Vietnam J Comput Sci (2018) 5:45–57
https://doi.org/10.1007/s40595-017-0100-x

REGULAR PAPER

A bat-inspired algorithm for prioritizing test cases

Muhammed Maruf Öztürk[1]

Received: 20 June 2017 / Accepted: 22 August 2017 / Published online: 7 September 2017

Abstract By ordering test cases, early fault detection is focused on test case prioritization. In this field, it is widely known that algorithm and coverage criteria focused works are common. Previous works, which are related to test case prioritization, showed that practitioners need a novel method that optimizes test cases according to the cost of each test case instead of regarding the total cost of a test suite. In this work, by utilizing local and global search properties of a bat algorithm, a new bat-inspired test cases prioritization algorithm (BITCP) is proposed. In order to develop BITCP, test case execution time and the number of faults were adapted to the distance from the prey and loudness, respectively. The proposed method is then compared with four methods which are commonly used in this field. According to the results of the experiment, BITCP is superior to the conventional methods. In addition, as the complexity of the code of test cases increases, the decline in average percentage of fault detection is less in BITCP than the other four comparison algorithms produced.

Keywords Bat-inspired algorithm · Test case prioritization · Regression testing

1 Introduction

As the number of versions of a software increases, it is expected to reduce the number of defects. To accelerate this decline, various tests are prepared and run on software systems. As such, the check of a software version after required changes reveals whether the functions work well, and the check has been named as the regression test. This check is determined as exploring the presence of functional issues [1], and regression tests can either be executed in a specific time or planned in every version of a software [2].

Given a test suite T_n which consists of n test cases, test execution process should be conducted in an efficient way. This case can be called test execution effectiveness, and it depends on execution time and faults triggered by the test cases. The test suite is generated from a great number of test cases where test cases should be executed in the best order. In addition, the total cost of a test suite and individual cost of test cases should be considered while deciding the order of test cases [3]. Thus, the most effective test suite is desired to be executed as soon as possible.

Using test case prioritization, which is one type of regression tests, ordered test suites such as $T_3, T_5 \ldots T_1$ are obtained from some tests $T_i \ldots T_n$. Meanwhile, the execution of test suite reveals faults as $F_1 \ldots F_x$. Here x is the number of faults. In the minimum time of the execution [1], it is desired that the value of x becomes high as much as possible. In doing so, early fault detection reduces the effort which is required for testing.

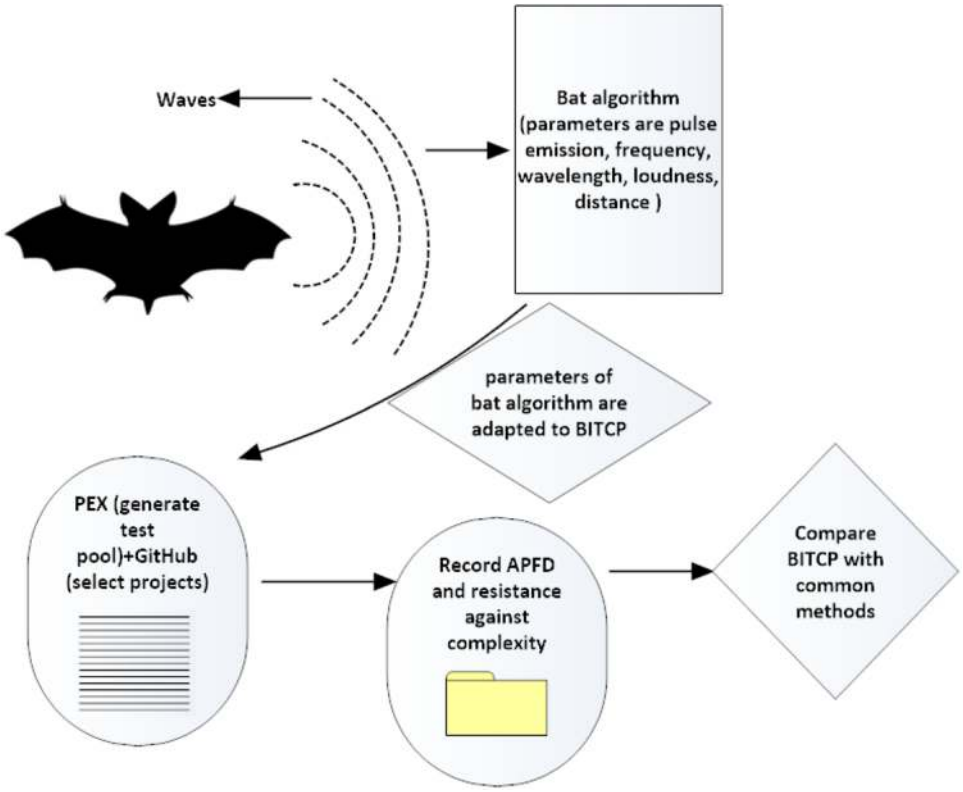
If there is a limited test source and budget allocated for testing, test cases are prioritized [4]. To figure out how test cases are executed, making required inferences is important for each version of a program. This tracing should be considered in terms of whether the test suite is correctly ordered. In test case prioritization, which was first proposed by Wong et al. [5], despite the method of performance evaluation is not changing much, the methods used for deciding the order of test cases have evolved and varied dramatically.

To perform test case prioritization, historical data are utilized. However, it may not be feasible to combine historical data with some common methods such as genetic algorithm

✉ Muhammed Maruf Öztürk
muhammedozturk@sdu.edu.tr

[1] Department of Computer Engineering, Faculty of Engineering, Suleyman Demirel University, Isparta, Turkey

Fig. 1 Main steps of the study



(GA) [6] due to the unstable experimental condition. As such, local beam search (LBS) proposed by Jiang et al. produced better APFD results than GA [7]. In this respect, this result indicates that although GA does not show bad performance in every experimental condition, superior methods need to be explored in depth.

There are many works, which are algorithm-focused, that study test case prioritization, and they include various methods that are based on particle swarm optimization (PSO) [8], GA [9], greedy search [10], and hill climbing [11]. When these works are analyzed, it is clearly seen that these works are focused on test suite cost rather than the individual cost of test cases. For instance, PSO has been evaluated in terms of some coverage criteria, but it does not involve an experiment regarding individual cost of test cases for each iteration. The structure of GA is suitable for achieving high APFD. However, it strongly depends on observing final result. On the other hand, greedy finds the global best solution step by step but it requires much time and effort. The criteria determining the cost is the total execution time of test cases in a traditional approach. Hence, a novel technique is needed regarding the individual cost of test cases that yields promising results in terms of APFD [12]. The main factor affecting the cost of test case prioritization is the computation cost of algorithms. Consequently, a new method that can fill the gap is strongly needed to prioritize the test case.

This study proposes a new test case prioritization algorithm which has been developed based upon the principles of the bat algorithm presented by Yang [13]. However, it is quite difficult to develop a prioritization algorithm which yields high APFD in various data sets, because the common methods are focused on the total cost of test cases rather than the individual cost of test cases. The main objective of the study is to develop a novel test case prioritization algorithm by regarding the individual cost of test cases. The echolocation behavior of bats and their decision mechanism of finding preys are considered while devising BITCP, and an adaptation is performed to fit the notions of the test case design. The performance of BITCP is then compared to that of ant colony optimization (ACO), greedy search, PSO, and LBS. The main steps of the study is shown in Fig. 1.

The paper aims at answering the following research questions: Research Question 1 (RQ1): What are the drawbacks of the test case prioritization techniques developed so far? Research Question 2 (RQ2): Does BITCP produce promising results with respect to APFD? Research Question 3 (RQ3): How do APFD rates of comparison algorithm change in a test suite? Research Question 4 (RQ4): What are the effects of complexity of the codes on the APFD results recorded with comparison algorithms? Research Question 5 (RQ5): Which algorithm shows the best resistance against code complexity? Research Question 6 (RQ6): Will BITCP be able to become popular among researchers to be investigated further?

The paper will contribute to the existing literature by presenting the following: (1) presentation of a nature-inspired test case prioritization algorithm which has not been employed in this field previously, (2) presentation of a proposed method that considers the individual cost of test cases rather than using total cost of test suites, (3) development of a better algorithm than traditional ones in terms of APFD by addressing some of the issues which have been encountered so far, and (4) presentation of a investigation of how faults revealed by test cases change depending on the complexity of code.

The rest of the paper has the following sections: Section 2 summarizes the related works by stressing the need for a nature-inspired algorithm. Section 3 explains how BITCP has been designed in detail. Section 4 reports the data sets and the results of the experiment. Section 5 presents the threats to validity. Last, Sect. 6 concludes the study.

2 Background and related works

In this section, the background information of test case prioritization is presented. These information comprises test cases, test pools, prioritization, and performance evaluation methods. Nature-inspired algorithms, which are selected for comparison, are then explained. Finally, the need for this study is explained by depicting the shortcomings of summarized related works.

2.1 Test case prioritization

Given a test case group $T_1, \ldots, T_n$ from a test pool P, n denotes the number of test cases. In addition, possible count of orders of a test suite T_s generated from P is as much as n factorial. The success of the ordering can be evaluated not only with code coverage and model coverage, but also with some measurement methods such as APFD [14] and APXC [11]. If a search-based optimization is preferred, the code coverage criteria is suitable. In this study, APFD is used for performance evaluation.

Let T_s denote a test suite and the order of T_s is T_r. The number of test case is n. If T_r reveals m faults of $F_1, \ldots, F_m$, first, one should eliminate well which faults are triggered by related test cases. Afterwards, the order of test case is controlled with T_r. For instance, in TF_1, T is the order of the test case of T_r which induces F_1. Likewise, inducing order of each fault is computed according to Eq. (1) which evaluates the effectiveness of T_r. T_r includes all the test cases presented in Eq. (1). Having these information, the formula emerges as in Eq. (1):

$$\text{APFD} = 1 - \frac{TF_1 + TF_2 + TF_3 + \cdots + TF_m}{nm} + \frac{1}{2n} \quad (1)$$

As known from Eq. (1), the numerator of the equation should be low so that APFD becomes high. In order to do this, the optimal case is that one test case, which is the first in the order, of a test suite reveals all of the faults. Due to the difficulty of making such a test case, the best order of test cases is investigated. As the value of APFD approaches one, the success of prioritization increases.

2.2 Nature-inspired algorithms

While choosing comparison algorithms, some steps have been done. First, similar methods have been examined in literature. In this step, the criterion is citation count and publishing date. Some of them are not suitable for comparison because of their working principles so they have been eliminated in the second step. The most suitable methods are then selected afterwards. For instance, LBS, which is involved in the comparison, is not an old method that it was published in 2015 [7].

2.2.1 Ant colony optimization

Some notions such as distance and pheromone are used for finding direction in ant colony optimization (ACO) [15] which is a heuristic method of ants, and it also helps in searching for foods. Ants leave pheromone at decision making points to show way for other ants. As the ants see the pheromone, the ants decide their directions. There is an inverse relationship between distance and decision making. Distance is a negative factor while making a route decision. In contrast, pheromones lead to a better choice decision. In a function $f(x) = qa + (1/w)b$ where these parameters are employed to input, if a and b are arbitrary generated values, the direction is computed by substituting pheromone and distance for q and w, respectively. The direction is then determined according to the obtained results. ACO has been used in following areas before: various subfields of data mining [16], real world problems [17], mathematical modeling [18], network routing problems [19], prediction of energy consumption [20], and software testing [21].

2.2.2 Particle swarm optimization

PSO was theoretically introduced by Eberhart and Kennedy to solve complex numeric optimization problems [8]. This algorithm is based on the moving behavior of bird and fish swarms. Velocities and positions are updated for each iteration in PSO. While *pbest* is the best values of each particle and it is utilized for finding optimal solution, *gbest* is the best solution of the group. Additionally, particles change their velocities and next positions by regarding previous recorded *gbest*. Note that *gbest* is recorded for every step that the best value is expected to be found at the end of the iteration [22].

PSO has been employed in many applications such as control problem of optimization, various subfields of data mining, network design, geotechnical engineering, and software quality. The common feature of these areas is that each of them needs an optimization process.

2.2.3 Greedy algorithms

One of the heuristic methods is greedy algorithm which calculates the local optimal solution in every phase of the iteration to reach global optimum solution [10]. Coin changing, huffman encoding, and traveling salesman are the types of greedy algorithm. Coin changing is selected for comparison in this study, and coin changing handles with an optimization problem by dividing it into small parts. Given an amount of money M, it is assumed that M is divided into some amounts as $n, 2n, 5n, 50n$. The main aim of coin changing algorithm is to divide M into the minimum number of coins, in which some cases, the optimal solution is more than one. We selected this algorithm because it is easy to construct and apply.

2.2.4 Local beam search

LBS was first used by Jiang ve Chan [7] in test case prioritization. LBS is one of the input-based test prioritization techniques and it was developed on the basis of adaptive prioritization [23]. Comprising all input space along with few test cases is the underlying goal of LBS which is key to detect more faults. Jiang and Chan also reported that LBS was superior to the other similar search-based algorithms such as greedy, two-optimal, hill climbing, and GA in terms of code coverage criteria.

2.2.5 Bat algorithm

Bats do hunt to get their foods. The hunting method of bats is considered as their echolocation behavior [24]. Bats have following talents for hunting:

1. They can easily find the location of their prey;
2. bats distinguish prey and obstacles;
4. bats prefer optimal location during finding prey.

Some algorithms have been developed by taking inspiration from echolocation behavior of bats. For instance, bat algorithm is a nature-inspired optimization method and it was proposed by Yang et al. [13]. Some parameters, which are constantly updated with the moves of bats, are used in bat algorithm [25]. These are v_i, f_i, x_i, r_i, and A_i. The character "i" of the parameters denotes iteration. Bats move in a position x_i along with a v_i. While v represents the velocity of a bat, f is the frequency value. If velocity is divided with frequency, wavelength is obtained with $\lambda = \frac{v}{f}$. Each bat emits ultrasonic sound, namely pulses, to get signal from a prey. Depending on the proximity of the target, pulse emission r_i increases and changes between 0 and 1. Having $r_i = 1$ means that a bat is at the target. Loudness A_i can be determined as the environmental hardship during finding prey. On the contrary, A_i decreases dramatically if a bat is near the prey and A_i becomes "0" at the target. Velocity and frequency are updated as in Eqs. (3–4) and position of a bat changes according to these iterations.

These notions are used for deciding the right path so that bats can find their prey as soon as possible. Moreover, as the distance from a bat to prey increases, the related frequency f_i reduces. Pulse emission r_i is $[0-1]$, but it depends on the wavelength emitted by a bat. The steps of the bat algorithm can be summarized as follows:

1. Starting points and velocities are determined,
2. r_i and A_i are assigned,
3. local solution is found by checking rand $> r_i$ in a specific iteration such as 20, 50, and 100,
4. new solution is calculated by flying randomly,
5. velocities and frequencies are updated,
6. global best location is denoted by x_* and it is found by ordering local solutions.

It is feasible to develop a bat-inspired test case prioritization algorithm regarding individual cost of test cases when the structure of bat algorithm is examined in detail. Moreover, distance, frequency, and loudness can be easily adapted to test case notions such as execution time, number of faults, and code coverage. This is the reason why bat algorithm has been selected as a base method in this work to develop a nature-inspired test case prioritization algorithm.

The bat algorithm has been used in a wide area applications; over the past 6 years, it has been used in gas tribune optimization [26], vehicle routing problem [27], dc motor optimization [28], and structure modeling problem [29].

2.3 Related work

This section briefly mentions the works which are directly related to our work. The need for a bat-inspired test case prioritization technique is explained by depicting the drawbacks of related works.

ACO is one of the widely known optimization methods among researchers. It has also been used for software quality to address common optimization issues. For example, mutation testing was combined with ACO to generate a test-input space [30]. Thus, from the comparison including random, hill climbing, GA, and ACO that ACO outperformed the others in terms of mutation score. In addition, ACO has promi-

nent advantages with respect to the test input generation that gives new ideas for future works [31]. Mao et al. [32] also created test inputs using ACO for branch coverage testing. They tested their method on seven different programs where simulated annealing (SA) showed better coverage performance than PSO and GA. However, ACO is not cost effective compared with PSO and SA in terms of execution time. Furthermore, employing non-industrial data sets and gathering them only from academic projects constitute main threats for the study.

ACO was also applied on data flow testing [33]. However, data sets used in this field are not comprehensive and the comparison works are not sufficient. This case creates a constraint for applying ACO to software quality. In addition, ACO was employed on test case prioritization by Singh et al. [34]. Although they compared ACO with common techniques such as random and optimal order, it has not been involved in an experiment including greedy, LBS, and PSO yet. In addition, this study did not test a large-scale data set. Prioritization techniques are either white-box or black-box focused. However, a recent study reported that using white-box or black-box focused techniques does not greatly affect the performance of prioritization techniques [35].

PSO has become popular as much as ACO in optimization practitioners. A great number of studies, which employ PSO to solve software engineering issues, are available in literature. For example, in one study, which can be considered the first to apply PSO on software testing, the design of PSO is suitable for structural software testing [36]. It performed better than genetic algorithms, and this bias was corrected by verifying testing coverage over a fitness value. Although it has detailed information regarding PSO-GA, the work is not comprehensive due to the lack of the comparison of similar optimization algorithms. Improving PSO is a convenient way to use it in generating test cases [6]. To measure the degree of this improvement, time during the iterations was measured, and the proposed method produced one-way and multi-way fitness values in a shorter time than the values recorded in PSO and GA. However, selecting two algorithms for comparison limited the scope of the work. PSO was also employed in GUI testing [37], and test coverage was examined in five different programs. However, an algorithm-based comparison is needed instead of utilizing a program-based comparison. PSO was investigated in generating test suite of combinatorial testing [38], and PSO outperformed SA and ACO in terms of the size of test suite and detected faults. Some variants of PSO such as PSO + GA were applied on test case prioritization, but the evaluation of test execution time and APFD through one algorithm is not sufficient to make a reliable decision [39].

Greedy algorithms Please consider rephrasing the following sentence: Greedy algorithms may not be very effective in selecting test cases if this case is considered as a one-way optimization problem ...which the additional and total strategies were created to cope with this problem.]may not be very effective in selecting test cases if this case is considered as a one-way optimization problem [40], which the additional and total strategies were created to cope with this problem. Additional and total strategies aim at augmenting detected faults in terms of unit and total base of test cases. Additional strategy, which handles with test cases as unit base, performs better than total strategy in works associated with these two strategies [41]. Greedy-based test case prioritization algorithm, named two-optimal, was proposed by Li et al. [11], and this algorithm yielded better APFD results than traditional greedy, hill climbing, and GA. However, a new nature-inspired optimization algorithm is needed to decide whether traditional algorithms are still up-to-date.

LBS, which is one of search-based algorithms, was recently tested on C programs [7]. Thus, there is a need for making new experiments to be performed on projects including different languages. Besides, LBS should be compared with similar randomized test case prioritization techniques since it has randomized steps. Further LBS yielded promising results in the comparison including adaptive random test case prioritization and GA. Therefore, these two algorithms were not involved in the experiment of our study.

BAT is a new algorithm relative to the PSO and GA that few works, which are associated with software testing, have studied it. One of them is Srivasta et al.'s work [42] which includes a method combined with BAT algorithm in predicting test effort. Their findings indicate that the bat-inspired method produces close results to the real effort values when it is compared with the methods recorded with PSO and TPA (tests point analysis). Although literature includes some works including the bat-inspired method to the handle with combinatorial testing, there is not any bat-inspired method which has been developed to prioritize test cases. Summary of the related works is presented in Table 1. According to the data of this table, Singh et al.'s work seems to be the first to have applied ACO on test case prioritization. However, the data set used in their experiment is not a large-scale one. In addition, Kaur et al. provide valuable results which do not support their hypothesis with a comprehensive experiment, and the experiment of prioritizing test cases is evaluated using different coverage based analyzes in Zhang et al.'s work. The main constraint of their work is to utilize the total cost for a test suite rather than the individual cost during the execution. Despite Li et al.'s work has a detailed analysis for two-optimal with various evaluation parameters, it needs to be validated with some coverage analysis except for code coverage. Although Jiang et al.'s study has rich content in terms of the comparison of their technique in various programs, it was only tested for code coverage. Panichella et al. also proposed a multi-objective GA that has a great diver-

Table 1 Summary of the works which are most closely related with our work

Method	Experimental design	Approach	References
ACO	Yes	Test suite focused	Singh et al. [34]
PSO	Yes	Test suite focused	Kaur and Bhatt [39]
Total + additional strategies	Yes	Test suite focused	Zhang et al. [41]
Two-optimal (greedy)	Yes	Test suite focused	Li et al. [11]
Adaptive random	Yes	Test suite focused	Jiang and Chan [23]
GA	Yes	Test suite focused	Panichella et al. [9]
BITCP	Yes	Test case focused	Proposed method in the study

sity mechanism. Interestingly, bat-inspired techniques give tips to creating a similar mechanism. Table 1 shows that the common works consider the text suite cost instead of the individual cost of test cases.

3 Bat-inspired test case prioritization

The experiment in this study focuses on developing a nature-inspired prioritization algorithm which is able to overwhelm common methods. BITCP has been inspired by bat algorithm. Consequently, the rules and restrictions of test case prioritization should be considered to adapt the bat algorithm, which originates from the echolocation behavior of bats. First, the test suite is created by selecting a specific number of test cases from test case pools. The reason why the selection is required in this case is that the effort allocated for test processes is planned. The population of bat algorithm is devised as equal as to the number of test cases. The steps of BITCP is seen in Algorithm 1, and two parameters of test cases are adopted to the bat algorithm, test execution time and the number of faults. First, distances $x_1, \ldots, x_n$ are determined as directly proportional to the test case execution times, and the frequencies $f_1, \ldots, f_n$ of bats are inversely proportional to the distances. Wavelengths are computed depending on the velocity and frequency of bats as in Eq. (2). The pulse emission r, which is linked to target proximity, has values between 0 and 1. Similar to the frequency, pulse emission increases as distance reduces. This way, BITCP assigns r to a value between 0 and 1 by looking the distance from the prey. A vector, namely rA, combines the pulse emission r and loudness A. In BITCP, A values are determined by looking the number of faults of related test cases, and a sorting is done in A to make a local solution first of list. This list is then assigned to a new vector as rANew = xStar(rA) afterwards. Then xStar(rA) arranges the population by taking rA in order to change its sequences. To this end, the pulse emission and loudness are considered. In this process, high values have the greatest priority. Once the first ordering change has been completed, the following steps start. The loop has a specific iteration t that computes the different values presented in Eq. (4) are done. While $minMax[0].Item2$ records the index of highest frequency, $minMax[0].Item1$ records the index of lowest frequency, and these values calculate f_i as in Eq. (3). The function $generateBeta()$ generates numbers between 0 and 1, randomly. v_i denotes the velocities of bats, pop includes the first values of the population, and $rANew$ has the values of new population that rely on frequencies and velocities. rand() compares r_j with numbers which are randomly generated between 0 and 1. If the generated number is greater than r_j, x^* is computed by utilizing pulse emission and loudness. Otherwise, r_j is increased and loudness a_j is reduced. In the end of the iteration, sorted test suite is obtained according to the best solution values.

$$\lambda = \frac{v}{f} \tag{2}$$

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta \tag{3}$$

$$v_i = v_i + (x_i - x^*) f_i \tag{4}$$

4 Case study

4.1 Data sets

Five different data sets have been used for evaluating BITCP, and these data sets have been retrieved from GitHub repository.[1] Data sets presented in Table 2 consist entirely of mathematical projects. In this table, Cyclomatic Complexity, $v(g)$, shows the complexity level of a software. If $v(g) > 10$, the related software is defect-prone. Additionally, experimental test pool was created with the PEX tool [43]. PEX is a white-box test case generation tool which was developed for .NET which is one of the widely known web development frameworks. It investigates potential faults by trying all the accessible paths of a program. As such, possible faults are presented with their input values. PEX can conveniently

[1] https://github.com/riyadparvez/data-structures-csharp
https://github.com/open-epicycle/Epicycle.Math-cs
https://github.com/Philip-Trettner/GlmSharp
https://github.com/marcusanth/ProjectEuler
https://github.com/FlorianRappl/YAMP.

Algorithm 1 BITCP algorithm

Determine distances of bats as execution times of test cases $t_1, ..., t_n$
f_i the frequency of related t_i is in inverse relation $f_1, ..., f_n$
Initialize bat population with x_i, v_i
Assign r_i and A_i by looking the frequencies and faultness (rA)
minMax=fMinMax(f)
rANew = xStar(rA);
while t<number of iteration **do**
  for `j=0 to length(`f_n`)` **do**
    $frequency[k] = frequency[minMax[0].Item1] + (frequency[minMax[0].Item2] - frequency[MinMax[0].Item1]) * generateBeta()$
    $v_i = v_i + (pop[j].Item1 - rANew[0].Item1)$
    R=rand()
    if $R >= r_j$ **then**
      rANew = xStar(rANew);
    else
      increase r_j reduce a_j
    end if
  end for
  Return the list including bats ranked by test case execution time and the number of faults
end while

Table 2 Details of experimental data sets

Name	Description	Cyclomatic complexity	Class coupling	LOC	Number of test cases	Percentage of test cases	Language
Data-structures-csharp	A library for all the data structures in C#	150	29	507	110	5	C#
Epicycle.math	Epicycle .NET math library	2450	238	4443	890	14	C#
GlmSharp	Math library for small vectors and matrices	3100	350	5400	910	22	C#
ProjectEuler	Functions solving euler equations	54	29	250	20	4	C#
YAMP	Math parser	5355	366	12,097	2400	55	C#

extract data from mathematical projects and it can also be used as a .NET extension. To provide a compatibility, the projects cloned with Team Foundation Server are selected where the programming language is C#.

Additionally, test pool has been created as directly proportional to the number of test cases. Due to its highest number of test cases, YAMP, which is one of the experimental data set, constitutes 55% of the test pool. The main reason why execution times and faults revealed by test cases have been recorded during the use of PEX is that BITCP adapts them to the bat algorithm. Although experimental data sets are derived from mathematical projects, all their properties are quite different except programming languages. Project web sites are given at the bottom of this page. Consequently, this variety has enriched the experiment. In spite of using open-source data sets in such experiment is feasible, having one type of programming languages threats the results for the comprehensiveness. On the other hand, cloning projects in to Team Foundation Server and working with PEX accelerated the experiment, remarkably.

Experimental data sets generally have the following faults: *nullReferenceException*, *pathBoundsExceeded*, *divideByZero*, *overflowException*, *indexOutOfRangeException*, and *objectNotToReference*. Exceptionally, some faults such as *notEqual* and *Nan* have also been encountered during the experiment. The number of faults changes depending on the size of the project. However, 45, 52, 58, 7, 80 are number of faults for data-structures-csharp, epicycle.math, GlmSharp, ProjectEuler, YAMP, respectively.

4.2 Experimental environment

Experimental environment of this study was Windows 7, 64-bit, Intel Xenon 3.1Ghz server with 16 GB RAM. BITCP has been designed with Visual Studio development environment along with C#. The framework including the code of BITCP, APFD measurement module, and other details can be accessed via https://github.com/marcusanth/Bat-inspired-test-case-prioritization. Additionally, test case and fault information are given to this framework to per-

form an evaluation analysis of prioritization. Consequently, an exterior library was not needed during the design. To validate APFD results of the comparison algorithms, one-way ANOVA and effect size measure have been performed with R package. Since computation complexity is an important target of the prioritization, the experiment also involves measuring prioritization time of comparison algorithms.

4.3 Results

This section answers the research questions of the paper presented in Sect. 1.

RQ1: Test case prioritization techniques developed so far have some shortcomings. First, as detected in the experiment, they have unreasonable prioritization times. Second, they show poor ability to regard individual cost of a test case rather than total cost of a test case. Third, prioritization requires adaptable parameters. However, the parameters of traditional methods are not much suitable to be adapted.

RQ2: APFD values of five algorithms are demonstrated on five different data sets in Fig. 2. These figures show that BITCP, having values which are close to 0.9, is better than the others. LBS produced distinctive APFD results that are close to 0.8. Although greedy, ACO, and PSO yielded close results to each other, greedy has the highest APFD among them.

RQ3: Initially, a test suite is selected from the test pool to compare the performances of prioritization techniques. These techniques are discussed by investigating the changes of APFD in the test suite. The results of APFD obtained from a randomly selected test suite are observed in Figs. 3, 4, 5, 6 and 7. In these figures, x records the ratio of executed test cases and y gives the average percentage of detected faults. According to these results, PSO shows the worst performance, since it could detect roughly 50% of faults when 60% of test cases was executed. ACO, PSO, and Greedy yielded close results (49, 48, 47%, respectively), but the best was ACO among them in terms of APFD. LBS performs better than these three algorithms, because it could detect 60% of faults when 2% of the test suite was executed. Regardless of the size of the data sets, test case-based adaptation in bat algorithm has led to a drastic increment in performance that BITCP detected 89% of faults. In addition, 80% of faults was detected when 2% of test suite has been completed. The most important point affecting the success of BITCP is that the first test cases of test suite find a great amount of faults as demonstrated as in Fig. 7. Such figures are crucial to determine how the algorithm works.

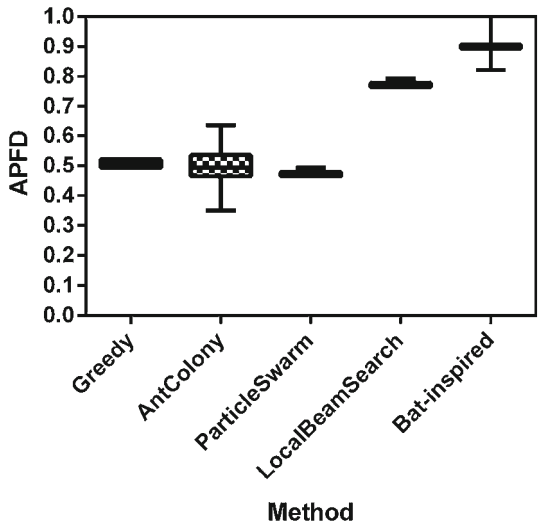


Fig. 2 Mean APFD *box plots* of all algorithms on all data sets

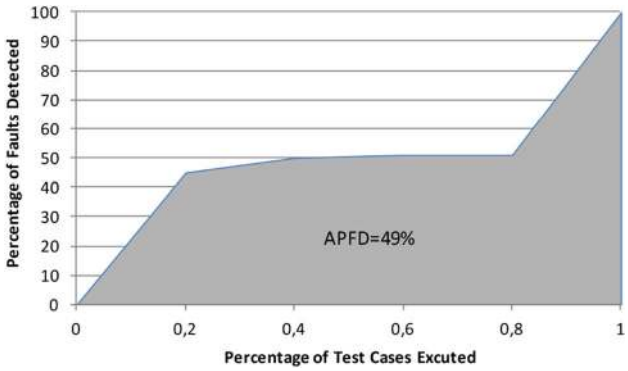


Fig. 3 APFD of ant colony algorithm for an arbitrary test suite

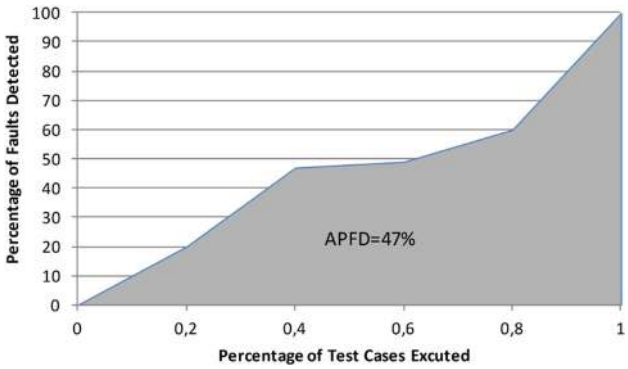


Fig. 4 APFD of particle swarm optimization for an arbitrary test suite

RQ4: With regard to the comparison algorithms, Figs. 8, 9, 10, 11 and 12 demonstrate how APFD changes as the complexity of the codes increases where test cases are generated. In these figures, x records complexity and y shows percentage of faults detected. Additionally, the area between dashed and straight line is the deviation

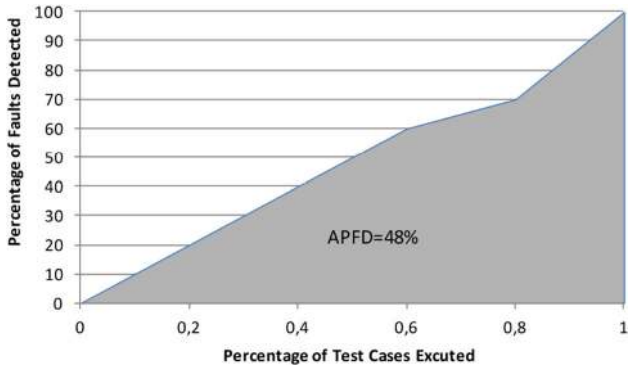


Fig. 5 APFD of greedy algorithm for an arbitrary test suite

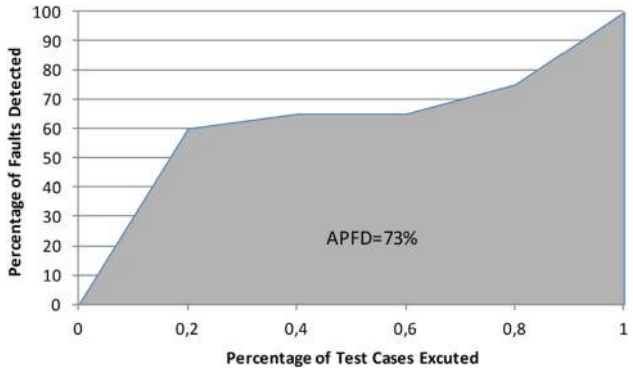


Fig. 6 APFD of local beam search for an arbitrary test suite

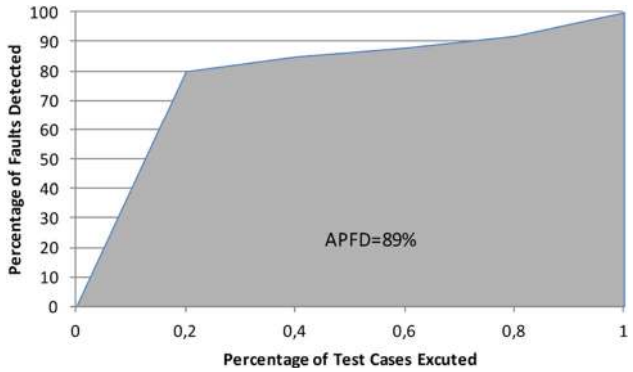


Fig. 7 APFD of BITCP algorithm for an arbitrary test suite

region of the curve. Circles denote the general distribution of the data, and curves of these figures also indicate the results of Spearman analysis. Spearman analysis examines the monotonic relation of two variable groups. According to this analysis, r_s gives the degree of monotonic relation for two variable groups. Having r_s which is close to ± 1 means that there is a strong monotonic relation.

The analysis of one-way ANOVA of APFD yielded from five algorithms is seen in Table 3. Having $p < 0.05$ proves

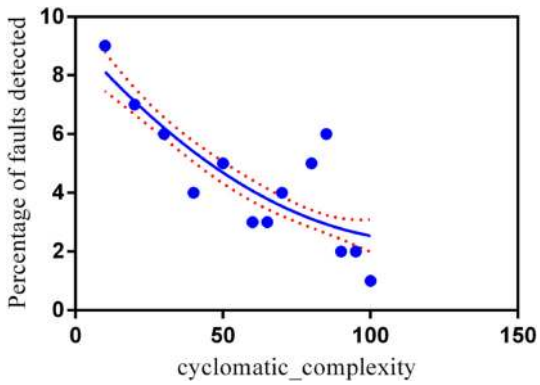


Fig. 8 Complexity-fault analysis of ant colony optimization (Spearman $r_s = -0.7$)

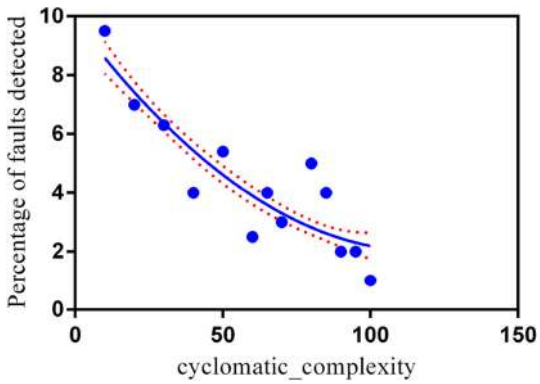


Fig. 9 Complexity-fault analysis of PSO (Spearman $r_s = -0.8$)

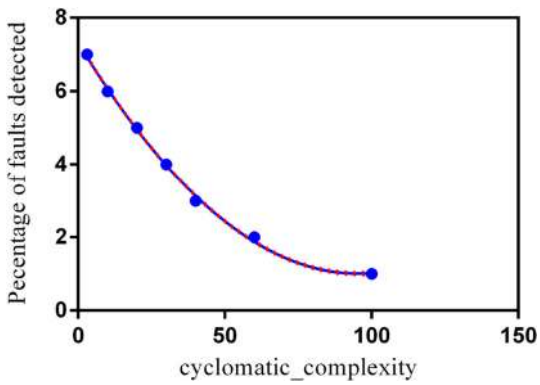


Fig. 10 Complexity-fault analysis of greedy (Spearman $r_s = -0.1$)

that the difference between selected algorithms is significant. However, in such analyses, it is also required to measure the degree of this difference. To measure this degree, effect size measure has been used in the study. If delta estimate of this analysis is close to ± 1, it means that there is a great difference among the groups. The results of effect size measure are presented in Table 4. Delta estimate of BICTP is ± 1 supporting the bias that BITCP has produced worthwhile results.

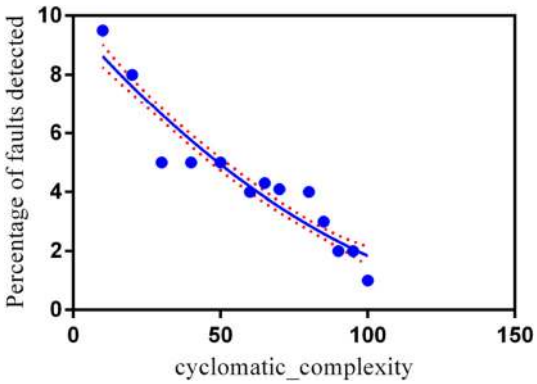


Fig. 11 Complexity-fault analysis of LBS (Spearman $r_s = -0.96$)

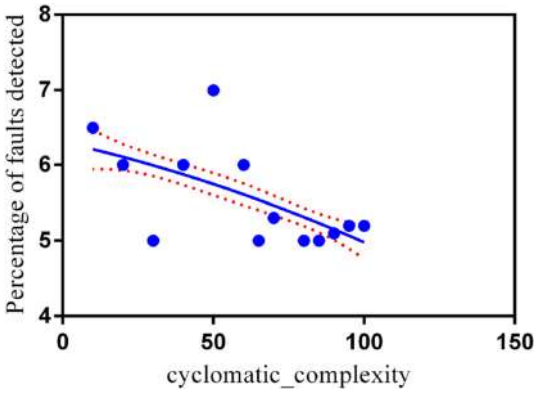


Fig. 12 Complexity-fault analysis of BITCP (Spearman $r_s = -0.48$)

RQ5: There is a monotonic decreasing relation between two variable groups, and r_s obtained from the Spearman is negative. The values generated by the algorithms, which are close to -1, can be considered as they are devoid of resistance against the complexity. Figure 8 shows the relation between the complexities and detected faults of test cases. ACO is seen in Fig. 8. Because $r_s = -0.7$ that there is a strong monotonic decreasing relation between two variables, and the Spearman analysis of PSO ($r_s = -0.8$) is presented in Fig. 9 where the monotonic relation is greater than the result recorded in ACO. Spearman analysis of greedy algorithm is as shown in Fig. 10. Here the deviation in relation curves and the distribution of circles are the lowest relative to the other algorithms, while the degree of monotonic relation is highest ($r_s = -0.1$). This means that greedy is the worst algorithm in terms of showing resistance against the complexity. The second worst one is LBS, whose analysis is presented in Fig. 11. Figure 12 indicates that the decrease in percentage of detected faults is remarkably minimal in BITCP as the complexity increases. The slope of the curve is smaller than the other curves. In this respect, $r_s = -0.48$ of BITCP proves the highest resistance against the complexity. The results of Spearman may not be sufficient to make a bias, because the data sets used in the experiment were retrieved from the projects coded with the same programming language. However, the experiment produced consistent results that may help explain future studies. The results demonstrate how the nature-inspired algorithm can adapt to prioritize test cases.

RQ6: There is not any work using a bat-inspired method to prioritize test cases. So this work has been developed based on first version of bat algorithm. Figures 13 and 14 present prioritization times of BITCP and other results recorded in terms of millisecond. Whereas Fig. 13 includes all the algorithms, Fig. 14 encompasses similar and better ones. The figures show that PSO and ACO have highest prioritization times. On the other hand, BITCP produced the lowest prioritization time among the comparison algorithms. This case may lead to attract interest of researchers. The time used for the comparison was recorded with 20 test cases which were randomly selected before the execution of prioritization algorithms. This means that much more experiments should be performed to validate the bias.

Table 3 One-way ANOVA results of APFD values produced by comparison algorithms

ANOVA	SS	DF	MS	F	P value
Treatment (between columns)	10.03	4	2.508	$F(4, 258) = 1319$	$P < 0.0001$
Residual (within columns)	0.4905	258	0.001901		
Total	10.52	262			

Table 4 Delta estimate values of one-way ANOVA with 95% confidence interval

Greedy-ACO	Greedy-PSO	Greedy-LBS	Greedy-BITCP	ACO-PSO	ACO-LBS
0.2 (small)	0.92 (large)	0.8 (large)	−1 (large)	0.1733333 (small)	0.7 (large)
ACO-BITCP	PSO-LBS	LBS-BITCP	PSO-BITCP		
−1 (large)	0.7 (large)	0.5 (medium)	1 (large)		

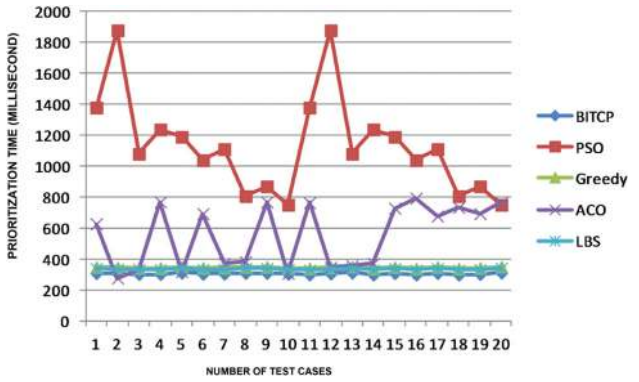


Fig. 13 Prioritization performances of all algorithms in terms of time. The most unstable algorithm is PSO that its performance does not depend the number of iteration in which the iteration range is 8–13. LBS, Greedy, and BITCP seem to have the best results with regard to the stability

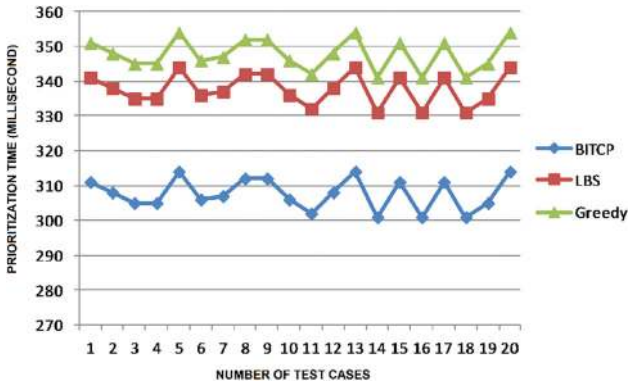


Fig. 14 A closer view of the prioritization time results

5 Threats to validity

This section enhances the depth of the article by discussing the threats of experiment which could occur within the scope of the BITCP algorithm. These threats are explained depending on the sequence of the steps of the experiment.

Selecting GitHub repository for gathering experimental data sets is so important, because GitHub has become immensely popular among the practitioners due to its usage properties. However, selected data sets consist of mainly projects which have mathematical functions. To alleviate this problem, different scale projects are selected. The values of the complexity of these projects are also different. To the best of our knowledge, the experimental data sets have not been used in prioritizing test cases previously. In addition, employing different data sets is prominent to enhance the reliability of investigating the success of the algorithms such as ACO, PSO, and greedy. On the other hand, the experiment does not include widely known data sets used for test case prioritization (grep, flex, and bash [44]). Repeating the same data sets for each experiment creates a vicious cycle. To break this vicious cycle, unconventional data sets have been selected. Therefore, the test pool has been generated with PEX, and its paper [43] is one of the highest cited in this field.

ACO, PSO, greedy, and LBS are selected to compare with BITCP in the experiment. Rather than involving genetic algorithm and random optimization, LBS is selected since search-based algorithms have recently yielded promising APFD results [7]. This bias is not generalizable; consequently, genetic algorithms may produce better results in some cases. Thus, an improved method based on genetic algorithm is better than its basic structure [9].

Test execution times and defect proneness of test cases have been adapted to bat algorithm while developing BITCP. However, parameters used in BITCP can be extended with other metrics including depth of inheritance (DIT), weighted methods per class (WMC), and complexity of the codes. In doing so, the experiment becomes more comprehensive. Data used for the experiment are restricted to software projects coded with C#. Therefore, it is ambiguous regarding how the method behaves with common data sets such as Grep and Flex.

Bat algorithm has been improved by some researchers several times. On the other hand, the method presented in this paper has been improved based on first algorithm proposed by Yang et al. [13]. This creates a threat for the validity of the paper. However, none of them was for prioritizing test cases. Further, using a basic method could facilitate adapting it to perform prioritization. Thus, the way used in this study may be employed for its improved versions.

Last, by using statistical methods such as ANOVA and effect size measure, the difference of comparison algorithms were investigated with respect to the APFD results. The degree of this difference was discussed as well. Hence, the most similar and different algorithms have been detected.

6 Conclusion and future remarks

In this paper, a new nature-inspired test case prioritization algorithm, namely BITCP, is proposed by considering the basic steps of bat algorithm. While devising the algorithm, some properties such as test execution time and code defectiveness are adapted to notions of the algorithm. Promising results have been obtained after applying the method on five different data sets.

BITCP yielded the highest APFD results among the comparison algorithms, and it has the best complexity-percentage of fault detection correlation. LBS is better than ACO, PSO, and greedy as in the work which was first employed in test case prioritization. In addition, LBS cannot cope with the complexity, since it produced worse results ($r_s = -0.96$) than the results obtained with ACO $r_s = -0.7$ and PSO

$r_s = -0.8$. It was detected in the experiment that the most similar algorithms are ACO and PSO in terms of closeness of their results.

BITCP, which was developed by considering the individual cost of test cases, has the potential to be an alternative to create test-suite cost focused methods. To validate the generality of the method, it is necessary to develop new nature-inspired methods for comparing with BITCP.

Predicting faults before the execution of test cases is difficult because this requires some tips to construct test case prioritization methods. The use of different metrics such as WMC and DIT could support the robustness of prioritization techniques. From the point of the perspective of developers, this issue can be addressed by revising the main steps of BITCP.

To date, this work is the first to use bat echolocation behavior in test case prioritization. Therefore, future works may extend the scope of BITCP. To make improvements on the working structure of BICTP regarding the properties of code metrics which give tips to faults, the success of the algorithm will be tested on the data retrieved from the repositories including projects coded with various programming languages.

References

1. Rothermel, G., Untch, R.H., Chengyun, C., Harrold, M.J.: Prioritizing test cases for regression testing. IEEE Trans. Softw. Eng. **27**, 929948 (2001)
2. Leung, H.K.N., White, L.: A cost model to compare regression test strategies. In: Proceedings of Conference on Software Maintenance, pp. 201–208. IEEE Comput. Soc. Press (1991)
3. Hao, D., Zhang, L., Mei, H.: Test-case prioritization: achievements and challenges. Front. Comput. Sci. **10**, 769777 (2016)
4. Huang, R., Chen, J., Towey, D., Chan, A.T.S., Lu, Y.: Aggregate-strength interaction test suite prioritization. J. Syst. Softw. **99**, 3651 (2015)
5. Wong, W.E., Horgan, J.R., London, S., Agrawal, H.: A study of effective regression testing in practice. In: Proceedings The Eighth International Symposium on Software Reliability Engineering, pp. 264–274. IEEE Comput. Soc. (1997)
6. Huang, M., Zhang, C., Liang, X.: Software test cases generation based on improved particle swarm optimization. In: Proceedings of 2nd International Conference on Information Technology and Electronic Commerce, p. 5255. IEEE (2014)
7. Jiang, B., Chan, W.K.: Input-based adaptive randomized test case prioritization: a local beam search approach. J. Syst. Softw. **105**, 91–106 (2015)
8. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science. MHS95, p. 3943 (1995)
9. Panichella, A., Oliveto, R., Penta, M.Di, De Lucia, A.: Improving multi-objective test case selection by injecting diversity in genetic algorithms. IEEE Trans. Softw. Eng. **41**, 358–383 (2015)
10. Edmonds, J.: Matroids and the greedy algorithm. Math. Program. **1**, 127–136 (1971)
11. Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Trans. Softw. Eng. **33**, 225–237 (2007)
12. Solanki, K., Singh, Y., Dalal, S.: Test case prioritization: an approach based on modified ant colony optimization (m-ACO). In: 2015 International Conference on Computer, Communication and Control (IC4), p. 16. IEEE (2015)
13. Yang, X.-S.: A new metaheuristic bat-inspired algorithm. In: Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), p. 6574. (2010)
14. Elbaum, S., Rothermel, G., Kanduri, S., Malishevsky, A.G.: Selecting a cost-effective test case prioritization technique. Softw. Qual. J. **12**, 185210 (2004)
15. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B **26**, 1941 (1996)
16. Parpinelli, R.S., Lopes, H.S., Freitas, A.A.: Data mining with an ant colony optimization algorithm. IEEE Trans. Evol. Comput. **6**, 321332 (2002)
17. Bell, J.E., McMullen, P.R.: Ant colony optimization techniques for the vehicle routing problem. Adv. Eng. Inform. **18**, 4148 (2004)
18. Blum, C., Sampels, M.: An ant colony optimization algorithm for shop scheduling problems. J. Math. Model. Algorithms **3**, 285–308 (2004)
19. Wang, J., Osagie, E., Thulasiraman, P., Thulasiram, R.K.: HOPNET: a hybrid ant colony optimization routing algorithm for mobile ad hoc network. Ad Hoc Netw. **7**, 690–705 (2009)
20. Duran Toksar, M.: Ant colony optimization approach to estimate energy demand of Turkey. Energy Policy **35**, 3984–3990 (2007)
21. Huaizhong, L., Peng Lam, C.: An ant colony optimization approach to test sequence generation for statebased software testin. In: Fifth International Conference on Quality Software (QSIC05), pp. 255–264. IEEE (2005)
22. Yang, Q., Tian, J., Si, W.: An improved particle swarm optimization based on difference equation analysis. J. Differ. Equ. Appl. **23**(1-2), 135–152 (2017)
23. Jiang, B., Zhang, Z., Chan, W.K., Tse, T.H.: Adaptive random test case prioritization. In: 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 233–244. IEEE (2009)
24. Yang, X.S.: Bat algorithm for multi-objective optimisation. Int. J. Bio-Inspired Comput. **3**(5), 267–274 (2011)
25. Wang, G.G., Chu, H.E., Mirjalili, S.: Three-dimensional path planning for UCAV using an improved bat algorithm. Aerosp. Sci. Technol. **49**, 231–238 (2016)
26. Lemma, T.A., Hashim, F.B.M.: Use of fuzzy systems and bat algorithm for energy modeling in a gas turbine generator. In: 2011 IEEE Colloquium on Humanities, Science and Engineering, pp. 305–310. IEEE (2011)
27. Zhou, Y., Luo, Q., Xie, J., Zheng, H.: A Hybrid Bat Algorithm with Path Relinking for the Capacitated Vehicle Routing Problem. In: Yang XS., Bekdaş G., Nigdeli S. (eds) Metaheuristics and Optimization in Civil Engineering, pp. 255–276. Springer, Cham (2016)

28. Bora, T.C., dos Coelho, L.S., Lebensztajn, L.: Bat-inspired optimization approach for the brushless DC wheel motor problem. IEEE Trans. Magn. **48**, 947–950 (2012)
29. Hasanebi, O., Teke, T., Pekcan, O.: A bat-inspired algorithm for structural optimization. Comput. Struct. **128**, 7790 (2013)
30. Ayari, K., Bouktif, S., Antoniol, G.: Automatic mutation test input data generation via ant colony. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation—GECCO 07, p. 1074. ACM Press, New York (2007)
31. Biswas, S., Kaiser, M.S., Mamun, S.A.: Applying Ant Colony Optimization in software testing to generate prioritized optimal path and test data. In: 2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT), p. 16. IEEE (2015)
32. Mao, C., Xiao, L., Yu, X., Chen, J.: Adapting ant colony optimization to generate test data for software structural testing. Swarm Evol. Comput. **20**, 23–36 (2015)
33. Ghiduk, A.S.: A new software data-flow testing approach via ant colony algorithms. Univ. J. Comput. Sci. Eng. Technol. **1**, 6472 (2010)
34. Singh, Y., Kaur, A., Suri, B.: Test case prioritization using ant colony optimization. ACM SIGSOFT Softw. Eng. Notes **35**, 1 (2010)
35. Henard, C., Papadakis, M., Harman, M., Jia, Y., Le Traon, Y.: Comparing white-box and black-box test prioritization. In: Proceedings of the 38th International Conference on Software Engineering—ICSE 16, pp. 523–534. ACM Press, New York (2016)
36. Windisch, A., Wappler, S., Wegener, J.: Applying particle swarm optimization to software testing. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation—GECCO 07, p. 1121. ACM Press, New York (2007)
37. Rauf, A., Aleisa, A.E.: PSO based automated test coverage analysis of event driven systems. Intell. Autom. Soft Comput. **21**, 491–502 (2015)
38. Ahmed, B.S., Zamli, K.Z.: A variable strength interaction test suites generation strategy using particle swarm optimization. J. Syst. Softw. **84**, 2171–2185 (2011)
39. Kaur, A., Bhatt, D.: Hybrid particle swarm optimization for regression testing. Int. J. Comput. Sci. Eng. **3**, 1815–1824 (2011)
40. Yoo, S., Harman, M.: Pareto efficient multi-objective test case selection. In: Proceedings of the 2007 international symposium on Software testing and analysis—ISSTA 07, p. 140. ACM Press, New York (2007)
41. Zhang, L., Hao, D., Zhang, L., Rothermel, G., Mei, H.: Bridging the gap between the total and additional test-case prioritization strategies. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 192–201. IEEE (2013)
42. Srivastava, P.R., Bidwai, A., Khan, A., Rathore, K., Sharma, R., Yang, X.S.: An empirical study of test effort estimation based on bat algorithm. Int. J. Bio Inspired Comput. **6**, 57 (2014)
43. Tillmann, N., De Halleux, J.: PexWhite box test generation for .NET. In: International conference on tests and proofs, pp. 134–153 (2008)
44. Do, H., Elbaum, S., Rothermel, G.: Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact. Empir. Softw. Eng. **10**, 405–435 (2005)