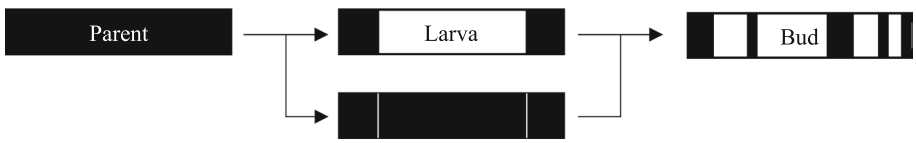

Fig. 1 Asexual Reproduction Mechanism

$$x_{ij}^{t+1} = \begin{cases} x_{\text{best}} + \Theta * |\text{mean}_j^t - x_{ij}^t| * \ln(\frac{1}{G_{ij}^t}), & rand_j(0,1) < 0.5, \\ x_{\text{best}} - \Theta * |\text{mean}_j^t - x_{ij}^t| * \ln(\frac{1}{G_{ij}^t}), & \text{otherwise} \end{cases} \quad (18)$$

4.3 Test case minimization

The current best solution of each iteration is passed through the redundancy check, and the first *k* test cases that covered the faults/statements early are selected, thereby minimizing suite size and cost. In other words, the algorithm prioritized the test cases followed by the redundancy reduction at every iteration. The advantage of this process is that it tells us how precisely the test cases are prioritized. The better the prioritization, the lesser number of test cases that fulfil the criteria of full coverage. The algorithm for fault coverage is presented in Algorithm 2, and the pseudo-code of the improved discrete novel bat algorithm is described in Algorithm 3.

5 Experimental settings

This section explains an empirical study, including research questions, subject programs, and performance metrics. Table 3 presents the research questions and their motives. The proposed algorithm is compared with the baseline approach: random search (RS), state-of-the-art algorithm: GA (Bajaj and Sangwan 2018), bat algorithms: BAT (Bajaj and Sangwan 2020), and novel bat algorithm (nBAT). Apart from these algorithms, the results are also compared with the relatively new nature-inspired algorithms, i.e., birds swarm optimization (BSA) and whale optimization algorithm (WA). These algorithms are implemented in MATLAB R2017 installed on an HP laptop with Windows 10, Intel i5 processor, and 4 GB RAM. Each algorithm is executed 30 times due to the stochastic behavior of the nature-inspired approaches.

Further, the algorithms are validated on the three different open-source Java programs (*jtopas, ant,* and *jmeter*) obtained from the popular software infrastructure repository (*SIR*) (Do et al. 2010). *Jtopas* is a Java library to parse

Algorithm 3. Improved Discrete Novel Bat Algorithm

Begin
Generate initial population x_i randomly
Define the pulse emission rates r_i and loudness A_i
Define frequencies f_i between f_{min} and f_{max}, probability of habitat selection (p), compensation rates (C), contraction/expansion coefficient (□), inertia weight (w) and threshold δ
For t=1:Max_generations
    If rand< p
        Generate new solutions using (18)
    Else
        Generate new solutions by adjusting frequencies using (1) and (12)
        Update solutions and velocities using (13) and (14)
    End if
    Repair the solutions using asexual reproduction operator
    If (rand>r_i)
        Select a solution from the best ones
        Generate local best solutions from the selected best solution x_{best} using (15) and (16)
    End if
    Repair the solutions using asexual reproduction operator
    Evaluate the fitness of the solutions
    If rand<A_i and f(x_i)>f(x_{best})
        Accept the new solutions
        decrease A_i and increase r_i according to (5) and (6)
    End if
    Test suite minimization procedure (Algorithm 3)
    If the x_{best} does not improve for δ attempts
    Reinitialize the loudness A_i and update the pulse emission rates r_i ϵ[0.85, 0.9]
End for
Store the final results
End

Table 3 Research questions and their motives

RQ	Research Questions	Motivation
RQ 1	What is the performance of the proposed algorithm for test case prioritization?	
RQ 1.1	Does the proposed algorithm robust against all the testing criteria compared to random search and existing algorithms?	The motive is to check whether the proposed algorithms performed better than random search (sanity check). Additionally, it finds out which algorithm achieves better performance (effectiveness check) and the influence of different testing criteria on the performance of the algorithms
RQ 1.2	What is the impact of different testing criteria on test case prioritization?	
RQ 2	What is the performance of the proposed algorithm for test case minimization?	
RQ 2.1	Does the proposed algorithm perform better than existing algorithms?	The goal is to investigate about the performance of the proposed algorithm with respect to the random search (sanity check) and also against BAT, BSA, GA, nBAT and WA algorithms. Furthermore, to identify which testing criteria maximize the test case minimization, i.e., achieved lower test selection percentage. Additionally, to see its impact on the coverage loss and fault detection capability loss of the reduced test suite. Furthermore, it also classifies the algorithm based on the reduction in testing cost percentage
RQ 2.2	How the algorithms react to different testing criteria for test case minimization?	
RQ 2.3	What is the effect of the coverage loss and fault detection capability loss of the selected test suite using different testing criteria?	
RQ 2.4	Which algorithm achieved better cost reduction percentage?	

Table 4 Subject Programs

Programs[a]	Versions	KLOC	#Classes	#Methods	#Test Cases	Type
Jtopas	4	5.4	50	748	209	JUnit
Ant	7	80.4	650	7524	878	JUnit
jmeter	5	43.4	389	3613	97	JUnit

[a]Source: http://www.sir.csc.ncsu.edu

the text data. *Ant* is a Java-based build tool while *jmeter* is a desktop java application for loading test functional behavior and performance measurement. We have extracted several versions of these objects. The details are described in Table 4.

Each program is hand seeded with a limited number of faults. However, to perform an adequate statistical comparison amongst the algorithms, the number of faults should be large enough (Eghbali and Tahvildari 2016). We increased the number of faults using mutation faults generated with *μJava* following the procedure described by (Mei et al. 2012) and Eghbali and Tahvildari (2016). The execution time and the coverage information of these programs are obtained from *JUnit* and *Emma*, an open-source code coverage tool. The experiments are conducted on every version of each program, and the average results of all the versions are used for comparing the overall algorithmic performance of each program.

5.1 Parameter settings

We selected the parameters with the extensive analysis from related literature and also through the trial and error method for relevant values. Furthermore, these values are analyzed systematically with the help of the Taguchi method (Chouhan et al. 2020), and the extracted values are presented in Table 5. Observations from the Taguchi Method also suggest that common parameters are the same for all the algorithms, which is also required for a fair comparison.

5.2 Performance measurements

The performance of the algorithms must be evaluated to confirm their effectiveness and efficiency. We have used several performance measures to evaluate the algorithms for test case prioritization and minimization as described below:

5.2.1 Test case prioritization

Mostly, the test cases are prioritized with two different testing criteria, i.e., fault and statement coverage. The code coverage information is generally available for all the software, so various researchers widely use it (Li et al. 2007). At the same time, others consider fault coverage as

Table 5 Parameter Settings

Algorithms	Parameters values
GA	Tournament selection
	Ordered crossover
	Swap mutation
	Crossover probability = 0.8
	Mutation probability = 0.1
BAT	$r_0 = 0.001$, $r0_{min} = 0$, $r0_{max} = 1$, $f_{min} = 0$, $f_{max} = 1.5$, $\alpha = 0.9$, $\gamma = 0.99$
BSA	$c_1 = 1.5$, $c_2 = 2$, $a1 = 1$, $a2 = 1.5$
nBAT	$r_0 = 0.002$, $r0_{min} = 0$, $r0_{max} = 1$, $A_{min} = 1$, $A_{max} = 2$, $f_{min} = 0$, $f_{max} = 1.5$, $\alpha = 0.8$, $\gamma = 0.9$, $Prob_{max} = 0.8$, $Prob_{min} = 0.5$, $w_{min} = 0.5$, $w_{max} = 0.9$, $\Theta_{max} = 1.0$, $\Theta_{min} = 0.5$, $C_{max} = 0.8$, $C_{min} = 0.2$, $\delta = 10$
WA	$a_{max} = 2$, $a_{min} = 0$, b = 1
iBAT	$r_0 = 0.001$, $r0_{min} = 0$, $r0_{max} = 1$, $A_{min} = 1$, $A_{max} = 2$, $f_{min} = 0$, $f_{max} = 1.5$, $\alpha = 0.9$, $\gamma = 0.99$, $Prob_{max} = 0.8$, $Prob_{min} = 0.6$, $w_{min} = 0.5$, $w_{max} = 0.9$, $\Theta_{max} = 1.0$, $\Theta_{min} = 0.5$, $C_{max} = 0.9$, $C_{min} = 0.2$, $\delta = 10$
Common Parameters	Population size = 100
	Generation Size = 1000
	Number of executions = 30

an essential criterion for ordering the test cases if the prior knowledge of the faults is present (Marchetto et al. 2015). Here, we have used both the testing criteria to check the robustness of the proposed algorithm. Therefore, the well-known metrics used as the fitness metrics and the effectiveness measures are defined below:

Average Percentage of Fault Detection (APFD): It measures the weighted average of the covered faults according to their position in the test suite (Elbaum et al. 2002). It is calculated as:

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{m} TF(i)}{n * m} + \frac{1}{2 * n} \tag{19}$$

TF(i) denotes the position of the test case, which first detects the *ith* fault; *m* denotes the number of faults covered by the test suite of size *n*. It lies in (0, 100), and higher is better.

Average Percentage of Fault Detection with Cost (APFDc): APFD assumes uniform test cases cost and fault severities, which are usually non-uniform. Therefore, a cost cognizant metric APFDc has been proposed (Malishevsky et al. 2006) which includes different costs and faults severities in APFD which is formulated as:

$$\text{APFDc} = 1 - \frac{\sum_{i=1}^{m} fs(i) * \left(\sum_{j=TF(i)}^{n} \cos t(j) - \frac{1}{2} \cos t(TF(i)) \right)}{\sum_{i=1}^{n} \cos t(i) * \sum_{i}^{m} fs(i)} \tag{20}$$

Here, *fs(i)* is the fault severity of the *ith* fault, *cost (i)* is the test execution cost of an *ith* test case, and *cost (TF(i))* is the execution cost of the test case which detects the *ith* fault first.

Formulation of the Average Percentage of Statement Coverage (APSC) and APSC with cost (APSCc) is similar to the APFD and APFDc. However, the only difference is that they calculate the statement coverage instead of fault coverage. These metrics are also used as the fitness function to guide the search-based algorithms in the search space.

5.2.2 Test case minimization

The commonly used effectiveness measures are test suite reduction percentage/test selection percentage and cost reduction percentage. Followed by the test case prioritization, the test case minimization reduces the size of the test suite using 100% fault coverage or 100% statement coverage. It is evident if we minimize the test suite on one coverage basis, then it affects the other coverage criteria. For example, fault coverage-based reduction leads to some statement coverage loss and vice versa. Therefore, we have used coverage loss percentage, fault detection capability loss percentage as the performance measure for 100% fault coverage and 100% statement coverage, respectively, as described below:

Test Selection Percentage (TSP): It is the ratio of the size of the reduced test suite to the size of the original test suite.

$$TSP = \frac{r}{n} * 100 \tag{21}$$

where r denotes the reduced test cases in the test suite of n test cases.

Coverage Loss Percentage (CLP): It is the ratio of the number of the statements left uncovered by the minimized test suite to the total number of statements covered by the original test suite.

$$CLP = \frac{nsl}{ts} * 100 \tag{22}$$

where nsl number of statements left uncovered ts is the total number of statements.

Fault Detection Capability Loss Percentage (FLP): It is the ratio of the number of the faults uncovered by the minimized test suite to the total number of faults covered by the original test suite.

$$FLP = \frac{nfl}{tf} * 100 \tag{23}$$

Here nfl is number of faults left uncovered tf is the total number of faults.

Cost Reduction Percentage (CRP): It is the reduction in the cost of the test suite compared to the cost of the original suite.

$$CRP = \frac{cr}{tr} * 100 \tag{24}$$

cr is the cost of reduced test suite and tr is the cost of original test suite.

6 Results and discussions

This section evaluates the algorithm for TCP and TCM considering different fitness functions. One-way ANOVA test with a p-value of 0.05 is used to compare the performance of the algorithms statistically. To further investigate the performance of the algorithms in comparison to each other, ANOVA is followed by the Tukey simultaneous test to rank them. Furthermore, the graphical statistics are validated with the boxplots and 95% confidence limit interval plots for fitness metrics and their corresponding test selection percentages.

On the other hand, it finds out the algorithm's strength for multiple testing criteria, viz., fault coverage, statement coverage, and combination of both coverage with the cost. Also, to check whether the algorithm is robust against different fitness functions or not. Furthermore, the effect of test case minimization on the fault coverage loss, statement coverage loss, and cost reduction is also analyzed. The experimental results of a program are calculated as the cumulative average on all of its versions. The values of performance measures of each version are the mean of 30 independent runs.

6.1 Performance analysis of test case prioritization (RQ 1)

The mean fitness values of different performance metrics and their Tukey group rankings are shown in Table 6 for all the subject programs.

6.1.1 RQ 1.1 Effectiveness check among Nature-Inspired algorithms

Table 6 shows that all the nature-inspired algorithms are better than RS. The mean fitness value of the bat algorithm is closer to a random search. Moreover, it suggests an insignificant difference between means of BAT and RS in APFDc and APSCc for the *jmeter* program. Overall, the algorithm works better than RS. On the other hand, all the nature-inspired algorithms are statistically better than the BAT algorithm, except that the pair (BSA, BAT) has equivalent performance for the fault coverage criteria. The reason may be the improper mapping of the problem with the algorithm. As in the BAT algorithm, we correlated the bat's properties with the TCP, i.e., loudness represents the test cases' fault status. It is the point that motivated us to find a variant of the bat algorithm.

Therefore, in the nBAT algorithm, we used the usual algorithm procedure without altering its working. Instead, we used the permutation encoding and fixed the illegal values with the legal ones, and the results are significantly different from BAT. The nBAT algorithm performed better than BSA and GA for APFD, but they all tend to converge alike as we consider the fault coverage with cost and statement coverage criteria. In other words, when we have more data variations, these algorithms have similar performance. Whereas nBAT and WA are statistically indifferent to fault coverage criteria, conversely, WA performed better in statement coverage criteria. Hence, the proposal of an improved novel bat algorithm to enhance the performance of nBAT. The proposed algorithm iBAT is statistically different from all the compared algorithms for all the testing criteria except the pair (WA, iBAT) while ranking the algorithms according to Tukey Tests. In other words, the iBAT performed statistically equivalent to WA for the statement coverage criteria and one out of three programs for APFDc. Still, the proposed algorithm has higher mean values in both cases, which makes the proposed algorithm superior to all algorithms, as shown in the bar charts of Fig. 2. It can be concluded that the iBAT

Table 6 Comparison of the algorithms for different fitness functions

Program Versions	Algorithms	Fitness functions wise Performance (%) and Tukey Group Ranking (TR)							
		APFD	TR	APFDc	TR	APSC	TR	APSCc	TR
Jtopas	iBAT	95.621	A	95.272	A	98.298	A	97.480	A
	WA	93.930	B	94.120	B	98.161	AB	97.292	A
	nBAT	93.118	B	93.503	B	97.706	BC	96.711	B
	GA	91.918	C	92.801	BC	97.463	C	96.179	C
	BSA	91.180	D	91.705	CD	97.820	BC	96.732	B
	BAT	89.951	E	90.695	D	96.863	D	95.600	D
	RS	88.062	F	87.467	E	95.553	E	94.928	E
Ant	iBAT	93.937	A	93.651	A	97.482	A	96.640	A
	WA	93.292	B	93.209	A	97.239	AB	96.366	A
	nBAT	92.149	BC	91.837	B	96.586	C	95.642	B
	GA	91.304	CD	91.603	B	96.657	BC	95.704	B
	BSA	90.577	DE	90.912	B	96.719	BC	95.746	B
	BAT	90.094	E	89.491	C	95.924	D	94.952	C
	RS	87.372	F	87.037	D	95.164	E	94.275	D
Jmeter	iBAT	93.696	A	95.051	A	98.987	A	98.728	A
	WA	92.137	B	93.626	B	98.916	AB	98.543	AB
	nBAT	91.943	B	93.242	B	98.625	BC	98.126	BC
	GA	90.672	C	91.059	C	98.331	C	97.865	CD
	BSA	89.396	D	91.834	C	98.655	BC	98.346	B
	BAT	89.145	D	89.217	D	97.748	D	97.518	DE
	RS	87.759	E	87.994	D	97.153	E	97.200	E

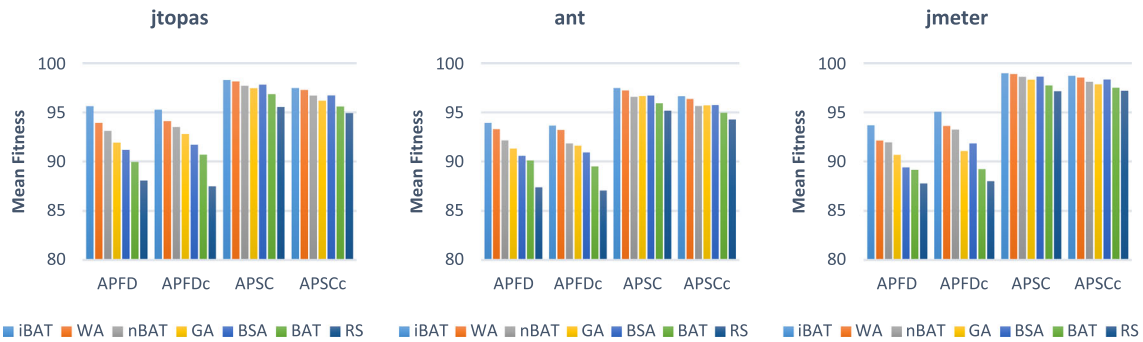

Fig. 2 Bar charts of the programs showing the performance of the algorithms for different testing criteria

algorithm is robust against all the testing criteria compared to the other algorithms.

6.1.2 RQ 1.3 Impact of different testing criteria

The distribution of fitness values is also pictorially represented in the boxplots, as shown in Fig. 3. The boxplots reveal that the variance in the algorithmic performance is higher for fault coverage than the statement coverage. It is because the faults are widely spread in the whole program. On the other side, most test cases cover almost the same statements, so due to redundant coverage, the boxplots of the statement coverage criteria are more squeezed than the fault coverage criteria.

Another point is that the variance in fault coverage with the cost is highest among all the testing criteria. Additionally, the widespread fault distribution and different execution costs of the algorithms lead to a separate ranking of the test cases and, hence, the variations. Therefore, the execution cost of the test cases also plays a vital role in TCP. If two test cases cover a similar number of faults and

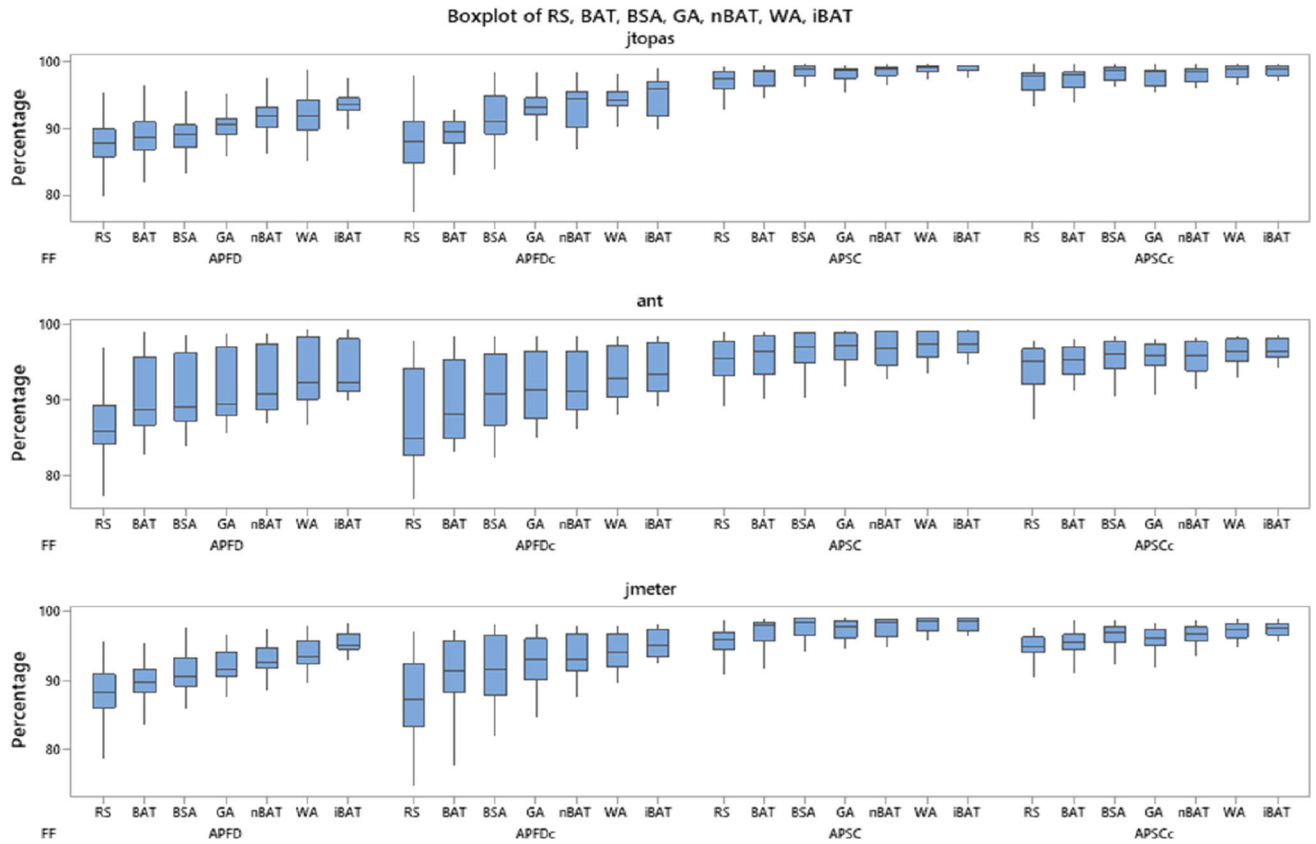

Fig. 3 Boxplots of RS, BAT, BSA, GA, nBAT, WA and iBAT for test case prioritization using APFD, APFDc, APSC and APSCc

their cost varies, then the algorithm will select the least-cost test case first compared to the other one.

6.2 Performance analysis of test case minimization (RQ 2)

The average values of a test selection percentages of all the algorithms and their Tukey group rankings are shown in Table 7 for all the subject programs.

6.2.1 RQ 2.1 Effectiveness check among Nature-Inspired algorithms

Observations from Table 7 suggest that there is not a massive difference in the algorithms' performance for redundancy reduction in the case of BAT and RS for APSCc. It is also verified from the ANOVA test that the pairs (nBAT, BSA) (nBAT, WA) are statistically equivalent in most of the cases. Overall, the numerical results of WA are better than the nBAT, followed by BSA. Also, all other nature-inspired algorithms and RS mean values are different in all the testing criteria. It also shows that the suite size obtained from the BAT and nBAT algorithms is less significant than the iBAT algorithm, showing the algorithms' inefficiency compared to the iBAT.

Furthermore, the Tukey group ranking suggests a statistically insignificant difference between the pair of GA and BSA. On the other side, there is an apparent significant difference between the performance of the pairs of WA and iBAT for the fault coverage criteria. Although they are close competitors, the proposed algorithm's numerical mean value is the lowest, as evident from the bar charts in Fig. 4.

6.2.2 RQ 2.2 Impact of different testing criteria

The pictorial representation of the distribution of test selection percentages is shown in Fig. 5 in the form of interval plots. The interval plots show the superiority of the proposed algorithm to all the other nature-inspired algorithms for all the testing criteria. The algorithms seem to be significantly different from each other. One interesting fact is that the test suite reduction is higher in the case of statement coverage criteria than the fault coverage, as shown in Table 7 and Fig. 4. It is because redundancy in statement coverage is high while the faults are distributed in the whole program. Another observation reveals that the variance in the algorithmic performance depends on the program's characteristics. For example, *jtopas* has a high variance in statement coverage while at the same time

Table 7 Comparisons of the algorithms for test case minimization over different fitness functions

Program Versions	Algorithms	Fitness functions wise TSP and Tukey Group Ranking (%)							
		TSP_{APFD}	TR	TSP_{APFDc}	TR	TSP_{APSC}	TR	TSP_{APSCc}	TR
jtopas	iBAT	12.467	A	12.208	A	11.000	A	10.950	A
	WA	12.833	B	12.900	B	11.042	B	12.142	B
	nBAT	13.533	C	13.692	B	13.108	C	13.775	C
	GA	14.367	D	14.150	C	14.200	D	15.175	D
	BSA	13.683	D	14.700	C	12.425	C	13.058	C
	BAT	15.992	E	15.567	D	15.117	D	16.258	E
	RS	17.083	F	17.133	E	17.850	E	16.983	E
ant	iBAT	18.300	A	18.676	A	17.133	A	17.552	A
	WA	18.795	AB	18.957	A	17.557	AB	17.733	A
	nBAT	19.981	B	20.871	B	18.905	B	19.981	B
	GA	20.933	C	21.143	BC	19.005	BC	20.057	B
	BSA	22.081	D	21.729	C	18.933	B	19.876	B
	BAT	22.295	D	23.148	D	20.957	C	22.000	C
	RS	24.800	E	25.210	E	23.295	D	22.910	C
jmeter	iBAT	16.507	A	16.167	A	7.047	A	7.457	A
	WA	18.083	B	17.270	B	7.420	AB	8.033	B
	nBAT	17.907	B	17.620	B	8.267	BC	8.940	C
	GA	19.180	C	18.797	C	9.053	C	9.400	C
	BSA	20.070	D	18.477	BC	7.917	B	8.260	BC
	BAT	20.033	D	19.733	D	10.077	D	10.293	D
	RS	21.117	D	20.900	D	10.750	D	11.067	D

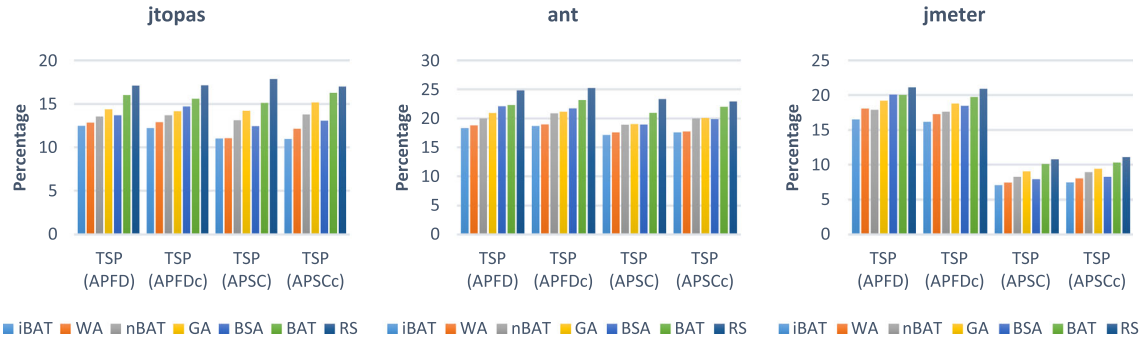

Fig. 4 Test selection percentage of the algorithms for different testing criteria

jmeter response opposite to that. The variation in *ant* is almost identical. The reason behind this is that we have merged the results of different versions of the program into one and found the cumulative test suite reduction. This test suite reduction also tells the underlying difference between the performances of the algorithms for test case prioritization, i.e., whether the proposed algorithm rightly prioritizes the test cases or not. If an algorithm is performing well, but it does not better reduce the test suite according to the 100% testing criteria, it may be by chance effect or due to program characteristics.

6.2.3 RQ 2.3 Impact on code and fault coverage percentage

Here, we have analyzed the losses incurred due to the reduction in test suite size by different testing criteria. As evident from Table 8 that there is statement coverage loss and fault detection capability loss when we perform minimization with fault coverage criteria and statement coverage criteria, respectively.

Observations from Table 8 suggest that the proposed algorithm has the least statement coverage loss for APFD and APFDc but high fault coverage loss compared to other algorithms for APSC and APSCc. In other words, the algorithms RS, BAT, and GA have less fault coverage loss than different algorithms for APSC and APSCc. Figure 6

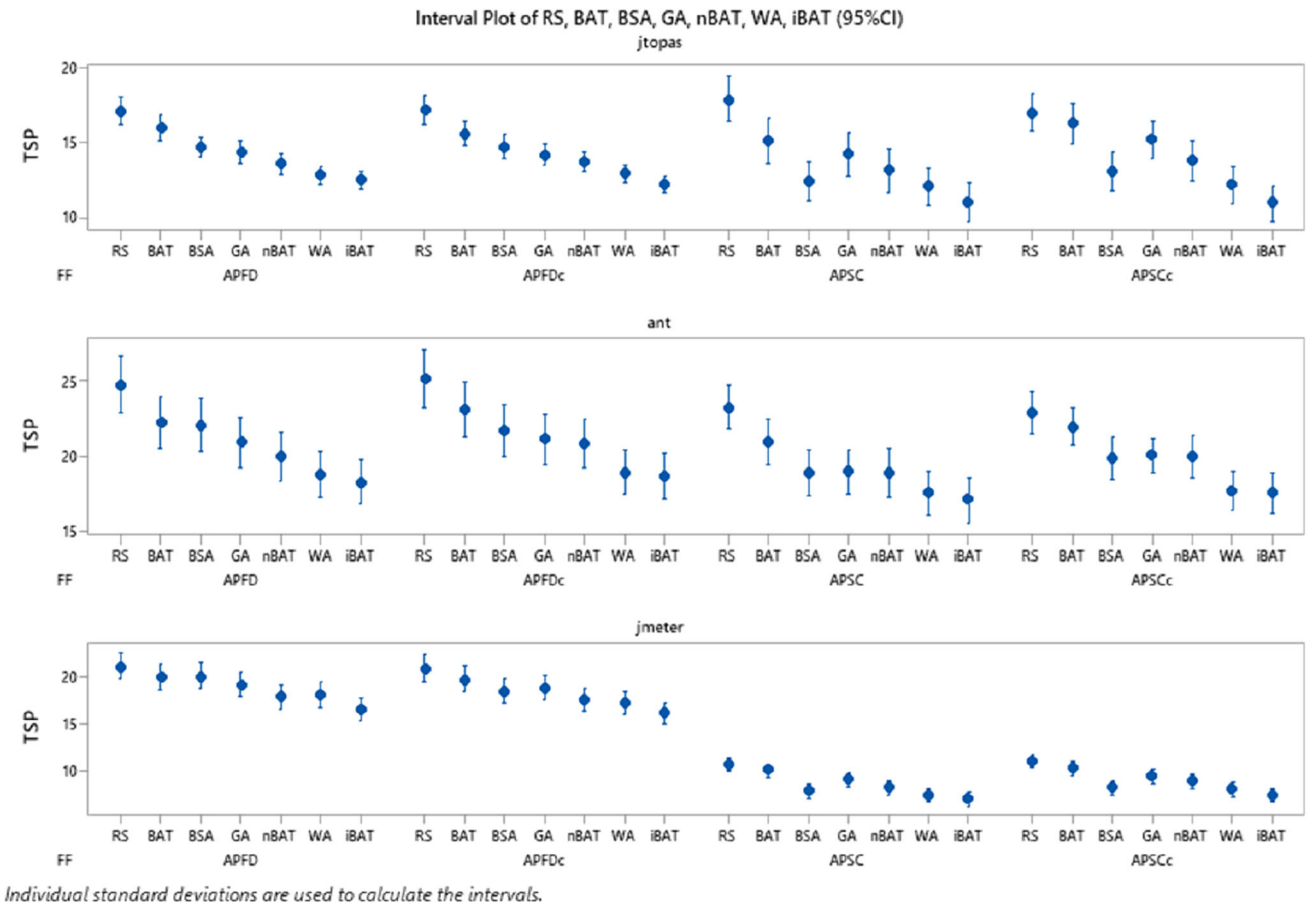

Fig. 5 Interval plots of RS, BAT, BSA, GA, nBAT, WA and iBAT for test selection percentage using APFD, APFDc, APSC and APSCc

shows that the fault loss in APSC and APSCc is higher compared to the statement loss in APFD and APFDc. Again, the reason behind this is that the redundancy in the test suite due to statement coverage is high, and if we reduce the test suite according to fault coverage, then there is less reduction in coverage. It can be inferred that the decrease in test suite size and loss in coverage are inversely proportional to each other.

Another point to consider is that if we reduce the size of the test suite according to one criterion, it may lose some percentage of other measures. It may happen due to the high reduction in test suite size. If we perform moderate selection, then this problem may be solved. Alternatively, we need to make a trade-off between the two criteria to produce optimal results by addressing it as a multi-objective problem.

6.2.4 RQ 2.4 Impact on cost reduction percentage

The cost reduction is directly proportional to the test case minimization, i.e., the higher the decline in test suite size, the lower the execution cost of the test suite. Figure 7 and Table 8 show that the cost reduction percentage of the proposed algorithm is higher than the other algorithms, comparatively. However, GA performed better for APSCc in two out of three programs. Observations also suggest that the second-highest cost reduction is by GA, BSA after WA for fault and statement coverage. Therefore, the iBAT, GA, WA, nBAT, BSA, BAT, and RS follow it in the succeeding manner for fault coverage. On the other side, they are ranked as iBAT, WA, BSA, GA, nBAT, BAT, and RS for statement coverage.

6.3 Major findings and further discussions

The main findings of this research are as follows:

- All the nature-inspired algorithms are statistically better than the random search for prioritizing the test cases on all the testing criteria.
- There is a statistically significant difference between the performances of the pairs of algorithms except (BAT, BSA), (nBAT, GA), (nBAT, BSA), (nBAT, WA), and (GA, BSA) for all the testing criteria of test case prioritization. Though iBAT and WA are

Table 8 Coverage loss, fault detection loss and the cost reduction percentage of the algorithms

Program Versions	Algorithms	Fitness functions wise Coverage Loss, Fault Loss and Cost Reduction Percentages							
		Coverage Loss (%)		Fault Loss (%)		Cost Reduction (%)			
		CLP_{APFD}	CLP_{APFDc}	FLP_{APSC}	FLP_{APSCc}	CRP_{APFD}	CRP_{APFDc}	CRP_{APSC}	CRP_{APSCc}
jtopas	iBAT	**9.647**	**9.437**	30.904	29.729	**86.225**	**86.953**	**89.728**	**89.203**
	WA	11.338	11.561	31.531	31.258	85.695	86.098	89.010	87.986
	nBAT	13.628	14.585	26.802	26.757	85.119	85.439	87.371	85.226
	GA	10.962	11.173	26.442	24.407	85.711	86.474	85.911	88.355
	BSA	12.608	13.330	29.424	28.002	84.721	84.967	88.014	87.166
	BAT	12.062	10.914	24.745	24.238	83.888	84.005	85.110	84.440
	RS	13.393	15.393	**24.414**	**21.893**	82.646	82.697	83.078	83.215
ant	iBAT	**6.167**	**6.695**	21.502	20.666	**79.912**	**80.712**	**82.828**	81.841
	WA	7.441	7.191	20.849	20.331	78.517	78.854	82.146	80.929
	nBAT	8.047	8.006	20.190	18.499	78.012	78.478	80.859	79.356
	GA	7.388	7.489	19.768	17.432	78.621	78.732	80.849	**82.022**
	BSA	8.071	8.203	19.278	18.649	76.938	77.920	81.037	79.962
	BAT	6.855	6.889	17.682	**16.304**	76.648	76.193	78.882	78.046
	RS	7.693	7.729	**17.228**	17.710	74.477	74.131	76.864	76.775
jmeter	iBAT	5.386	5.002	30.476	30.977	**82.083**	82.296	**92.938**	91.312
	WA	4.894	5.293	31.756	31.656	81.808	**82.436**	92.447	91.858
	nBAT	5.012	5.587	29.220	28.425	81.168	80.811	91.545	90.047
	GA	5.225	5.190	**27.429**	**25.375**	81.827	81.862	90.602	**92.328**
	BSA	5.109	5.310	31.633	31.011	80.670	81.484	91.801	91.538
	BAT	**4.829**	**4.878**	28.167	26.422	79.777	79.832	89.599	89.327
	RS	5.230	5.298	28.070	27.897	79.413	78.823	89.225	88.240

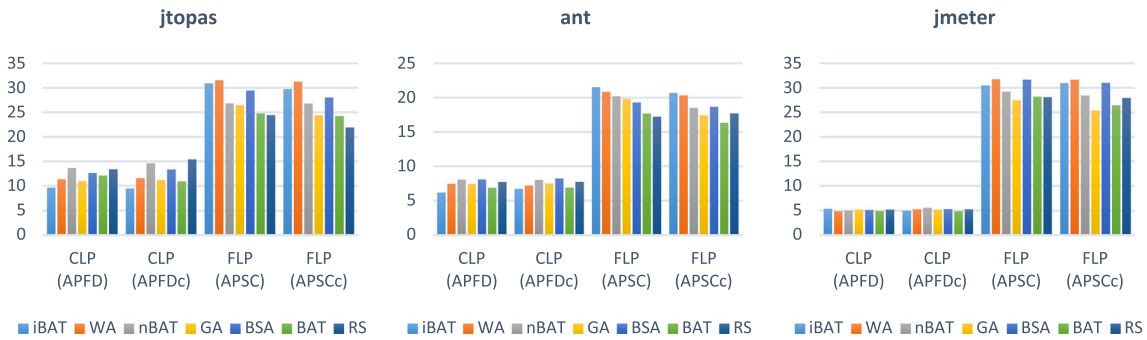

Fig. 6 Coverage loss percentage of the algorithms for different testing criteria

statistically insignificant for statement coverage, the numerical mean values of iBAT are higher than WA.

- It is observed that the iBAT and WA have statistically better results than other algorithms for test case minimization though they are apparently different in most of the cases. However, the iBAT outperformed the WA algorithm in terms of the numerical mean.
- The test case minimization has not witnessed the statistically significant difference between the pairs of (nBAT, BSA), (nBAT, WA), and (GA, BSA) in half of the cases during the programwise analysis.

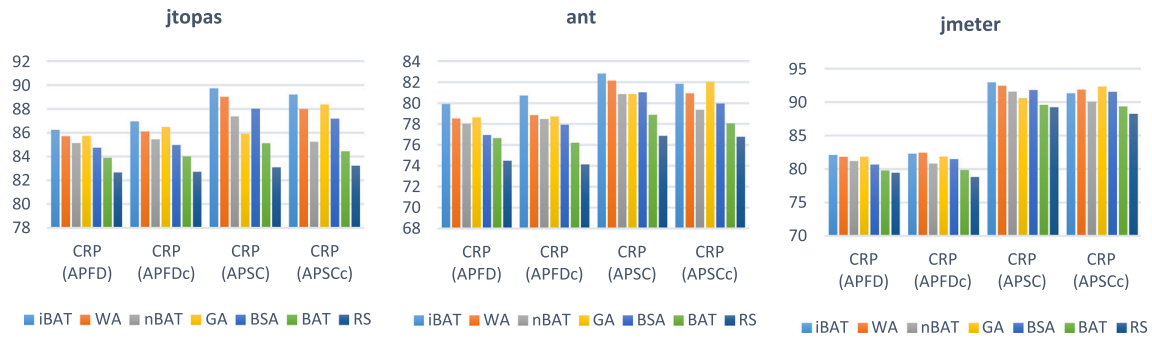

Fig. 7 Cost reduction percentage of the algorithms for different testing criteria

- The fault detection capability loss in code coverage-based testing criteria is higher than the statement coverage loss in the fault coverage-based approaches. Generally, we perform the test suite reduction based on statement coverage. Still, this study reveals that if we minimize the test suite based on statement redundancy, it may leave some of the faults undetected, which is an unfavourable condition. BAT and RS showed a minor loss in coverage compared to other algorithms for statement coverage, while iBAT is better for fault coverage criteria. The test selection percentage and coverage loss are inversely proportional to each other.
- The cost reduction percentage is directly proportional to the test suite reduction percentage. The iBAT outperformed other algorithms.

To further investigate the performance of the algorithms compared to each other, we extend the analysis of the algorithms on different versions of the program as shown in Tables 9 and 10. Version-based validation of the algorithms' performance suggests a significant difference in the mean values of the performance metrics for the test case prioritization and minimization. Version analysis also shows that nBAT, BSA, and GA have almost similar performance; however, BSA performed better than nBAT and GA in most cases for statement coverage criteria. WA is always superior to these, so it follows the order iBAT, WA, nBAT, GA, BSA, BAT, and RS for fault coverage and iBAT, WA, BSA, nBAT, GA, BAT, and RS in case of statement coverage criteria. Therefore, both the program and version analysis suggests that the proposed algorithm outperformed the other algorithms for APFD, APFDc and equivalent with WA for APSC and APSCc.

Program analysis shows that iBAT and WA have equivalent performance in several cases of test case minimization. In contrast, the statistical analysis on versions reveals a significant difference in the means of test selection percentages for both the testing criteria except a few versions for APSC. Similarly, the (nBAT, BSA), (GA, BSA) pairs are statistically insignificant for the code coverage criteria and APFDc but different for APFD. The CLP is least for iBAT, while BAT and RS have the least FLP for most versions. However, iBAT wins the race for the cost reduction percentage. Therefore, the observations suggest that the TCM procedure tells about the precision of the prioritization. Even a minor difference in the value of the performance metric affects the ranking of the test cases. It is also observed that the algorithms performed differently with different testing criteria. Overall, iBAT algorithm is superior to all algorithms for the test case prioritization and minimization.

7 Conclusions and future work

We have proposed an improved novel bat algorithm for TCP and TCM and compared it with the random search and other nature-inspired algorithms like BSA, BAT, GA, nBAT, and WA. The results prove that the iBAT algorithm performed better than these algorithms for all the performance metrics. The statistical test confirmed the superiority of the proposed algorithm for test case prioritization. Moreover, boxplots and interval plots demonstrated the effectiveness of the iBAT algorithm. Also, the proposed algorithm and the whale optimization algorithm showed similar performance for statement coverage in test case prioritization. On the other side, the iBAT algorithm worked better than the WA for reducing the test suite in terms of statement coverage and cost reduction percentage. Our future work includes the development of a test case selection technique that selects an affordable number of test cases without compromising the quality of the software. We will also explore other versions of bat algorithms or hybridized bat algorithms with different nature-inspired

Table 9 Comparison of the algorithms for different fitness functions

Program Versions	Algorithms	Fitness functions wise Performance and Tukey Group Ranking (%)								Fitness functions wise TSP and Tukey Group Ranking (%)							
		APFD	TR	APFDc	TR	APSC	TR	APSCc	TR	TSP_{APFD}	TR	TSP_{APFDc}	TR	TSP_{APSC}	TR	TSP_{APSCc}	TR
Jtopas v1	iBAT	97.736	A	98.047	A	99.105	A	98.787	A	7.000	A	7.000	A	5.167	A	5.467	A
	WA	96.846	B	97.675	B	98.997	A	98.676	A	7.667	B	7.667	B	5.467	AB	6.467	B
	nBAT	96.144	B	97.600	B	98.955	B	98.292	B	7.833	B	8.200	B	5.500	B	7.033	B
	GA	94.832	C	97.518	B	98.697	B	97.930	C	8.200	B	7.167	B	6.567	C	8.333	C
	BSA	95.903	D	97.528	B	98.976	A	98.345	B	8.000	B	8.000	B	6.367	C	7.200	B
	BAT	93.912	E	96.592	C	98.388	C	97.623	C	9.000	C	9.067	C	6.833	C	9.533	D
	RS	92.876	F	95.441	D	97.293	D	96.783	D	9.467	C	9.867	D	9.533	D	10.700	E
Jtopas v2	iBAT	94.954	A	95.815	A	97.047	A	96.183	A	13.000	A	12.900	A	21.933	A	21.567	A
	WA	93.164	B	94.449	B	96.850	A	95.893	A	13.700	B	13.400	B	22.133	B	22.400	B
	nBAT	92.154	C	94.012	BC	96.226	B	95.496	B	13.767	B	14.500	C	23.467	C	23.567	C
	GA	91.390	D	93.374	CD	96.220	B	95.071	C	14.900	C	14.067	C	23.867	C	23.867	C
	BSA	90.570	E	93.145	D	96.514	B	95.485	B	15.500	CD	14.933	C	23.567	C	23.633	C
	BAT	89.467	F	92.394	E	95.805	C	94.713	D	15.867	D	15.400	D	25.633	D	25.667	D
	RS	88.018	G	87.922	F	95.119	D	94.315	E	17.000	E	17.167	E	27.467	E	25.800	D
Jtopas v3	iBAT	95.231	A	94.199	A	97.918	A	97.261	A	13.900	A	13.500	A	10.667	A	10.300	A
	WA	94.032	B	93.260	B	97.748	A	96.995	A	14.000	A	14.267	B	11.667	B	12.800	B
	nBAT	93.045	C	92.377	C	96.616	B	96.007	B	15.700	B	15.033	C	16.733	D	16.667	C
	GA	91.868	D	91.588	D	96.183	B	95.147	B	15.800	B	15.367	C	19.100	E	18.433	D
	BSA	90.167	E	91.259	D	96.717	B	96.011	B	16.300	B	15.967	C	13.033	C	13.200	B
	BAT	89.498	F	89.538	E	94.715	C	94.018	C	17.700	C	17.267	D	19.633	E	19.400	E
	RS	87.689	G	86.122	F	92.830	D	93.064	D	18.700	D	18.000	D	22.867	F	19.833	E
Jtopas v4	iBAT	94.563	A	93.026	A	99.123	A	97.688	A	15.200	A	15.300	A	5.233	A	6.467	A
	WA	91.680	B	91.094	B	99.050	A	97.605	A	15.967	B	16.267	B	5.900	AB	6.900	AB
	nBAT	91.130	B	90.021	B	99.026	A	97.049	B	17.600	C	17.500	C	6.733	BC	7.833	B
	GA	89.581	C	88.725	C	98.753	B	96.568	C	18.567	D	18.533	C	7.267	C	10.067	D
	BSA	88.079	D	84.887	D	99.070	A	97.087	B	19.867	E	19.900	D	6.733	BC	8.200	C
	BAT	86.928	E	84.256	D	98.544	C	96.045	D	21.400	F	20.533	D	8.367	D	10.433	D
	RS	83.664	F	80.384	E	96.971	D	95.550	E	23.167	G	23.500	E	11.533	E	11.600	E
Ant v1	iBAT	91.096	A	90.890	A	99.158	A	98.127	A	21.433	A	21.600	A	4.267	A	5.900	A
	WA	89.167	B	90.002	B	99.004	B	98.100	A	22.867	B	22.400	B	5.100	AB	6.267	B
	nBAT	88.330	B	88.400	C	99.007	B	97.786	B	23.867	C	23.933	C	5.433	B	8.567	C
	GA	87.278	C	86.391	D	98.668	D	97.598	C	24.733	D	24.033	C	6.700	C	10.900	D
	BSA	86.147	D	84.908	E	98.867	C	97.902	B	25.633	E	24.800	C	6.633	C	8.200	C
	BAT	85.973	D	84.365	E	98.312	E	97.015	D	25.800	F	25.467	D	8.333	D	11.700	DE
	RS	83.896	E	81.186	F	97.491	F	96.666	E	27.133	G	26.633	E	10.767	E	11.833	E

Table 9 (continued)

Program Versions	Algorithms	Fitness functions wise Performance and Tukey Group Ranking (%)								Fitness functions wise TSP and Tukey Group Ranking (%)							
		APFD	TR	APFDc	TR	APSC	TR	APSCc	TR	TSP_{APFD}	TR	TSP_{APFDc}	TR	TSP_{APSC}	TR	TSP_{APSCc}	TR
Ant v2	iBAT	92.366	A	93.286	A	99.017	A	98.196	A	20.767	A	20.038	A	5.433	A	6.500	A
	WA	92.085	A	92.631	B	98.929	A	97.878	B	20.867	A	20.733	B	5.900	B	7.367	B
	nBAT	90.545	B	91.649	C	98.964	A	97.483	C	21.733	B	23.467	C	6.033	B	8.767	C
	GA	89.073	C	91.548	C	98.775	B	97.170	D	23.333	C	23.600	C	6.933	C	10.333	D
	BSA	88.094	D	91.405	C	98.916	A	97.700	B	24.467	D	23.800	C	6.400	B	7.900	B
	BAT	88.138	D	88.438	D	98.636	C	96.539	E	26.467	E	26.833	D	9.067	D	11.767	E
	RS	85.374	E	86.203	E	97.698	D	96.281	F	29.400	F	29.267	E	12.433	E	11.933	E
Ant v3	iBAT	95.223	A	94.223	A	98.971	A	97.844	A	16.633	A	17.000	A	5.567	A	7.400	A
	WA	94.280	B	93.260	B	98.923	A	97.689	AB	17.400	B	17.567	B	6.067	AB	8.733	B
	nBAT	92.556	C	91.572	C	98.731	B	97.559	B	18.133	C	19.300	C	7.300	BC	10.467	C
	GA	91.338	D	92.118	C	98.687	B	97.382	C	19.467	D	19.367	C	8.033	C	12.733	D
	BSA	90.214	E	90.205	D	98.736	B	97.667	B	20.667	E	20.633	C	8.200	D	10.433	C
	BAT	89.034	F	88.311	E	98.466	C	97.278	C	21.533	F	21.367	D	9.967	E	12.900	D
	RS	83.228	G	82.835	F	97.940	D	97.104	D	24.400	G	25.200	E	13.733	F	13.867	E
Ant v4	iBAT	98.848	A	98.211	A	95.707	A	95.055	A	3.300	A	3.000	A	22.400	A	22.667	A
	WA	98.660	A	98.181	A	95.140	A	94.421	B	3.600	B	3.533	B	23.367	B	22.867	A
	nBAT	98.110	B	98.066	A	94.043	B	93.150	C	3.700	B	3.867	B	24.233	C	23.633	B
	GA	97.974	B	98.056	A	93.306	C	93.682	C	3.833	B	3.733	B	25.467	D	23.433	B
	BSA	97.786	B	98.100	A	94.312	B	92.996	D	3.933	B	3.800	B	23.367	B	25.267	C
	BAT	97.279	C	98.041	A	92.555	D	91.967	E	3.867	B	3.900	B	25.933	D	25.567	C
	RS	96.750	D	96.242	B	91.780	E	91.203	E	4.800	C	5.167	C	28.300	E	25.733	C
Ant v5	iBAT	98.518	A	97.642	A	95.905	A	95.418	A	3.333	A	3.700	A	22.733	A	22.133	A
	WA	98.553	A	97.238	A	95.196	B	94.887	A	3.433	B	4.400	B	23.767	B	23.100	B
	nBAT	97.646	B	96.611	B	93.538	D	93.872	B	3.633	B	4.633	B	25.000	C	24.467	C
	GA	97.341	B	96.579	B	94.541	C	93.939	B	3.933	BC	4.400	B	23.867	B	23.433	B
	BSA	96.326	C	96.276	BC	94.189	C	93.981	B	4.333	CD	5.333	C	24.567	C	25.400	C
	BAT	96.232	C	95.691	C	92.877	E	93.737	B	4.600	D	5.533	C	25.867	C	26.033	D
	RS	91.572	D	95.078	D	91.985	F	91.541	C	5.867	E	5.767	C	26.633	D	26.400	D

Table 9 (continued)

Program Versions	Algorithms	Fitness functions wise Performance and Tukey Group Ranking (%)								Fitness functions wise TSP and Tukey Group Ranking (%)							
		APFD	TR	APFDc	TR	APSC	TR	APSCc	TR	TSP_{APFD}	TR	TSP_{APFDc}	TR	TSP_{APSC}	TR	TSP_{APSCc}	TR
Ant v6	iBAT	90.759	A	89.547	A	96.383	A	95.667	A	33.433	A	32.200	A	27.733	A	27.000	A
	WA	89.852	B	88.588	B	96.199	A	95.134	B	33.500	A	33.667	B	28.700	B	28.533	B
	nBAT	88.060	C	86.810	C	95.209	C	93.963	D	37.333	B	37.980	C	30.600	C	30.733	C
	GA	87.448	D	86.658	C	95.736	B	94.577	C	38.233	C	38.233	CD	30.467	C	28.767	B
	BSA	86.961	E	85.843	D	95.120	C	94.039	D	39.533	D	39.033	D	31.633	D	31.233	D
	BAT	85.558	F	84.052	E	94.428	D	93.025	E	40.800	E	41.200	E	31.867	D	31.633	D
	RS	84.258	G	82.811	F	93.885	E	92.202	F	43.300	F	43.200	F	33.933	E	34.467	E
Ant v7	iBAT	91.405	A	91.756	A	97.232	A	96.325	A	29.200	A	30.400	A	29.667	A	28.600	A
	WA	90.450	B	92.561	B	97.279	A	96.298	A	29.900	B	31.200	B	30.800	B	29.933	B
	nBAT	89.930	B	89.749	C	96.606	C	95.682	BC	31.467	C	34.233	C	34.067	C	33.233	C
	GA	88.864	C	89.868	C	96.888	B	95.578	C	33.000	D	34.833	C	31.567	B	30.800	B
	BSA	88.191	C	89.645	C	96.894	B	95.936	B	36.000	E	34.700	C	31.733	B	30.700	B
	BAT	88.446	C	87.539	D	96.194	D	95.102	D	33.000	D	37.733	D	35.667	D	34.400	D
	RS	86.526	D	84.901	E	95.368	E	94.927	D	38.700	E	41.233	E	37.267	E	36.133	E
Jmeter v1	iBAT	93.395	A	96.968	A	99.542	A	99.401	A	14.084	A	14.340	A	2.633	A	3.000	A
	WA	90.623	B	94.675	B	99.527	A	99.372	A	15.634	B	15.400	B	2.884	AB	3.550	AB
	nBAT	90.835	B	95.255	B	99.498	A	98.936	C	15.134	B	15.867	B	2.950	B	4.234	C
	GA	89.502	C	90.797	C	98.970	B	98.677	D	16.717	C	17.067	C	3.934	C	5.367	D
	BSA	87.699	D	90.657	C	99.512	A	99.172	B	17.350	D	17.017	C	2.967	B	3.650	B
	BAT	87.659	D	89.872	D	98.838	C	98.340	E	17.050	D	17.500	C	6.250	D	6.434	E
	RS	86.524	E	89.156	D	97.395	D	98.168	F	18.317	E	18.400	D	6.700	E	7.534	F
Jmeter v2	iBAT	93.627	A	95.997	A	99.3512	A	99.600	A	17.700	A	17.800	A	2.700	A	3.350	A
	WA	92.509	B	94.203	B	99.2033	B	99.598	A	20.167	B	19.834	B	3.100	B	4.417	C
	nBAT	92.712	B	94.405	B	99.0196	C	99.485	B	19.950	B	19.767	B	4.250	C	5.667	D
	GA	90.841	C	92.132	C	98.8673	D	99.514	B	21.217	C	20.434	B	5.467	D	5.184	D
	BSA	89.371	D	94.291	B	99.0494	C	99.516	B	22.434	D	21.667	C	3.367	B	3.917	B
	BAT	89.342	D	89.988	D	98.6261	E	99.325	C	22.200	D	22.900	D	3.934	B	6.534	E
	RS	87.987	E	89.211	E	98.4997	F	99.270	C	23.584	E	23.417	D	7.234	E	7.284	E

Table 9 (continued)

Program Versions	Algorithms	Fitness functions wise Performance and Tukey Group Ranking (%)								Fitness functions wise TSP and Tukey Group Ranking (%)							
		APFD	TR	APFDc	TR	APSC	TR	APSCc	TR	TSP_{APFD}	TR	TSP_{APFDc}	TR	TSP_{APSC}	TR	TSP_{APSCc}	TR
Jmeter v3	iBAT	89.952	A	91.248	A	99.3616	A	98.939	A	24.667	A	24.543	A	4.584	A	5.584	A
	WA	88.013	B	87.644	B	99.2829	B	98.886	A	27.667	B	26.684	B	5.984	B	5.984	A
	nBAT	87.797	B	88.073	B	99.1926	BC	98.605	C	27.184	B	26.817	B	6.067	C	7.500	C
	GA	86.528	C	84.054	D	99.1334	C	98.444	D	29.050	C	28.734	C	7.584	D	7.634	C
	BSA	85.590	CD	86.641	C	99.249	B	98.748	B	29.800	C	28.217	C	6.867	C	6.317	B
	BAT	84.697	D	81.788	E	98.9185	D	98.204	E	29.984	C	30.084	D	8.512	E	8.417	D
	RS	82.910	E	80.298	F	98.8651	D	97.787	F	31.084	D	31.200	E	8.684	E	9.117	E
Jmeter v4	iBAT	97.721	A	98.833	A	98.724	A	97.968	A	3.850	A	3.817	A	9.417	A	11.200	A
	WA	96.888	B	97.615	B	98.629	A	97.536	B	4.184	B	4.367	B	9.600	A	11.417	AB
	nBAT	96.400	B	98.102	C	98.1	B	96.611	D	4.150	B	4.417	B	10.717	B	11.600	B
	GA	95.538	C	97.277	C	97.403	C	96.160	E	4.784	C	4.884	B	12.300	C	12.484	C
	BSA	94.800	CD	97.570	C	98.117	B	97.158	C	5.434	D	4.584	C	10.634	B	11.584	B
	BAT	94.743	D	96.444	D	95.856	D	95.692	F	5.567	D	4.950	C	12.650	C	12.567	C
	RS	93.167	E	95.564	E	95.043	E	95.211	G	6.267	E	6.117	D	13.484	D	13.517	D
Jmeter v5	iBAT	93.786	A	92.208	A	97.954	A	97.784	A	22.234	A	19.484	A	14.417	A	13.750	A
	WA	92.653	B	93.992	B	97.936	A	97.352	B	22.767	B	20.790	B	14.867	A	15.200	B
	nBAT	91.969	B	90.372	D	97.315	B	96.994	C	23.117	B	21.384	C	15.717	B	15.700	B
	GA	90.952	C	91.033	C	97.283	B	96.533	D	24.134	C	21.117	C	15.984	B	16.334	C
	BSA	89.517	D	90.011	D	97.35	B	97.050	BC	25.334	D	22.084	C	15.667	B	15.834	B
	BAT	89.283	D	87.993	E	96.503	C	96.031	E	25.367	D	23.234	D	16.034	C	17.517	D
	RS	88.206	E	85.739	F	95.962	D	95.563	F	26.334	E	25.367	E	17.650	D	17.884	D

Table 10 Coverage loss, fault detection loss and the cost reduction percentage of the algorithms

Program Versions	Algorithms	Fitness functions wise Coverage Loss, Fault Loss and Cost Reduction Percentages							
		Coverage Loss (%)		Fault Loss (%)		Cost Reduction (%)			
		CLP_{APFD}	CLP_{APFDc}	FLP_{APSC}	FLP_{APSCc}	CRP_{APFD}	CRP_{APFDc}	CRP_{APSC}	CRP_{APSCc}
Jtopas v1	iBAT	**4.290**	**3.432**	27.917	27.500	**88.981**	**89.549**	**93.320**	**92.575**
	WA	4.754	5.331	27.361	25.833	88.914	89.439	93.056	92.324
	nBAT	5.401	4.698	25.278	23.610	88.787	88.502	92.836	90.504
	GA	5.176	5.373	25.972	22.360	89.270	89.893	91.674	91.905
	BSA	5.049	5.443	28.056	24.440	88.878	89.189	91.668	90.502
	BAT	4.459	5.176	**24.306**	19.861	88.626	88.198	91.181	89.098
	RS	5.148	4.318	20.560	**17.500**	88.280	87.503	89.006	87.192
Jtopas v2	iBAT	**16.034**	**17.856**	33.222	33.560	**88.396**	**88.806**	**82.718**	**81.540**
	WA	19.610	19.850	34.220	34.000	88.000	87.569	82.328	79.611
	nBAT	26.540	30.720	30.560	31.000	87.621	87.733	80.904	79.368
	GA	17.870	17.814	31.440	30.440	88.530	88.439	78.977	82.001
	BSA	23.800	26.890	33.110	34.000	87.309	86.815	80.491	80.284
	BAT	23.450	19.020	31.330	28.780	86.235	86.265	77.540	78.906
	RS	27.910	37.360	**27.330**	**26.890**	84.739	85.126	76.998	77.558
Jtopas v3	iBAT	**16.112**	**14.280**	27.093	25.780	**83.624**	**85.700**	**89.473**	**89.715**
	WA	16.995	17.742	30.543	32.326	83.070	84.614	88.198	88.775
	nBAT	17.830	18.040	21.400	22.290	82.962	84.305	83.561	82.256
	GA	16.920	17.076	19.920	19.110	83.002	84.963	81.282	87.381
	BSA	18.320	18.117	23.915	23.876	82.494	83.793	87.200	87.142
	BAT	17.049	16.606	**17.600**	20.850	81.933	82.181	80.984	81.370
	RS	17.221	16.278	19.380	**18.488**	80.947	81.418	79.059	80.622
Jtopas v4	iBAT	**2.152**	**2.180**	35.385	32.077	**83.900**	**83.755**	**93.402**	**92.980**
	WA	3.994	3.319	34.000	32.872	82.795	82.770	92.459	91.235
	nBAT	4.740	4.880	29.970	30.128	81.107	81.216	92.181	88.774
	GA	3.882	4.430	28.436	25.718	82.042	82.601	91.711	92.131
	BSA	3.263	2.869	32.615	29.692	80.202	80.069	92.696	90.735
	BAT	3.291	2.855	**25.744**	27.462	78.756	79.375	90.735	88.385
	RS	3.291	3.615	30.385	**24.692**	76.619	76.742	87.247	87.489
Ant v1	iBAT	**0.155**	**0.197**	30.567	29.113	**77.550**	**78.862**	**95.786**	**93.108**
	WA	0.338	0.239	29.858	29.255	76.603	77.794	94.208	91.269
	nBAT	0.577	0.492	30.887	27.695	76.512	77.109	94.555	88.777
	GA	0.295	0.309	29.113	24.787	76.862	77.356	92.671	92.192
	BSA	0.535	0.422	28.156	27.624	74.868	75.126	92.792	91.310
	BAT	0.169	0.324	27.837	**24.362**	74.546	74.269	91.490	88.109
	RS	0.338	0.394	**25.993**	24.787	73.408	73.363	88.555	87.547
Ant v2	iBAT	**1.406**	**1.758**	33.627	32.000	**80.993**	**81.219**	**94.080**	**92.512**
	WA	2.574	2.447	32.431	29.882	78.549	79.281	93.950	90.809
	nBAT	4.051	3.812	32.765	27.725	78.344	77.626	93.634	90.143
	GA	1.575	1.786	31.255	25.667	79.123	77.909	93.348	93.311
	BSA	3.615	4.712	32.314	29.490	75.297	78.358	93.987	92.165
	BAT	2.166	1.828	29.765	**23.784**	78.260	74.804	90.969	88.535
	RS	3.629	3.826	**24.882**	28.431	71.730	72.550	87.487	88.238

Table 10 (continued)

Program Versions	Algorithms	Fitness functions wise Coverage Loss, Fault Loss and Cost Reduction Percentages							
		Coverage Loss (%)		Fault Loss (%)		Cost Reduction (%)			
		CLP_{APFD}	CLP_{APFDc}	FLP_{APSC}	FLP_{APSCc}	CRP_{APFD}	CRP_{APFDc}	CRP_{APSC}	CRP_{APSCc}
Ant v3	iBAT	**0.633**	**1.308**	42.222	36.470	**80.467**	**80.708**	**93.194**	**89.245**
	WA	1.238	1.308	40.350	37.890	79.878	78.482	92.833	91.540
	nBAT	1.660	1.702	38.530	35.093	78.768	79.193	91.896	86.754
	GA	1.322	1.617	34.490	26.450	80.513	80.202	91.746	92.473
	BSA	1.224	1.238	35.880	34.167	77.628	77.801	91.655	88.672
	BAT	1.027	0.872	**30.080**	**29.690**	76.856	77.166	89.330	86.637
	RS	1.252	1.055	31.330	30.910	73.926	72.841	85.953	85.940
Ant v4	iBAT	**15.280**	**16.199**	0.000	0.000	**95.713**	**95.839**	**77.559**	**77.008**
	WA	17.397	16.985	0.000	0.000	95.569	95.581	76.811	76.266
	nBAT	17.844	17.424	0.000	0.000	95.454	95.326	75.782	76.051
	GA	18.997	18.351	0.000	0.000	95.536	95.756	74.537	77.451
	BSA	18.965	18.825	0.000	0.000	95.392	95.634	76.361	74.776
	BAT	16.514	16.322	0.000	0.000	94.679	95.166	73.874	74.416
	RS	18.558	17.914	0.000	0.000	93.449	93.997	73.358	73.308
Ant v5	iBAT	**10.850**	**13.111**	0.000	0.000	**97.778**	**95.933**	**77.545**	**77.786**
	WA	12.351	12.425	0.000	0.000	96.364	95.586	76.537	75.031
	nBAT	14.035	13.317	0.000	0.000	96.175	95.220	74.681	76.663
	GA	12.488	12.546	0.000	0.000	96.265	95.544	75.672	76.729
	BSA	13.918	13.845	0.000	0.000	95.700	94.679	75.372	74.429
	BAT	12.440	13.328	0.000	0.000	95.318	94.514	73.835	73.844
	RS	14.206	14.440	0.000	0.000	94.244	94.098	73.266	72.479
Ant v6	iBAT	**11.186**	**10.976**	25.476	28.386	**64.633**	**63.994**	**71.735**	**72.511**
	WA	13.378	12.533	23.730	25.053	60.500	60.872	71.717	71.783
	nBAT	13.744	13.967	23.122	22.434	59.660	59.575	70.148	71.210
	GA	12.652	12.978	23.492	25.952	60.363	59.444	69.990	72.125
	BSA	13.134	13.422	**20.714**	20.926	59.358	59.423	69.126	69.006
	BAT	11.391	11.141	20.899	**20.767**	56.315	55.944	68.849	68.962
	RS	11.555	11.680	22.460	23.413	55.385	53.805	66.806	66.290
Ant v7	iBAT	**3.662**	**3.315**	18.619	18.690	**62.250**	**68.426**	**69.894**	**70.717**
	WA	4.808	4.400	19.571	20.238	62.156	64.380	68.965	69.805
	nBAT	4.421	5.327	16.024	16.548	61.174	65.298	65.314	65.891
	GA	4.388	4.835	20.024	19.167	61.683	64.913	67.978	69.871
	BSA	5.104	4.960	17.881	18.333	60.326	64.418	67.969	69.376
	BAT	4.275	4.408	**15.190**	**15.524**	60.563	61.487	63.826	65.816
	RS	4.316	4.796	15.929	16.429	59.196	58.264	62.620	63.620
Jmeter v1	iBAT	0.197	0.197	32.292	31.875	**83.249**	**82.941**	**96.986**	**96.309**
	WA	**0.155**	0.141	32.381	31.339	82.559	82.294	96.834	96.110
	nBAT	0.324	**0.127**	32.381	30.089	81.884	80.430	96.149	93.523
	GA	0.267	0.253	27.649	**26.786**	82.623	82.091	95.023	96.360
	BSA	0.183	0.141	33.333	32.202	81.437	81.198	96.569	95.684
	BAT	0.169	0.211	31.488	28.065	81.196	79.887	93.146	93.287
	RS	0.338	0.099	**27.768**	29.732	80.823	79.847	91.970	91.335

Table 10 (continued)

Program Versions	Algorithms	Fitness functions wise Coverage Loss, Fault Loss and Cost Reduction Percentages							
		Coverage Loss (%)		Fault Loss (%)		Cost Reduction (%)			
		CLP_{APFD}	CLP_{APFDc}	FLP_{APSC}	FLP_{APSCc}	CRP_{APFD}	CRP_{APFDc}	CRP_{APSC}	CRP_{APSCc}
Jmeter v2	iBAT	1.316	1.369	37.390	38.388	**80.841**	**80.333**	**96.524**	**95.250**
	WA	1.210	1.168	37.430	36.765	80.163	80.127	96.024	94.798
	nBAT	0.902	**0.945**	35.090	33.344	78.949	77.585	94.987	94.121
	GA	1.200	1.136	35.420	30.088	80.130	79.323	94.199	96.830
	BSA	1.263	1.157	35.161	36.140	78.855	79.741	95.183	95.997
	BAT	**1.115**	1.083	33.660	**31.480**	76.803	76.635	92.348	92.660
	RS	1.285	1.115	**33.040**	35.669	76.777	75.135	92.339	92.280
Jmeter v3	iBAT	**0.183**	0.230	35.240	33.310	**72.810**	**73.216**	**95.256**	92.457
	WA	0.210	0.216	34.310	34.707	72.552	72.991	94.280	**95.271**
	nBAT	0.223	0.196	33.951	31.836	72.392	71.345	93.047	91.570
	GA	0.216	0.210	30.350	31.227	72.513	72.025	92.495	95.303
	BSA	0.216	0.216	35.390	35.833	70.998	72.362	93.985	93.728
	BAT	0.210	0.230	**29.655**	**29.367**	69.904	70.320	91.482	91.156
	RS	0.210	**0.169**	33.565	31.890	69.341	69.925	91.380	90.062
Jmeter v4	iBAT	19.973	18.185	16.220	15.110	95.767	**95.831**	**90.314**	87.968
	WA	**17.487**	19.077	21.480	23.260	**95.814**	95.460	89.911	**88.167**
	nBAT	18.929	20.646	14.590	18.070	95.512	95.080	89.045	87.243
	GA	19.518	19.106	14.960	12.440	95.928	95.892	87.404	88.135
	BSA	18.734	18.480	24.963	21.850	95.209	95.172	89.000	88.101
	BAT	18.607	**18.090**	16.740	15.780	94.214	95.017	87.138	87.072
	RS	19.534	19.524	**16.220**	**13.630**	94.045	93.747	86.713	86.393
Jmeter v5	iBAT	5.262	5.030	31.240	36.200	77.746	79.157	**85.611**	84.577
	WA	5.407	5.865	33.180	32.210	**77.950**	**81.307**	85.187	**84.945**
	nBAT	**4.683**	6.023	30.090	28.786	77.103	79.614	84.497	83.776
	GA	4.921	5.243	**28.764**	**26.336**	77.943	79.981	83.891	85.011
	BSA	5.149	6.556	29.316	29.030	76.853	78.947	84.267	84.180
	BAT	4.544	**4.775**	29.290	27.420	76.766	77.303	83.882	82.460
	RS	4.782	5.582	29.757	28.565	76.080	75.463	83.724	81.131

algorithms and test the same on more real-world case studies for better validation.

Funding The University Grants Commission, India supports this work under the JRF-NET scheme with reference number 3469/(NET-DEC. 2014).

Data availability Enquiries about data availability should be directed to the authors.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Ahmed BS (2016) Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. Eng Sci Technol Int J 19(2):737–753

Bajaj A, Abraham A (2021) Prioritizing and minimizing test cases using dragonfly algorithms. Int J Comput Inf Syst Ind Manage Appl 13:062–071

Bajaj A, Sangwan OP (2018) A Survey on Regression Testing using Nature-Inspired Approaches. In: Proceedings of 4th International Conference on Computing, Communication and Automation (ICCCA), IEEE, pp 1–5

Bajaj A, Sangwan OP (2019a) A systematic literature review of test case prioritization using genetic algorithms. IEEE Access 7:126355–126375

Bajaj A, Sangwan OP (2019b) Study the impact of parameter settings and operators role for genetic algorithm based test case prioritization. In: Proceedings of International Conference on Sustainable Computing in Science, Technology and

Management, Available at SSRN: https://ssrn.com/abstract=3356318 or https://doi.org/10.2139/ssrn.3356318, Elsevier, pp 1564–1569

Bajaj A, Sangwan OP (2020) Nature-inspired approaches to test suite minimization for regression testing. In: Computational Intelligence Techniques and Their Applications to Software Engineering Problems CRC Press, pp 99–110

Bajaj A, Sangwan OP (2021a) Discrete and combinatorial gravitational search algorithms for test case prioritization and minimization. Int J Inf Technol 13:817–823

Bajaj A, Sangwan OP (2021b) Discrete Cuckoo search algorithms for test case prioritization. Appl Soft Comput. https://doi.org/10.1016/j.asoc.2021.107584

Bajaj A, Sangwan OP (2021c) Test case prioritization using bat algorithm. Recent Adv Comput Sci Commun. https://doi.org/10.2174/2213275912666190226154344

Bajaj A, Sangwan OP (2021d) Tri-level regression testing using nature-inspired algorithms. Innovations Syst Softw Eng 17(1):1–16

Bharathi M (2022) Hybrid particle swarm and ranked firefly metaheuristic optimization-based software test case minimization. Int J Appl Metaheuristic Comput (IJAMC) 13(1):1–20

Chouhan VK, Khan SH, Hajiaghaei-Keshteli M, Subramanian S (2020) Multi-facility-based improved closed-loop supply chain network for handling uncertain demands. Soft Comput 24:7125–7147. https://doi.org/10.1007/s00500-020-04868-x

Dhiman G, Kumar V (2018) Emperor penguin optimizer: a bio-inspired algorithm for engineering problems. Knowl-Based Syst 159:20–50

Do H, Mirarab S, Tahvildari L, Rothermel G (2010) The effects of time constraints on test case prioritization: a series of controlled experiments. IEEE Trans Software Eng 36(5):593–617

Eghbali S, Tahvildari L (2016) Test case prioritization using lexicographical ordering. IEEE Trans Softw Eng 42(12):1178–1195

Elbaum S, Malishevsky AG, Rothermel G (2002) Test case prioritization: a family of empirical studies. IEEE Trans Softw Eng 28(2):159–182

Farasat A, Menhaj MB, Mansouri T, Moghadam MRS (2010) ARO: A new model-free optimization algorithm inspired from asexual reproduction. Appl Soft Comput 10(4):1284–1292

Fister Jr I, Yang XS, Fister I, Brest J, Fister D, (2013) A brief review of nature-inspired algorithms for optimization. *arXiv preprint arXiv*, pp 1307.4186.

Gandomi AH, Yang XS (2014) Chaotic bat algorithm. J Comput Sci 5(2):224–232

Gautham S, Rajamohan J (2016) Economic load dispatch using novel bat algorithm. In: 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), IEEE, pp 1–4

Hashim NL, Dawood YS (2018) Test case minimization applying firefly algorithm. Int J Adv Sci Eng Inform Technol 8(4–2):1777–1783

Huang X, Li C, Pu Y, He B (2019) Gaussian quantum bat algorithm with direction of mean best position for numerical function optimization. Comput Intell Neurosci 2019:1–18

Kaur A, Agrawal AP (2017) A comparative study of bat and cuckoo search algorithm for regression test case selection. In: 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence, IEEE, pp 164–170

Khatibsyarbini M, Isa MA, Jawawi DN, Hamed HNA, Suffian MDM (2019) Test case prioritization using firefly algorithm for software testing. IEEE Access 7:132360–132373

Li Z, Harman M, Hierons RM (2007) Search algorithms for regression test case prioritization. IEEE Trans Softw Eng 33(4):225–237

Mahdi FP, Vasant P, Abdullah-Al-Wadud M, Kallimani V, Watada J (2019) Quantum-behaved bat algorithm for many-objective combined economic emission dispatch problem using cubic criterion function. Neural Comput Appl 31(10):5857–5869

Malishevsky AG, Ruthruff JR, Rothermel G, Elbaum S (2006) Cost-cognizant test case prioritization. Technical report TR-UNL-CSE-2006–0004, University of Nebraska-Lincoln, pp 97–106

Mann M, Tomar P, Sangwan OP (2018) Bio-inspired metaheuristics: evolving and prioritizing software test data. Appl Intell 48(3):687–702

Mansouri T, Farasat A, Menhaj MB, Moghadam MRS (2011) ARO: a new model free optimization algorithm for real time applications inspired by the asexual reproduction. Expert Syst Appl 38(5):4866–4874

Marchetto A, Islam MM, Asghar W, Susi A, Scanniello G (2015) A multi-objective technique to prioritize test cases. IEEE Trans Softw Eng 42(10):918–940

Mei H, Hao D, Zhang L, Zhang L, Zhou J, Rothermel G (2012) A static approach to prioritizing junit test cases. IEEE Trans Software Eng 38(6):1258–1275

Meng X, Liu Y, Gao X, Zhang H (2014) A new bio-inspired algorithm: chicken swarm optimization. In International conference in swarm intelligence, Springer, Cham, pp 86–94

Meng XB, Gao XZ, Liu Y, Zhang H (2015) A novel bat algorithm with habitat selection and Doppler Effect in echoes for optimization. Expert Syst Appl 42(17–18):6350–6364

Meng X, Gao X, Lu L, Liu Y, Zhang H (2016) A new bio-inspired optimisation algorithm: Bird Swarm Algorithm. J Exp Theor Artif Intell 28(4):673–687. https://doi.org/10.1080/0952813X.2015.1042530

Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67

Mohapatra SK, Prasad S (2015) Test case reduction using ant colony optimization for object oriented program. Int J Electr Comput Eng 5(6):2088–8708

Nawi NM, Rehman MZ, Khan A, Chiroma H, Herawan T (2016) A modified bat algorithm based on Gaussian distribution for solving optimization problem. J Comput Theor Nanosci 13(1):706–714

Osaba E, Yang XS, Diaz F, Lopez-Garcia P, Carballedo R (2016) An improved discrete bat algorithm for symmetric and asymmetric traveling salesman problems. Eng Appl Artif Intell 48:59–71

Osaba E, Yang XS, Fister I Jr, Del Ser J, Lopez-Garcia P, Vazquez-Pardavila AJ (2019) A discrete and improved bat algorithm for solving a medical goods distribution problem with pharmacological waste collection. Swarm Evol Comput 44:273–286

Öztürk MM (2018) A bat-inspired algorithm for prioritizing test cases. Vietnam J Computer Sci 5(2018):45–57

Riffi ME, Saji Y, Barkatou M (2017) Incorporating a modified uniform crossover and 2-exchange neighborhood mechanism in a discrete bat algorithm to solve the quadratic assignment problem. Egypt Inform J 18(3):221–232

Rothermel G, Untch RH, Chu C, Harrold MJ, (1999) Test case prioritization: An empirical study. In: Proceedings IEEE international conference on software maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360). IEEE, pp 179–188.

Saji Y, Riffi ME (2016) A novel discrete bat algorithm for solving the travelling salesman problem. Neural Comput Appl 27(7):1853–1866

Sugave SR, Patil SH, Reddy BE (2018) DIV-TBAT algorithm for test suite reduction in software testing. IET Softw 12(3):271–279

Tang J, Zhang R, Yao Y, Zhao Z, Wang P, Li H, Yuan J (2018) Maximizing the spread of influence via the collective intelligence of discrete bat algorithm. Knowl-Based Syst 160:88–103

Yang XS (2010) A new metaheuristic bat-inspired algorithm. Nature inspired cooperative strategies for optimization. Springer, Berlin, Heidelberg, pp 65–74

Yang XS, He X (2013) Bat algorithm: literature review and applications. Int J Bio-Inspired Comput 5(3):141–149

Yoo S, Harman M (2012) regression testing minimization, selection and prioritization: a survey. Softw Test Verif Reliab 22(2):67–120

Zhao D, He Y (2016) A novel binary bat algorithm with chaos and Doppler Effect in echoes for analog fault diagnosis. Analog Integr Circ Sig Process 87(3):437–450

Zhou Y, Luo Q, Xie J, Zheng H (2016) A hybrid bat algorithm with path relinking for the capacitated vehicle routing problem. In: Metaheuristics and Optimization in Civil Engineering, Springer, Cham, pp 255–276

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com