Search-Based Regression Testing Optimization

Abeer Hamdy Dr.
The British University in Egypt, abeer.hamdy@bue.edu.eg

Nagwa R. Fisal
Suez Canal University

Essam A. Rashed
Suez Canal University

International Journal of Open Source Software and Processes
Volume 12 • Issue 2 • April-June 2021

Search-Based Regression Testing Optimization

Nagwa R. Fisal, Suez Canal University, Egypt
Abeer Hamdy, British University in Egypt, Egypt
Essam A. Rashed, Suez Canal University, Egypt

ABSTRACT

Regression testing is one of the essential activities during the maintenance phase of software projects. It is executed to ensure the validity of an altered software. However, as the software evolves, regression testing becomes prohibitively expensive. In order to reduce the cost of regression testing, it is mandatory to reduce the size of the test suite by selecting the most representative test cases that do not compromise the effectiveness of the regression testing in terms of fault-detection capability. This problem is known as test suite reduction (TSR) problem, and it is known to be an NP-complete. The paper proposes a multi-objective adapted binary bat algorithm (ABBA) to solve the TSR problem. The original binary bat (OBBA) algorithm was adapted to enhance its exploration capabilities during the search for a Pareto-optimal surface. The effectiveness of the ABBA was evaluated using six Java programs with different sizes. Experimental results showed that for the same fault discovery rate, the ABBA is capable of reducing the test suite size more than the OBBA and the BPSO.

KEYWORDS

Binary Bat Algorithm, Multi-Objective Optimization, Mutation Testing, Regression Testing, Search-Based Software Engineering, Software Testing, Test Suite Reduction

INTRODUCTION

As software testing is known to be an expensive process, open-source software is usually released with several bugs; e.g., at the early releases of Mozilla and Eclipse, about 170 and 120 bugs respectively were reported daily (Abeer Hamdy & El-Laithy, 2020; Abeer Hamdy & Ellaithy, 2020; Abeer Hamdy & Ezzat, 2020). It is essential to design a cost-effective test plan that detects as many defects as possible before the release of the open-source software to ensure the quality of the delivered software. Especially, during the maintenance phase, enhancements and modifications are made to the software, which necessitates the development and execution of new test cases to test the modifications; in addition to the re-execution of the earlier test cases, to test the software stability after enhancements (Catal & Mishra, 2013). Testing the behavior of the whole system under test (SUT) before release and after each modification is called regression testing (Leung & White, 1989; Rosero, Gómez, & Rodríguez, 2016). The cost of regression testing increases over time due to the increase in the test suite size. So, it is important to find the smallest representative subset of the test suite without compromising the fault-detection capability of the original test suite (Gotlieb & Marijan, 2014; Nadeem & Awais, 2006). This problem is known as test suite reduction problem (TSR). One way to assess the capabilities of the reduced test suite, in discovering bugs, is through the utilization of a fault-based testing technique called mutation testing. Mutation testing calculates a score for the

DOI: 10.4018/IJOSSP.2021040101

test suite which indicates its capabilities on discovering bugs in the SUT (Jia & Harman, 2010). The TSR problem is known to be a combinational optimization problem that can be described as a set covering problem which is known to be NP-complete (Gary & Johnson, 1979). In practice, there is no efficient solution for NP-complete problems. However, suboptimal solutions could be found using search-based optimization (SBO) algorithms (Chen & Lau, 1998). Bat algorithm (BA) is a recent and efficient SBO algorithm, which mimics the echolocation behavior of bats to find a global optimal solution (Yang, 2010). The performance of the BA was reported in the literature to be superior to other SBO algorithms such as the particle swarm optimization (PSO) (Eberhart & Kennedy, 1995; A Hamdy & Mohamed, 2019) and Genetic algorithms (GA) (Abeer Hamdy, 2014), over the majority of benchmark functions and real applications.

Aims and Contributions

The aim of this paper is to reduce the cost of the regression testing, through reducing the test suite size using the BA. Our contributions to accomplish this aim are summarized as follows:

- Proposing modifications to the Original Binary Bat Algorithm (OBBA) (Mirjalili, Mirjalili, & Yang, 2014) to enhance its exploration and exploitation capabilities; so to reduce its occasionally failure to converge to global optimum solutions.
- Formulating the TSR problem in terms of two objectives which are: the cost of the reduced test suite and the mutation score. Then, applying the variable weighted sum method (Yang, 2011) to guide the Adapted binary BA (ABBA) search for the non-dominated solutions that form a Pareto-optimal surface.
- Evaluating the performance of the ABBA against each of the OBBA and the Binary Particle Swarm Optimization BPSO (Bansal et al., 2011) in solving the multi-objectives TSR problem over six Java programs of different test suite sizes and different number of mutants.

The rest of the paper is organized as follows: Section 2 introduces some important preliminaries for this work. Section 3 discusses the previous studies that tackled the TSR problem. Section 4 presents the multi-objective adapted binary bat algorithm for solving the TSR problem. Section 5 discusses the experiments and results. Finally, Section 6 concludes the paper and introduces possible extensions to this work.

BACKGROUND

Test Suite Reduction Problem

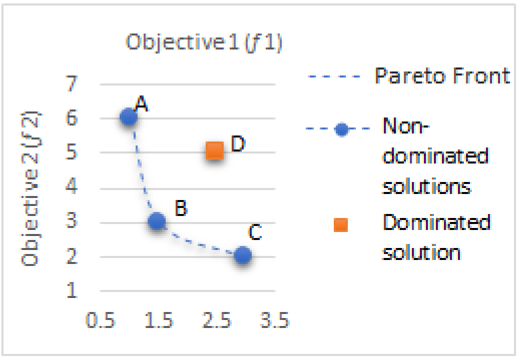
Given: A test suite TS which includes d test cases, and a set of n mutants $\left\{mu_1,\ldots,mu_n\right\}$, that should be killed to provide an adequate testing of the SUT. Each test case tc_j can kill one or more mutants mu_i.

Problem: Find an adequate subset $TS' \subseteq TS$ that can kill as many as possible number of mutants and includes as few as possible number of test cases. These two objectives are contradictory; this is the reason we formulated the TSR problem as a multi-objective optimization problem.

Pareto Optimal Concepts

In multi-objective optimization problems, there is no single solution but a set of multiple trade-off solutions (Ngatchou, Zarei, & El-Sharkawi, 2005). The vector of decision variables that optimizes the considered objective functions and satisfy the problem constrains is called a Pareto front. Thus, the Pareto front is a set of Pareto solutions which are not dominated by any other solution as shown in Figure 1. A solution $x = \left[x_1, x_2, \ldots, x_n\right]$ is said to dominate a solution $y = \left[y_1, y_2, \ldots, y_n\right]$, if and only if y is not better than x for any objective $i = 1, 2, \ldots, n$, and there exist at least one objective x_i in x which is better than its corresponding objective y_i in y. A better solution means it has a minimum value when the problem is minimization and on the contrary in maximization problem. On the contrary, two solutions are said to be non-dominated when none of them dominates the other. Figure 1 depicts the difference between dominated and non-dominated solutions and represents the Pareto front. In the figure, the objective functions $f1$ and $f2$ are to be minimized. It is obvious that solution A dominates solution D because $f1\left(A\right) < f1\left(D\right)$ and $f2\left(A\right) < f2\left(D\right)$. Moreover the solutions A, B and C are non-dominated solutions because none of them is better than the others in both objectives; as A is the best for objective $f1$, whereas C is the best for $f2$ objective, and B is better than A for objective $f2$ and better than C or the objective $f1$. The set of high-quality solutions of the multi-objective optimization problem is called the Pareto optimal set, and its representation in the objective space is the Pareto front. This set satisfy two properties: (i) any solution found is dominated by at least one solution in the Pareto set, and (ii) every two solutions in the set are non-dominated to each other. Generating the Pareto front assists the decision maker to take an informed decision through providing him with a wide range of solutions, Pareto set, that are optimum from different point of view.

Figure 1. A sample representation of dominated, non-dominated solutions and a Pareto front



Particle Swarm Optimization Algorithm

PSO is one of the swarm intelligence algorithms that was proposed in 1995 by Eberhart et al. (Eberhart & Kennedy, 1995). It is inspired by the social behavior of bird flocks that collaboratively work together to reach the point that has the most resources. The whole flock is called swarm, while each bird in the swarm is called a particle. Each particle is a solution in the search space and has three attributes namely, velocity, position, and best explored position by the particle. The velocity attribute guides the particle motion to its next position. The particle's position is updated every iteration according to: (i) the particle's current velocity value, (ii) the global best position that was found by the swarm, and (iii) the best explored position found by the particle. The PSO algorithm iterates for a predetermined number of iterations or until a minimum error value is achieved. PSO was originally developed for real valued spaces, but in 1997, Kennedy et al. (Kennedy & Eberhart, 1997) introduced a binary version of PSO (BPSO) for discrete optimization problems. In the BPSO particle's position is represented using a binary value, 0 or 1. The velocity of a particle is defined as the probability that a particle changes its position.

Bat Algorithm

Bat algorithm (BA) is a one of the recent metaheuristic swarm intelligence optimization algorithms which is proposed by Yang (Yang, 2010). BA was inspired by the behavior of the micro-bats. A bat b_i flies randomly with velocity V_i at position X_i with a frequency F_i, varying wavelength $\lambda_i = V_i F_i$ and loudness A_i to search for a food/prey in a d dimensional search space. The BA starts with generating randomly the initial population of bats. The values of the parameters of each bat b_i are updated over the iterations according to Eq. (1) - Eq. (3):

$$V_i\left(t+1\right) = V_i\left(t\right) + \left(X_i\left(t\right) - Gbest\right) F_i \tag{1}$$

$$X_i\left(t+1\right) = X_i\left(t\right) + V_i\left(t+1\right) \tag{2}$$

$$F_i = F_{min} + (F_{max} - F_{min})\beta \tag{3}$$

where, F_i is the i th bat frequency value, F_{min} and F_{max} are the minimum and maximum frequency values respectively, β is a random number of a uniform distribution, ***Gbest*** is the current global best location (solution). The bats perform a random walk procedure which is defined by Eq. (4) for exploring the space:

$$X_{new} = X_{old} + \varepsilon A^t \tag{4}$$

where, ε is a random number in the range $\left[-1,1\right]$. A^t is the average loudness of all the bats at time t. It could be stated that the BA is a balanced combination of the PSO and the intensive local search algorithms. The balance between these two techniques is controlled by both loudness $\left(A\right)$ and the pulse emission rate $\left(r\right)$ which are updated according to Eq. (5) and Eq. (6):

$$A_i\left(t+1\right) = \alpha A_i\left(t\right) \tag{5}$$

$$r_i\left(t+1\right) = r_i\left(0\right)\left[1 - exp\left(-\gamma t\right)\right] \tag{6}$$

where, α and γ are constants; α .is analogous to the cooling factor in the simulated annealing (SA).

Mirjalili *et al.* (Mirjalili et al., 2014) proposed the OBBA to solve optimization problems in the binary search space. In the OBBA the bat's position is changed from one to zero or vice versa based on the probability of the bat's velocity according to Eq. (7) and Eq. (8):

$$v\left(V_i^k\left(t+1\right)\right)=\left|\frac{2}{\pi}arctan\left(\frac{\pi}{2}V_i^k\left(t+1\right)\right)\right| \quad (7)$$

$$x_i^k\left(t+1\right)=\begin{cases}(x_i^k\left(t\right))^{-1} & if\ rand < v\left(V_i^k\left(t+1\right)\right)\\ x_i^k\left(t\right) & if\ rand \geq v\left(V_i^k\left(t+1\right)\right)\end{cases} \quad (8)$$

where $x_i^k\left(t\right)$ and $V_i^k\left(t\right)$ is the position and velocity of i th particle at iteration t in k th dimension, and $(x_i^k\left(t\right))^{-1}$ is the complement of $x_i^k\left(t\right)$.

LITERATURE REVIEW

Researchers proposed a significant number of approaches to reduce the size of the test suite, with no loss of quality (Bhatia, 2020; Mohapatra, Mishra, & Prasad, 2020; Xue & Li, 2020). TSR approaches can be categorized into three main categories: (i) Greedy based (Assi, Masri, & Trad, 2018; Lin, Tang, Wang, & Kapfhammer, 2017), (ii) Clustering based (Coviello et al., 2018) and (iii) Search based (Geng, Li, Zhao, & Guo, 2016; Xue & Li, 2020).

Greedy based approaches utilize one of the greedy algorithms to determine the reduced test suite based on the current best strategy. Over each iteration greedy algorithm adds to the reduced test suite the test case that has the highest coverage (local optimal solution); it stops when the desired percentage of coverage is reached. Greedy approaches are able to find a minimal-cardinality test suite but with some loss in fault-detection capability. On the other hand, clustering based approaches utilize one or more of the clustering algorithms to group similar test cases together according to a predefined similarity measure. Then a sampling mechanism is applied to select one or more test cases from each cluster to be included in the reduced test suite, while, the rest of the test cases are discarded. Finally, search-based approaches employ heuristic algorithms to search for a near-optimal solution. A fitness function based on cost and/or effectiveness measures is used to guide the search. Search-based approaches could be classified according to the search algorithm utilized into single objective (Mohanty, Mohapatra, & Meko, 2020; Sun & Wang, 2010) or multi-objective optimization (Coviello et al., 2018; Gupta, Sharma, Pachariya, & Sciences, 2020; Khan, Lee, Javaid, & Abdul, 2018; Wang, Ali, & Gotlieb, 2015; Wei et al., 2017; Yoo & Harman, 2007).

The multi-objective TSR optimization aims at establishing an acceptable tradeoff between the mentioned cost and effectiveness measures. However, according to a survey study conducted by Khan *et al.* (Khan et al., 2018) the majority of the previous TSR search based approaches (79%) are single-objective optimization. The work of Yoo and Harman (Yoo & Harman, 2007) is considered the first work that applies multi-objective optimization for test suite minimization. They utilized two versions of the non-dominated sorting genetic algorithm II (NSGA-II) and implemented a weighted sum greedy technique. They proved experimentally that the NSGAII is superior to the greedy approaches in case of multi-objective optimization. Geng *et al.* (Geng et al., 2016) utilized NSGA-II heuristic algorithm to minimize test suite and achieve high fault detection rate and fault localization accuracy. Their experiments show promising results in the reduction percentage of the test suite with high fault detection rate and fault localization accuracy. Wang *et al.* (Wang et al., 2015) aimed at reducing the test suite of product lines software. They proposed three weight-based genetic algorithms, where the different objectives are weighted summed together to form a single objective fitness function that guides the GA during the search for a pareto front. They compared the performance of their proposed weight-based genetic algorithms to the performance of seven other popular multi-objective search algorithms including: NSGA-II, strength pareto evolutionary algorithms (SPEA) and Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO). Their experimental results showed that the Random-Weight genetic algorithm is the superior algorithm. Wei *et al.* (Wei et al., 2017) proposed a many-objective optimization approach based on mutation testing for test suite reduction. They used the mutation score as a major objective tighter with cost and some standard well-known coverage criteria such as: statement coverage, branch coverage, and Modified Condition/Decision Coverage. They compared among six evolutionary multi-objective optimization algorithms including NSGAII and several variants of the multi-objective's decomposition-based evolutionary algorithm (MOEA/D). Their experimental results showed the superiority of the NSGAII over small programs but over large programs (space) the MOEA/D was superior. They also showed that the using the "mutation score" in the fitness function improved the performance. Gupta *et al.* (Gupta et al., 2020) proposed a code and mutant coverage based multi-objective approach for test suite reduction using NSGA-II algorithm. Their experiments showed promising result in the reduction percentage of the test suite and their fault detection capability. Agrawal *et al.* (Agrawal, Choudhary, Kaur, & Pandey, 2019) proposed fault coverage-based test suite optimization method based on Harrolds–Gupta–Soffa (HGS). Their performance measures are fault coverage, execution time and reduced optimized test suite size. Experimental results showed that their utilized technique outperforms the Greedy method, Additional Greedy, HGS, and Enhanced HGS.

The approach proposed in this paper is a search-based approach. The authors adapted one of the recent heuristic algorithms OBBA which proved its superiority over other evolutionary algorithms in different contexts (Chawla & Duhan, 2015; Jayabarathi, Raghunathan, & Gandomi, 2018). However, the OBBA occasionally fails to discover the global best solution for some multi modal functions; in addition, the OBBA is used for solving single objective optimization problems. So, we proposed modifications to the OBBA and utilized it to minimize the test suite size, without loss in its fault detection capability. In this paper the fault detection capability of the test suite was assessed using the mutation testing, as mutation testing has been recommended by a number of researchers as an effective method for assessing the quality of a test suite. Walsh (Walsh, 1985) empirically found that mutation testing is more effective than statement and branch coverage. Additionally, Frankl *et al.* (Frankl, Weiss, & Hu, 1997) and Offutt *et al.* (Offutt, Pan, Tewary, & Zhang, 1996) confirmed that finding faults using mutation testing is more powerful than using data flow-based testing. Finally, the recent empirical study of Andrews *et al.* (Andrews, Briand, Labiche, & Namin, 2006) reported that the mutation analysis is potentially useful in evaluating and comparing test suites.

PROPOSED MULTI-OBJECTIVE ABBA FOR TSR

TSR Solution Encoding

Consider that each bat b_i has a position vector X_i that represents a solution to the TSR problem, i.e., each X_i represents a reduced test suite. X_i is encoded as a binary vector $X_i = (x_{i1}, x_{i2}, ...,$ x_{id}, where, d is the size of the original test suite (the total number of test cases), each bit x_{ij} corresponds to a test case tc_j, the bit value is equal to "1" or "0". The value of "1"/"0" means that tc_j is included/excluded in the test suite, respectively.

Each tc_i is represented using a binary vector of size n, where n is the total number of mutants of a SUT, $tc_i = \left(mu_{i1}, mu_{i2}, \ldots, mu_{in}\right)$, where a bit mu_{ij} corresponds to the mutant number j. The value of the bit mu_{ij} is set to equal "1"/"0" if tc_i kill/do not kill the mutant number j.

Adapted Binary Bat Algorithm (ABBA)

Generally, the characteristics of any search algorithm including the BA is affected by two crucial capabilities which are: 1) exploration and 2) exploitation. Exploration is the ability of an algorithm to find promising solutions by searching various unknown regions. While, exploitation leads the algorithm to the best solution among the discovered ones. Exploration capability can get the algorithm away from a local optimum it gets stuck in; while exploitation capability increases the convergence speed of an algorithm. To enhance the performance of any search algorithm including the BA, it is important to keep the balance between the global and local search, such that the global search is amplified at the early iterations. While the local search is amplified at the late iterations so the algorithm converges to the global optimum.

The update formula of the bat velocities, $V_i\left(t+1\right)$, includes two components. The first component is the previous velocity of the bat, $V_i\left(t\right)$, which is responsible for the global search (exploration). As, $V_i\left(t\right)$ directs the bat to keep its velocity and direction, thus it overflows the search space. While the second component, $(X_i\left(t\right)-Gbest)F_i$, is responsible for the local search (exploitation). As it directs all the bats to a region near to the best-found global solution (*Gbest*). So, the following modifications were proposed to the $V_i\left(t+1\right)$ formula: Firstly, multiplying the term $V_i\left(t\right)$ by an inertia weight factor $"w"$, which is given by Eq. (9). The value of w will decrease linearly over iterations. The inertia weight was recommended by a number of previous studies that aimed at enhancing each of the BA [11] and the PSO (Bansal et al., 2011; Xin, Chen, & Hai, 2009):

$$w = w_{\max} - \left(w_{\max} - w_{\min}\right)\left(\frac{iter}{iter_{\max}}\right) \tag{9}$$

where $w_{\max}$ and $w_{\min}$ are pre-determined constant $iter$ is the current iteration number, $iter_{\max}$ is the maximum number of iterations.

The other suggested modification is to assume that each bat emits two frequencies instead of one before the bat decides on its moving direction. The first frequency is directed towards the location of the $Gbest$, while the second frequency is directed towards a randomly selected best solution discovered over the previous iterations $Rbest$. Any of the previously discovered best solutions could be a candidate for a global optimum solution. This way each bat benefits from the experiences of the other bats. Consequently, Eq. (1) is amended as follows:

$$V_i\left(t+1\right) = w\left(V_i\left(t\right)\right) + \left(X_i\left(t\right) - Gbest\right)F_{1i} + \left(X_i\left(t\right) - Rbest\right)F_{2i} \tag{10}$$

$$F_{1i} = F_i\delta, F_{2i} = F_i\left(1-\delta\right) \tag{11}$$

$$\delta = \left(\delta_{\min}\right)^{1-\frac{iter}{iter_{\max}}} \tag{12}$$

where $Rbest$ is a randomly selected best solution other than the $Gbest$. δ increases non-linearly from δ_{min} to 1 which increases the impact of the location of the $Gbest$ over the iterations, so the bats converge to the $Gbest$.

Multi-Objective Fitness Function

The formulation of the fitness function is crucial for the evolutionary algorithms; as it is used for assessing the discovered solutions and consequently guides the search process. In this paper, the aim of the ABBA is to select from a given test suite the smallest set of test cases that could kill the largest number of mutants; which are two contradictory objectives. To formulate the fitness function, we used the weighted sum method which is simple and traditional method for multi-objective optimization. it produces a pareto optimal set of solutions by changing the weights among the objectives functions. Yang (Yang, 2011) showed experimentally that the weighted sum method for combining the multi-objectives into a single-objective is very efficient even with highly nonlinear problems, complex constraints and diverse Pareto optimal sets. Moreover, Wang *et al.* (Wang et al., 2015) showed that the weighted- based GA (multi-objective GA based on weighted sum method) is superior to some popular multi-objective algorithms, *e.g.*, NSGAII and SPEA. The fitness function used in this work is defined by Eq. (13) and Eq. (14), the best solution is the one that maximizes the $fitness$:

$$fitness = weight_1 * \frac{\left|Killed\ mu\right|}{\left|mu\right|} + weight_2 * \frac{\left|tc\right|}{\left|Reduced\ TS\right|} \tag{13}$$

$$weight_1 = 1 + \left(weight_{init} - 1\right)\left(\frac{iter_{\max} - iter}{iter_{\max}}\right)^n, weight_2 = 1 - weight_1 \tag{14}$$

where $\left|Killed\ mu\right|$ is the number of killed mutants by the reduced size test suite, $\left|mu\right|$ is the total number of mutants, $\left|tc\right|$ is the total number of test cases, $\left|Reduced\ TS\right|$ is the size of the reduced test suite, $weight_1, weight_2$ are the weights used to find the pareto optimal set of solutions and they are defined using Eq. (14). Where, $weight_{init}$ denotes the initial value of $weight_1$, $iter$ is the current iteration number, and *n* is a modulation index. With the increase in the iteration the value of $weight_1$ increases from $weight_{init}$ to 1, whereas $weight_2$ decrease from (1 - $weight_{init}$) to 0. The values of $weight_1\ and\ weight_2$ determine the importance of each objective to the fitness function. Large values of $weight_1$, the solutions with large number of killed mutants are more preferred. In contrary with large values of $weight_2$, the solutions with minimum number of test cases are more preferred. The different values of $weight_1, weight_2$ produce different non-dominated solutions with sufficient diversity; so the Pareto front can be approximated correctly. Algorithm 1 shows the basic steps of the multi-objective ABBA.

EXPERIMENTS AND RESULTS

Research Questions

The experiments were designed to answer the following research questions:

RQ1: Does the performance of the ABBA surpasses the performance of the OBBA and the BPSO in solving the test suite reduction problem?
RQ2: Is the ABBA scalable, i.e. How does the increase in the test suite size and the number of mutants impact the performance of the ABBA?

Data Sets

We used a free available dataset [(1)], which was used in previous studies (Offutt, Lee, et al., 1996; Offutt & Lee, 1994; Pargas, Harrold, & Peck, 1999; Polo, Piattini, & García-Rodríguez, 2009; Usaola, Mateo, & Lamancha, 2012). The dataset includes six Java programs; their characteristics are listed in Table 1, which are the software size in terms of the Line of Code (LOC), number of first-order mutants ($\left|mu\right|$) generated by the MuJava tool [(2)], and the test suite size ($\left|tc\right|$) which are automatically generated using testooj tool [(3)]. The programs are arranged in Table 1 according to the test suite size from the smallest to the largest. It should be noted that the value of the ($\left|tc\right|$) is not proportional to the value of the ($\left|mu\right|$) or the (LOC).

Performance Metrics

Three metrics were used in the performance evaluation which are: (1) the fitness function values $fitness$, (2) Speed of convergence to the best solution measured in terms of the number of iterations i and (3) the fault detection rate (FDR) of the test suite calculated using equation (15) as follows:

$$FDR = \frac{Number\ \ of\ \ killed\ \ mutants}{Total\ \ number\ \ of\ \ mutants} \tag{15}$$

The higher the value of the ($fitness$), the better is the discovered solution. The higher the value of the FDR, the more capable is the test suite on discovering the faults. The lower the value of the (i), indicates that the algorithm converges faster to the best solution.

As the evolutionary algorithms are stochastic, N independent runs were performed over each dataset (in this paper N was set to equal 10). Then the mean and standard deviation (SD) of each metric was calculated at the end of the runs. The mean and SD of f are calculated by Eqs. (16), (17) respectively:

$$mean = \left(\sum_{k=1}^{N} f_k\right) / N \tag{16}$$

$$SD = \sqrt{\frac{\sum_{k=1}^{N} \left(f_k - mean\right)^2}{N}} \tag{17}$$

where, N is the number of runs, f_k is the fitness of the best solution discovered during the run number k.

The smaller the value of the SD, the more robust is the algorithm, as small SD values indicate that the algorithm can find acceptable solutions in the different runs, with small discrepancy.

Algorithm 1. Multi-objective adapted binary bat algorithm (ABBA)

```
Define the objective functions $f_1(X), \dots, f_k(X), X = (x_1, x_2, \dots, x_d)^T$
for j = 1 to Z (Z is the number of points on Pareto fronts)
    Generate $K$ weights, $weight_k \geq 0, \sum_{k=1}^{K} weight_k = 1$
    $fitness = \sum_{k=1}^{K} weight_k f_k$
    Initialize the Bat population: position $X_i = rand(0\ or\ 1)$ and
    velocities $V_i$ $(i = 1,2, \dots, N)$
    Specify the pulse frequencies $F_i$ at $X_i$
    Initialize loudness $A_i$ and pulse emission rate $r_i$
    Find the current $Gbest$ and $Rbest$
    $while(\ t \leq Max.\#\ of\ iterations)$
        Adjust $F_i$ and update $V_i$ $((i = 1,2, \dots, N))$ using
        Eq. (3) and Eq. (10)
        Calculate $v(V_i)$ using Eq. (7)
        Update $X_i$ using Eq. (8)
        $if\ (\ rand > r_i)$
            Select a solution among the best
            solutions randomly and generate local
            solution around it.
        $end\ if$
        Generate a new solution by flying randomly
        $if$ $(rand < A_i\ \&\ fitness(X_i) <$
        $fitness(Gbest)\ )$
            Accept the new solutions
            Reduce $A_i$ and Increase $r_i$
        $end\ if$
        Rank the bats and find the current $Gbest$ and
        $Rbest$
        $t = t + 1$
    $end\ while$
    Record $Gbest$ as a non-dominated solution
End for
Post-process results
```

Parameter Setting

The selection of the parameter values of the evolutionary algorithms has an impact on their performance. The authors experimented with the most recommended values for the parameters in literature (Carvalho, da Rocha, Silva, da Fonseca Vieira, & Xavier, 2017). Table 2 lists the parameter values that achieved the best performance. Table 3 lists the values of weight1 and weight2 which were used to calculate 10 non-dominated solutions on the Pareto surface.

Results

The fitness mean and SD values of the non-dominated solutions found by each of the ABBA, OBBA and BPSO, and their corresponding convergence speeds, measured in terms of the mean and SD of

Table 1. Experimental Software Under Test (SUT)

SUT	LOC	$\lvert mu \rvert$	$\lvert tc \rvert$
MBisectOk	31	44	25
MFourBall	47	168	96
MMidOK	59	138	125
MFindOk	79	179	135
MTringle	61	239	216
MBubCorrect	54	70	256

Table 2. Parameter settings for ABBA, OBBA, and BPSO

Parameter	Value		
	ABBA	OBBA	IBPSO
Population size	40	40	40
Max iteration	100	100	100
Stopping criterion	Max iteration	Max iteration	Max iteration
$F_{\min}$	0	0	-
$F_{\max}$	2	2	-
A	0.9	0.9	-
r	0.1	0.1	-
ε	[-1,1]	-	-
α	0.9	0.9	-
γ	0.9	0.9	-
$w_{\max}$, $w_{\min}$	0.8, 0.4	-	-
Modulation index, n	2	-	-
δ_{init}	0.6	-	-
C_1, C_2	-	-	1.5, 1.2
W	-	-	2
Max velocity	-	-	6

the number of iterations required to discover the best solutions are shown in Table 4. Table 5 lists the sizes of the reduced test suites, their corresponding number of killed mutants and fault detection rates.

Answer to RQ1

It could be observed from Table 4, that the ABBA algorithm surpasses both of the BPSO and OBBA in terms of the mean fitness values of the discovered non-dominated solutions, across the six Java programs. Moreover, the fitness standard deviation values of the 10 non-dominated solutions discovered by the ABBA are very small; most of them are equal to zero or approaches zero which indicates the robustness of the ABBA. However, the BPSO could converge to solutions close to the ones discovered by the ABBA over the small size program MBisectOK, e.g. the non-dominated solutions (NDS) number 1, 4, 7 and 9 of the MBisectOk program.

In terms of the convergence speed, the ABBA could converge to the best solutions in less number of iterations than the OBBA over the six Java programs except MBisectOk (NDS 2,7,8,9, 10), MbubCorrect (NDS 8, 10) and MTringle (NDS 9); where, the OBBA converged faster than the ABBA. Nevertheless, the ABBA converged to better solutions than the ones discovered by the OBBA. So. It could be stated that the OBBA was prematurely converged in these cases. It should be noted that the MBisectOk, MbubCorrect and MTringle are of different sizes. Furthermore, it could be observed that the ABBA converges faster than the BPSO across the six Java programs.

In terms of the size of the reduced test suites and their fault detection capabilities, as could be observed from Table 5 the ABBA, in comparison to the BPSO and the OBBA, was able to discover the smallest reduced test suite across three programs of different sizes, MBisectOk, MbubCorrect and MfindOk. Moreover, the reduced test suites discovered by the ABBA have the highest fault detection capabilities, in terms of the number of killed mutants and FDR.

Across the MfourBall, MFindOk, MMidOK programs, the solutions found by the ABBA, in comparison to the OBBA and the BPSO, either have the highest fault detection capabilities but at the expense of the sizes of the discovered reduced test suites, e.g. MfourBall program; or the solutions have the smallest reduced test suite sizes but with less fault detection capabilities. So, the selection of the best solution across MfourBall, MFindOk, MMidOK depends on the testers' targets and testing budgets.

Fig. 1 shows sample convergence curves of the ABBA, OBBA and BPSO over the six programs. As could be observed that the ABBA converge faster and to better solutions, in terms of the fitness function values, than the OBBA and the BPSO, although the parameters settings of both of the ABBA and the OBBA are the same.

Answer to RQ2

We experimented with six Java programs with different search space sizes. The size of the test suite and the number of mutants are the indicators of the size of the search space. As could be observed from table 4 that the performance of the ABBA, in terms of the mean and SD of best fitness, is superior to the OBBA and the BPSO over the six programs of different search space sizes, which indicates that the ABBA is scalable.

CONCLUSION AND FUTURE WORK

This paper proposed a search-based optimization methodology for reducing the cost of the regression testing, through reducing the size of the test suite, with minimum or no loss of fault discovery capabilities. The proposed methodology is based on formulating the TSR problem as a multi-objective optimization problem, in terms of the test suite size and mutation score. Then, the original BBA was modified to enhance its exploration capabilities and boost its performance. Finally, a weighted sum strategy was utilized to guide the ABBA during the search for a Pareto front.

The effectiveness of the proposed ABBA in solving the TSR problem was evaluated using six Java programs. The experimental results showed that the performance of the proposed ABBA is superior to each of the OBBA and BPSO. In addition to, the ABBA converged to the best solutions faster than each of the OBBA and the BPSO.

As a further extension for this work, we plan to redefine the fitness function by considering more objectives such as test suite execution time and branch coverage. In addition, different weighting mechanisms will be tried and compared to this work. Finally, Co-evolution based on the ABBA may be considered to optimize the mutation testing while optimizing the regression testing; as the mutation testing is known to be expensive but effective in assessing the quality of a test suite.

Table 3. Weights used during experiments to generate 10 non-dominated solutions (NDS) which represents the Pareto front

NDS	1	2	3	4	5	6	7	8	9	10
Weight1	0.6	0.676	0.744	0.804	0.856	0.9	0.936	0.964	0.984	0.9960
Weight2	0.4	0.324	0.256	0.196	0.144	0.1	0.064	0.036	0.016	0.0040

Table 4. Comparison among the fitness values (fitness) and the convergence speed (i) of the 10 non-dominated solutions (NDS) discovered by the ABBA, OBBA AND BPSO

NDS	Algorithm		MEAN ± STD. DEV					
			MBisectOk	MBubCorrect	MFindOk	MFourBall	MMidOK	MTringle
1	ABBA	*fitness*	**10.57 ± 0**	**8.34 ± 1.46**	**54.60 ± 0**	**38.62 ± 0.02**	**8.34 ± 1.46**	**11.53 ±3.25**
		i	**15.33 ± 8.09**	**91.60 ± 6.28**	**68.50 ± 12.25**	**60.20 ± 23.06**	**69.70 ± 19.30**	**94.22 ± 5.95**
	OBBA	*fitness*	10.52 ± 0.06	2.72 ± 0.24	4.34 ± 0.97	4.34 ± 0.97	4.69 ± 1.08	2.99 ± 0.33
		i	28.30 ± 17.72	**91.50 ± 8.20**	92.90 ± 8.41	88.50 ± 9.31	88.80 ± 9.19	95.00 ± 4.03
	BPSO	*fitness*	**10.57** ± 0.01	2.71 ± 0.19	4.69 ± 1.04	12.59 ± 4.28	5.84 ± 1.76	2.94 ± 0.38
		i	67.60 ± 22.08	98.40 ± 1.58	98.70 ± 1.06	98.60 ± 1.96	99.50 ± 0.85	99.40 ± 0.97
2	ABBA	*fitness*	**8.75 ± 0**	**8.01 ± 2.73**	**44.42 ± 0**	**31.36 ± 0.02**	**8.01 ± 2.73**	**8.03 ± 1.25**
		i	29.11 ± 29.03	**93.10 ± 5.53**	**65.60 ± 9.82**	**61.00 ± 29.82**	**73.20 ± 18.28**	95.11 ± 6.68
	OBBA	*fitness*	8.73 ± 0.03	2.36 ± 0.17	3.69 ± 0.53	9.31 ± 7.96	4.72 ± 1.29	2.66 ± 0.32
		i	**19.30 ± 11.88**	94.80 ± 3.36	80.80 ± 15.16	77.40 ± 17.06	79.10 ± 15.06	**93.80 ± 5.21**
	BPSO	*fitness*	8.74 ± 0.01	2.32 ± 0.13	3.94 ± 0.43	5.70 ± 0.90	4.07 ± 0.47	2.63 ± 0.16
		i	68.80 ± 24.12	99.20 ± 1.03	99.30 ± 1.06	98.40 ± 1.58	99.70 ± 0.48	99.00 ± 1.33
3	ABBA	*fitness*	**7.11 ± 0.01**	**5.73 ± 1.33**	**35.16 ± 0.31**	**24..84 ± 0.02**	**5.73 ± 1.33**	**7.58 ± 1.47**
		i	**25.44 ± 22.60**	**90.60 ± 8.14**	**58.00 ± 17.00**	**52.20 ± 25.62**	**74.10 ± 14.06**	**91.00 ± 5.36**
	OBBA	*fitness*	7.07 ± 0.06	2.13 ± 0.08	3.22 ± 0.64	4.86 ± 2.87	3.74 ± 1.36	2.21 ± 0.20
		i	38.50 ± 22.87	91.00 ± 7.56	81.30 ± 10.93	81.90 ± 10.87	93.50 ± 16.85	92.70 ± 8.62
	BPSO	*fitness*	7.10 ± 0.01	2.08 ± 0.14	3.11 ± 0.35	5.62 ± 2.61	3.58 ± 0.47	2.11 ± 0.12
		i	70.80 ± 26.33	99.60 ± 0.84	99.60 ± 0.70	98.50 ± 1.90	99.50 ± 2.21	99.30 ± 0.82
4	ABBA	*fitness*	**5.67 ± 0**	**4.87 ± 0.77**	**27.26 ± 0**	**19.10 ± 0.02**	**4.87 ± 0.77**	**5.58 ± 1.60**
		i	**20.56 ± 18.92**	**90.90 ± 7.46**	**59.50 ± 12.26**	**68.30 ± 24.10**	**76.70 ±17..78**	93.11 ± 5.01
	OBBA	*fitness*	5.65 ± 0.03	1.86 ± 0.10	2.54 ± 0.37	5.49 ± 5.02	3.18 ± 1.25	1.80 ± 0.08
		i	30.60 ± 21.48	93.50 ± 3.75	88.70 ± 12.20	79.90 ± 12.35	79.60 ± 6.36	**91.90 ± 8.10**
	BPSO	*fitness*	**5.67 ± 0**	1.82 ± 0.11	2.82 ± 0.30	4.46 ± 1.03	3.08 ± 0.56	1.84 ± 0.10
		i	78.50 ± 10.76	98.80 ± 1.48	98.70 ± 1.64	98.90 ± 1.10	99.30 ± 0.85	99.70 ± 0.70
5	ABBA	*fitness*	**4.42 ± 0**	**3.70 ± 0.61**	**20.13 ± 0.35**	**14.14 ± 0.02**	**3.70 ± 0.61**	**3.82 ± 0.72**
		i	**20.44 ± 9.88**	**92.70 ± 7.63**	**69.20 ± 11.13**	**55.70 ± 23.31**	**67.40 ± 21.57**	**93.78 ± 5.80**
	OBBA	*fitness*	4.31 ± 0.13	1.61 ± 0.06	2.13 ± 0.24	3.84 ± 3.64	2.34 ± 0.25	1.58 ± 0.07
		i	31.90 ± 22.20	92.90 ± 7.65	91.50 ± 6.11	79.80 ± 12.67	89.30 ± 7.15	92.50 ± 9.92
	BPSO	*fitness*	4.41 ± 0.01	1.61 ± 0.07	2.45 ± 0.33	3.25 ± 0.64	2.48 ± 0.23	1.58 ± 0.09
		i	81.80 ± 9.11	99.80 ± 0.42	99.00 ± 0.94	99.00 ± 1.56	99.50 ± 1.08	99.10 ± 0.74
6	ABBA	*fitness*	**3.36 ± 0.01**	**2.95 ± 0.84**	**14.40 ± 0**	**9.93 ± 0.03**	**2.95 ± 0.48**	**3.18 ± 0.92**
		i	**19.89 ± 15.83**	**91.70 ± 6.06**	**68.20 ± 10.46**	**57.80 ± 23.71**	**75.20 ± 20.57**	**94.56 ± 4.33**
	OBBA	*fitness*	3.29 ± 0.09	1.49 ± 0.06	1.80 ± 0.26	2.14 ± 0.25	1.92 ± 0.20	1.40 ± 0.04
		i	26.10 ± 22.48	92.50 ± 7.92	89.30 ± 13.48	81.70 ± 14.68	87.00 ± 13.50	96.20 ±4.54
	BPSO	*fitness*	3.35 ± 0.02	1.40 ± 0.03	2.03 ± 0.22	3.44 ± 2.37	2.03 ± 0.23	1.37 ± 0.03
		i	64.00 ± 18.01	98.60 ± 2.01	99.00 ± 1.25	99.60 ± 0.97	99.60 ± 0.97	99.80 ± 0.42
7	ABBA	*fitness*	**2.49 ± 0.01**	**2.36 ± 0.36**	**8.96 ± 1.36**	**6.48 ± 0.03**	**2.36 ± 0.36**	**2.07 ± 0.29**
		i	26.33 ± 21.42	**90.60 ± 5.48**	**54.80 ± 10.38**	**67.20 ± 19.72**	**69.90 ± 16.3**	**91.56 ± 5.08**
	OBBA	*fitness*	2.46 ± 0.04	1.29 ± 0.04	1.55 ± 0.11	1.72 ± 0.21	1.56 ± 0.10	1.24 ± 0.02
		i	**18.40 ± 16.81**	94.10 ± 4.01	91.90 ± 8.33	71.40 ± 19.13	85.00 ± 15.63	93.30 ± 5.60
	BPSO	*fitness*	**2.49 ± 0.01**	1.27 ± 0.01	1.58 ± 0.10	2.41 ± 1.45	1.64 ± 0.17	1.22 ± 0.02
		i	74.10 ± 6.97	99.50 ± 0.85	99.30 ± 1.06	98.20 ± 1.23	99.50 ± 0.53	99.50 ± 0.71
8	ABBA	*fitness*	**1.82 ± 0**	**1.70 ± 0.10**	**5.73 ± 0.30**	**3.80 ± 0.03**	**1.70 ± 0.10**	**1.45 ± 0.08**
		i	29.67 ± 20.35	93.90 ± 4.53	**69.30 ± 17.21**	**73.70 ± 16.79**	**76.80 ± 12.39**	**93.56 ± 6.88**
	OBBA	*fitness*	1.76 ± 0.06	1.16 ± 0.02	1.30 ± 0.05	1.41 ± 0.09	1.33 ± 0.10	1.13 ± 0.01
		i	**21.00 ± 13.44**	**92.70 ± 7.62**	89.90 ± 14.70	79.10 ± 14.55	89.80 ± 9.73	94.40 ± 4.65
	BPSO	*fitness*	1.80 ± 0.04	1.15 ± 0.01	1.35 ± 0.12	1.54 ± 0.13	1.35 ± 0.09	1.13 ± 0.01
		i	76.50 ± 25.25	99.10 ± 1.37	98.20 ± 1.93	98.50 ± 1.27	99.20 ± 0.92	98.90 ± 1.20
9	ABBA	*fitness*	**1.33 ± 0.01**	**1.31 ± 0.09**	**3.05 ± 0.31**	**1.36 ± 0**	**1.31 ± 0.09**	**1.19 ± 0.03**
		i	22.22 ± 26.71	**90.00 ± 9.83**	**68.50 ± 20.70**	**56.90 ± 23.41**	**79.30 ± 15.41**	91.44 ± 6.06
	OBBA	*fitness*	1.30 ± 0.04	1.07 ± 0.01	1.13 ± 0.03	1.17 ± 0.04	1.13 ± 0.03	1.06 ± 0.01
		i	**21.90 ± 14.04**	93.90 ± 6.98	89.90 ± 7.48	74.90 ± 16.13	85.90 ± 14.83	**87.20 ±10.26**
	BPSO	*fitness*	**1.33 ± 0.01**	1.06 ± 0.01	1.14 ± 0.03	1.21 ± 0.07	1.14 ± 0.02	1.05 ± 0
		i	75.60 ± 20.22	95.90 ± 11.23	98.50 ± 1.27	99.00 ± 1.25	99.60 ± 0.52	99.60 ± 0.52
10	ABBA	*fitness*	**1.05 ± 0**	**1.08 ± 0.02**	**1.54 ± 0**	**1.08 ± 0.01**	**1.08 ± 0.02**	**1.04 ± 0**
		i	42.22 ± 30.97	92.50 ± 6.08	**64.30 ± 13.65**	**54.30 ± 21.68**	**76.50 ± 14.16**	**88.56 ± 11.28**
	OBBA	*fitness*	1.04 ± 0.01	1.02 ± 0	1.03 ± 0.01	1.04 ± 0.01	1.03 ± 0.01	1.01 ± 0
		i	**21.90 ± 18.78**	**89.40 ± 8.37**	90.50 ±6.52	83.40 ± 12.03	90.30±9.32	91.60 ± 7.88
	BPSO	*fitness*	**1.05 ± 0.01**	1.02 ± 0	1.03 ± 0.01	1.05 ± 0.01	1.03 ± 0.01	1.01 ± 0
		i	71.90 ± 16.63	99.40 ± 0.84	98.40 ±2.22	99.50 ± 0.71	99.40±0.70	99.20 ± 0.92

Table 5. Comparison among the 10 non-dominated solutions discovered by the ABBA, OBBA and BPSO in terms of the fault detection rate (FDR), number of killed mutants (KM) and size of reduced test suite (RTS)

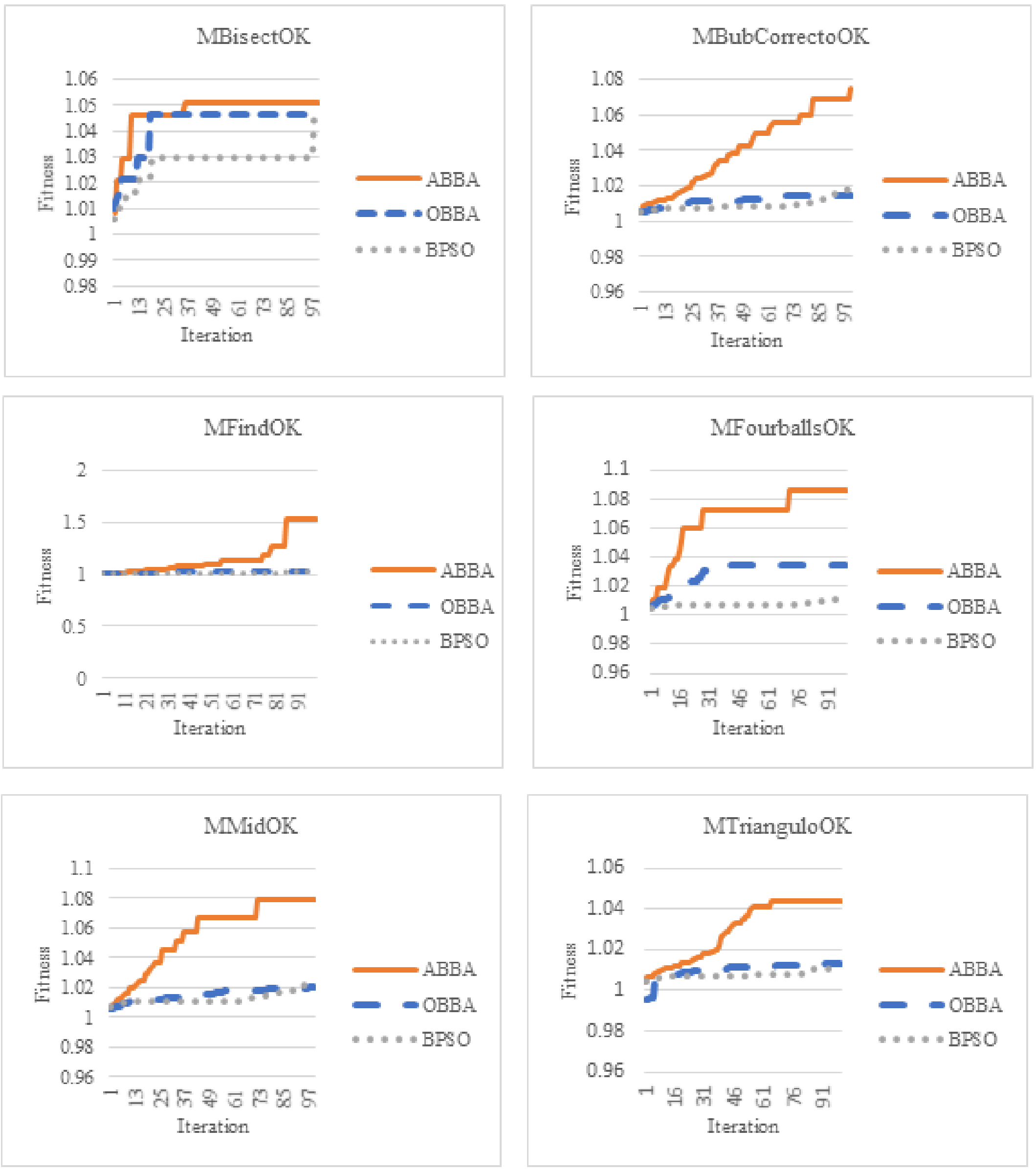
NDS	Algorithm		MEAN ± STD. DEV					
			MBisectOk	MBubCorrect	MFindOk	MFourBall	MMidOK	MTringle
1	ABBA	FDR	**95.45 ± 0**	**96.3 ± 0.32**	**99.4 ± 0**	37.4 ± 4.02	41.6± 11.06	73.7± 30.08
		KM	**42 ± 0**	**67.9 ± 0.32**	**178 ± 0**	63.2 ± 4.02	57.40 ± 11.06	161.60 ± 30.08
		RTS	**1 ± 0**	**13.88 ± 2.70**	**1 ± 0**	**1 ± 0**	**1.1 ± 0.31**	8.4 ± 2.32
	OBBA	FDR	86.4± 4.09	97.1 ± 0	99.4 ± 0	**83.4± 20.99**	**94.7± 8.62**	87.7±16.29
		KM	38 ± 4.09	68 ± 0	178 ± 0	**140.1 ± 20.99**	**130.7 ± 8.62**	209 ± 16.29
		RTS	1 ± 0	48.6 ± 5.19	15.3 ± 3.80	7.9 ± 2.42	12.6 ± 2.41	35.5 ± 3.66
	BPSO	FDR	95.0 ± 0.6	97.1 ± 0	99.4 ± 0	61.7± 15.33	94.4 ± 6.7	**87.9± 8.31**
		KM	41.8 ± 0.6	68 ± 0	178 ± 0	103.6 ± 15.33	130.3 ± 6.7	**210.1 ± 8.31**
		RTS	1 ± 0	50.6 ± 4.14	13.6 ±1.78	6.3 ± 1.49	11.9 ± 1.52	34.4 ± 3.02
2	ABBA	FDR	**95.45 ± 0**	**97.0 ± 0**	99.4±74.21	37.6± 3.85	41.2± 3.65	67.7± 23.96
		KM	**42 ± 0**	**68 ± 0**	142.80 ± 74.21	58.8 ± 3.85	56.80 ± 3.65	164.20 ± 23.96
		RTS	**1 ± 0**	**13.75 ± 2.66**	**1 ± 0**	**1 ± 0**	**1.1 ± 0.32**	**9.5 ± 1.72**
	OBBA	FDR	92.5 ± 1.55	97.1 ± 0	**99.4 ± 0**	75.4± 30.37	91.7± 11.55	**91.6 ± 9.33**
		KM	40.7 ± 1.55	68 ± 0	**178 ± 0**	126.6 ± 30.37	126.5 ± 11.55	**219 ± 9.33**
		RTS	1 ± 0	49.5 ± 4.85	14.9 ± 2.47	4.9 ± 1.97	10.6 ± 2.84	35 ± 4.30
	BPSO	FDR	95.0 ± 0.6	97.1 ± 0.32	**99.4 ± 0**	**82.6± 25.13**	**95.1± 11.12**	86.2± 13.72
		KM	41.8 ± 0.6	67.9 ± 0.32	**178 ± 0**	**138.7 ± 25.13**	**131.2 ± 11.12**	206 ± 13.72
		RTS	1 ± 0	50.6 ± 4.14	13.6 ±1.78	6.3 ± 1.49	11.9 ± 1.52	34.4 ± 3.02
3	ABBA	FDR	**95.0 ± 0.63**	97.1 ± 2.83	**79.8 ± 0**	35.0 ± 4.98	46.9± 26.62	78.7±32.72
		KM	**41.8 ± 0.63**	67 ± 2.83	**178 ± 0**	58.9 ± 4.98	64.70 ± 26.62	173.30 ± 32.72
		RTS	**1 ± 0**	**14.63 ± 2.56**	**1 ± 0**	**1 ± 0**	**1 ± 0**	**8.2 ± 1.99**
	OBBA	FDR	89.5 ± 3.23	**97.1 ± 0**	99.4 ± 0	71.2± 34.58	**97.0 ± 4.92**	**94.1 ± 7.94**
		KM	39.4 ± 3.23	**68 ± 0**	178 ± 0	119.6 ± 34.58	**133.8 ± 4.92**	**255 ± 7.94**
		RTS	1 ± 0	47.3 ± 2.75	14.8 ± 3.79	7.1 ± 2.56	12.2 ± 4.61	37.3 ± 5.56
	BPSO	FDR	94.5 ± 0.8	**97.1 ± 0**	99.4 ± 0	**83.2± 30.12**	94.9± 9.27	92.8± 9.69
		KM	41.6 ± 0.8	**68 ± 0**	178 ± 0	**139.7 ± 30.12**	130.3 ± 9.27	221.7 ± 9.69
		RTS	1 ± 0	49.5± 5.33	14.9 ± 2.08	5.7 ± 1.77	11.3 ± 1.95	39.3 ± 3.53
4	ABBA	FDR	**95.45 ± 0**	**95.7 ± 0.42**	99.4± 73.58	35.1 ± 4.06	40.1± 8.42	72.5± 29.21
		KM	**42 ± 0**	**67.8 ± 0.42**	143.10 ± 73.58	62.3 ± 4.06	55.40 ± 8.42	179.30 ± 29.21
		RTS	**1 ± 0**	**12.88 ± 2.80**	**1 ± 0**	**1 ± 0**	**1.3 ± 0.68**	**9.20 ± 2.62**
	OBBA	FDR	93.0 ± 1.58	97.1 ± 0	**99.4 ± 0**	76.7± 42.40	91.8± 14.41	**95.9± 2.57**
		KM	40.9 ± 1.58	68 ± 0	**178 ± 0**	128.9 ± 42.40	126.7 ± 14.41	**229.3 ± 2.57**
		RTS	1 ± 0	47.7 ± 5.16	15.8 ± 2.78	6.4 ± 3.34	11.4 ± 3.20	41.2 ± 3.08
	BPSO	FDR	**95.5 ± 0**	97.1 ± 0	**99.4 ± 0**	**82.0 ± 30.60**	**94.9± 4.33**	95.7 ± 4.89
		KM	**42 ± 0**	68 ± 0	**178 ± 0**	**137.8 ± 30.60**	**130.9 ± 4.33**	228.8 ± 4.89
		RTS	**1 ± 0**	49.8 ± 5.33	13.4 ± 2.12	5.3 ± 1.34	11 ± 2.45	40.1 ± 4.36
5	ABBA	FDR	**95.45 ± 0**	**96.9 ± 0.42**	79.9±73.58	37.1 ± 4.06	41.2± 5.47	75.0± 29.21
		KM	**42 ± 0**	**67.8 ± 0.42**	143.10 ± 73.58	62.3 ± 4.06	56.90 ± 5.47	179.30 ± 29.21
		RTS	**1 ± 0**	**13.86 ± 2.34**	**1 ± 0**	**1 ± 0**	**1 ± 0**	**10.30 ± 2.31**
	OBBA	FDR	83.2 ± 6.35	97.1 ± 0	**99.4 ± 0**	97.6± 40.20	**95.8± 0.32**	**97.7± 1.26**
		KM	36.6 ± 6.35	68 ± 0	**178 ± 0**	142.6 ± 40.20	**132.2 ± 0.32**	**233.6 ± 1.26**
		RTS	1 ± 0	48.7 ± 3.68	15.7 ± 2.90	7.2 ± 3.05	12.1 ± 2.33	42.1 ± 3.81
	BPSO	FDR	95.0 ± 0.6	97.1 ± 0	**99.4 ± 0**	**85.7± 25.01**	94.7± 8.88	96.7± 2.35
		KM	41.8 ± 0.6	68 ± 0	**178 ± 0**	**144 ± 25.01**	130.7 ± 8.88	231.2 ± 2.35
		RTS	1 ± 0	49 ± 4.27	12.7 ± 2.71	6 ± 1.94	10.9 ± 1.59	41.8 ± 4.89

continued on following page

Table 5.. Continued

NDS	Algorithm		MEAN ± STD. DEV					
			MBisectOk	MBubCorrect	MFindOk	MFourBall	MMidOK	MTringle
6	ABBA	FDR	**41.8 ± 0.63**	96.4 ± 0.71	**99.4 ± 0**	36.5 ± 5.33	41.5 ± 2.38	78.6± 27.23
		KM	**41.8 ± 0.63**	67.5 ± 0.71	**178 ± 0**	61.3 ± 5.33	57.30 ± 2.38	187.90 ± 27.23
		RTS	**1 ± 0**	**12.75 ± 1.98**	**1 ± 0**	**1 ± 0**	**1 ± 0**	**9.80 ± 3.05**
	OBBA	FDR	87.5 ± 4	**97.1 ± 0**	99.4 ± 0	**93.9± 17.21**	**97.7 ± 2.53**	**98.5± 1.64**
		KM	38.5 ± 4	**68 ± 0**	178 ± 0	**157.7 ± 17.21**	**134.8 ± 2.53**	**235.3 ± 1.64**
		RTS	1 ± 0	43.6 ± 4.85	16 ± 3.80	7.8 ± 1.69	12.3 ± 2.00	42.3 ± 3.27
	BPSO	FDR	94.1 ± 0.92	**97.1 ± 0**	99.4 ± 0	78.8± 41.94	93.6± 9.90	97.9± 3.49
		KM	41.4 ± 0.92	**68 ± 0**	178 ± 0	132.3 ± 41.94	129.1 ± 9.90	234 ± 3.49
		RTS	1 ± 0	51.5 ± 3.47	12.4 ± 2.41	5.5 ± 3.24	10.9 ± 2.29	44.6 ± 3.27
7	ABBA	FDR	**95.0 ± 0.63**	**96.9± 0.42**	97.9 ± 73.58	35.9 ± 5.10	41.8± 15.72	86.4±19.42
		KM	**41.8 ± 0.63**	**67.8 ± 0.42**	143 ± 73.58	60.3 ± 5.10	57.70 ± 15.72	206.60±19.42
		RTS	**1 ± 0**	**11.63 ± 2.67**	**1.1 ± 0.32**	**1 ± 0**	**1.1 ± 0.32**	**11.7 ± 3.06**
	OBBA	FDR	91.8 ± 1.9	97.1 ± 0	**99.4 ± 0**	**95.9± 12.49**	97.7 ± 2.71	**99.8± 0.70**
		KM	40.4 ± 1.9	68 ± 0	**178 ± 0**	**161.1 ± 12.49**	134.7 ± 2.71	**238.5 ± 0.70**
		RTS	1 ± 0	46.9 ± 5.15	14.5 ± 2.99	8 ± 2	12.5 ± 2.12	46 ± 3.27
	BPSO	FDR	**95.0 ± 0.60**	97.1 ± 0	**99.4 ± 0**	87.4 ± 33.79	**98.1± 3.30**	99.4± 1.84
		KM	**41.8 ± 0.60**	68 ± 0	**178 ± 0**	146.9 ± 33.79	**135.4 ± 3.30**	237.6 ± 1.84
		RTS	**1 ± 0**	48.7 ± 1.34	13.7 ± 2	5.7 ± 2.58	11.7 ± 2.98	48.3 ± 4.45
8	ABBA	FDR	**95.0 ± 0**	**97.1 ± 0**	89.7±55.34	35.8 ± 5.15	42.0± 6.01	95.0 ±7.41
		KM	**42 ± 0**	**68 ± 0**	160.50 ± 55.34	60.1 ± 5.15	57.90 ± 6.01	227 ±7.41
		RTS	**1 ± 0**	**12.25 ± 1.58**	**1 ± 0**	**1 ± 0**	**1 ± 0**	**15.9 ± 2.58**
	OBBA	FDR	88.9 ± 2.62	97.1 ± 0	**99.4 ± 0**	**98.5± 1.64**	98.4 ± 3.08	**100± 0.32**
		KM	39.1 ± 2.62	68 ± 0	**178 ± 0**	**165.4 ± 1.64**	135.8 ± 3.08	**238.9 ± 0.32**
		RTS	1 ± 0	46.3 ± 4.97	14.9 ± 2.23	7.8 ± 1.87	12.6 ± 3.13	47.9 ± 3.81
	BPSO	FDR	93.9 ± 0.6	97.1 ± 0	**99.4 ± 0**	98.2± 33.79	**95.7± 3.30**	99.9 ± 1.84
		KM	41.8 ± 0.6	68 ± 0	**178 ± 0**	146.9 ± 33.79	**135.4 ± 3.30**	237.6 ± 1.84
		RTS	1 ± 0	48.7 ± 1.34	13.7 ± 2	5.7 ± 2.58	11.7 ± 2.98	48.3 ± 4.45
9	ABBA	FDR	**95.45 ± 0.63**	**96.6 ± 0**	89.7±55.34	99.3 ± 0.42	41.4± 13.13	98.4 ±2.88
		KM	**41.8 ± 0.63**	**67.6 ± 0**	160.50 ± 55.34	166.8 ± 0.42	57.20 ± 13.13	235.1 ±2.88
		RTS	**1 ± 0**	**13.50 ± 4.14**	**1 ± 0**	**4 ± 0**	**1.1 ± 0.32**	**15 ± 2.77**
	OBBA	FDR	91.4 ± 1.83	97.1 ± 0	**99.4 ± 0**	**99.4 ± 0.94**	**99.9± 0.42**	**100 ± 0.0**
		KM	40.2 ± 1.83	68 ± 0	**178 ± 0**	**167 ± 0.94**	**137.8 ± 0.42**	**239 ± 0.0**
		RTS	1 ± 0	47.7 ± 3.19	15 ± 2.83	8.3 ± 1.64	14 ± 2.70	49 ± 4.94
	BPSO	FDR	95.0 ± 1.55	97.1 ± 0	**99.4 ± 0**	99.2± 4.03	99.1 ± 10.39	100± 0.68
		KM	41.3 ± 1.55	68 ± 0	**178 ± 0**	165 ± 4.03	132.1 ± 10.39	238.7 ± 0.68
		RTS	1 ± 0	48.7 ± 2.87	13.5 ± 4.03	6.1 ± 1.59	11.4 ± 3.34	47.5 ± 4.03
10	ABBA	FDR	**95.0 ± 1.05**	**97.1 ± 0**	**99.4 ± 0**	99.3 ± 0.42	99.1 ± 1.06	**100 ± 0**
		KM	**43 ± 1.05**	**68 ± 0**	**178 ± 0**	166.8 ± 0.42	136.70 ± 1.06	**239 ± 0**
		RTS	**1.1 ± 0.3**	**13 ± 3.38**	**1 ± 0**	**4.2 ± 0.42**	**5 ± 0.67**	**20.6 ± 1.51**
	OBBA	FDR	99.5 ± 0.6	97.1 ± 0	99.4 ± 0	**99.9 ± 0.6**	**100.0± 0**	100± 0.0
		KM	43.8 ± 0.6	68 ± 0	178 ± 0	**167.8 ± 0.6**	**138 ± 0**	239 ± 0.0
		RTS	2.6 ± 0.8	47.4±4.43	16.4 ± 3.17	9.1 ± 1.52	13.7 ± 3.09	46 ± 4.16
	BPSO	FDR	97.3 ± 0.68	97.1 ± 0	99.4 ± 0	99.8± 0.97	99.9± 0.32	100 ± 0
		KM	42.8 ± 0.98	68 ± 0	178 ± 0	167.6 ± 0.97	137.9 ± 0.32	239 ± 0
		RTS	1.6 ± 0.8	49.2 ± 4.87	14.4 ± 2.06	7.8 ± 1.93	13.2 ± 2.15	53.9 ± 4.25

Figure 2. Sample convergence curves of the ABBA, OBBA and BPSO over the dataset



REFERENCES

Agrawal, A. P., Choudhary, A., Kaur, A., & Pandey, H. M. (2019). Fault coverage-based test suite optimization method for regression testing: Learning from mistakes-based approach. *Neural Computing & Applications*, 1–16. doi:10.1007/s00521-019-04098-9

Andrews, J. H., Briand, L. C., Labiche, Y., & Namin, A. S. (2006). Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering*, *32*(8), 608–624. doi:10.1109/TSE.2006.83

Assi, R. A., Masri, W., & Trad, C. (2018). *Substate Profiling for Effective Test Suite Reduction.* Paper presented at the 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE). doi:10.1109/ISSRE.2018.00023

Bansal, J. C., Singh, P., Saraswat, M., Verma, A., Jadon, S. S., & Abraham, A. (2011). *Inertia weight strategies in particle swarm optimization.* Paper presented at the 2011 Third world congress on nature and biologically inspired computing. doi:10.1109/NaBIC.2011.6089659

Bhatia, P. K. (2020). Test Case Minimization in COTS Methodology Using Genetic Algorithm: A Modified Approach. *Proceedings of ICETIT*, *2019*, 219–228.

Carvalho, I. A., da Rocha, D. G., Silva, J. G. R., da Fonseca Vieira, V., & Xavier, C. R. (2017). *Study of parameter sensitivity on bat algorithm.* Paper presented at the International Conference on Computational Science and Its Applications. doi:10.1007/978-3-319-62392-4_36

Catal, C., & Mishra, D. (2013). Test case prioritization: A systematic mapping study. *Software Quality Journal*, *21*(3), 445–478. doi:10.1007/s11219-012-9181-z

Chawla, M., & Duhan, M. (2015). Bat algorithm: A survey of the state-of-the-art. *Applied Artificial Intelligence*, *29*(6), 617–634. doi:10.1080/08839514.2015.1038434

Chen, T. Y., & Lau, M. F. (1998). A simulation study on some heuristics for test suite reduction. *Information and Software Technology*, *40*(13), 777–787. doi:10.1016/S0950-5849(98)00094-9

Coviello, C., Romano, S., Scanniello, G., Marchetto, A., Antoniol, G., & Corazza, A. (2018). *Clustering support for inadequate test suite reduction.* Paper presented at the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). doi:10.1109/SANER.2018.8330200

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. doi:10.1109/MHS.1995.494215

Frankl, P. G., Weiss, S. N., & Hu, C. (1997). All-uses vs mutation testing: An experimental comparison of effectiveness. *Journal of Systems and Software*, *38*(3), 235–253. doi:10.1016/S0164-1212(96)00154-9

Gary, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company.

Geng, J., Li, Z., Zhao, R., & Guo, J. (2016). *Search based test suite minimization for fault detection and localization: A co-driven method.* Paper presented at the International Symposium on Search Based Software Engineering. doi:10.1007/978-3-319-47106-8_3

Gotlieb, A., & Marijan, D. (2014). FLOWER: optimal test suite reduction as a network maximum flow. *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. doi:10.1145/2610384.2610416

Gupta, N., Sharma, A., & Pachariya, M. K. (2020). *Multi-objective test suite optimization for detection and localization of software faults*. Academic Press.

Hamdy, A. (2014). Genetic fuzzy system for enhancing software estimation models. *International Journal of Modeling Optimization*, *4*(3), 227–232. doi:10.7763/IJMO.2014.V4.378

Hamdy, A., & El-Laithy, A. (2020). Semantic categorization of software bug repositories for severity assignment automation. In *Integrating Research and Practice in Software Engineering* (pp. 15–30). Springer. doi:10.1007/978-3-030-26574-8_2

Hamdy, A., & Ellaithy, A. (2020). Multi-Feature approach for Bug severity assignment. *International Journal of Open Source Software and Processes*, *11*(2), 1–15. doi:10.4018/IJOSSP.2020040101

Hamdy, A., & Ezzat, G. (2020). Deep mining of open source software bug repositories. *International Journal of Computers and Applications*, 1–9. doi:10.1080/1206212X.2020.1855705

Hamdy, A., & Mohamed, A. (2019). Greedy binary particle swarm optimization for multi-objective constrained next release problem. *International Journal of Machine Learning and Computing*, *9*(5), 561–568. doi:10.18178/ijmlc.2019.9.5.840

Jayabarathi, T., Raghunathan, T., & Gandomi, A. (2018). The bat algorithm, variants and some practical engineering applications: a review. In *Nature-Inspired Algorithms and Applied Optimization* (pp. 313–330). Springer. doi:10.1007/978-3-319-67669-2_14

Jia, Y., & Harman, M. (2010). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, *37*(5), 649–678. doi:10.1109/TSE.2010.62

Kennedy, J., & Eberhart, R. C. (1997). *A discrete binary version of the particle swarm algorithm.* Paper presented at the 1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation. doi:10.1109/ICSMC.1997.637339

Khan, S. U. R., Lee, S. P., Javaid, N., & Abdul, W. (2018). A systematic review on test suite reduction: Approaches, experiment's quality evaluation, and guidelines. *IEEE Access: Practical Innovations, Open Solutions*, *6*, 11816–11841. doi:10.1109/ACCESS.2018.2809600

Leung, H. K., & White, L. (1989). Insights into regression testing (software testing). *Proceedings. Conference on Software Maintenance*.

Lin, C.-T., Tang, K.-W., Wang, J.-S., & Kapfhammer, G. M. (2017). Empirically evaluating Greedy-based test suite reduction methods at different levels of test suite complexity. *Science of Computer Programming*, *150*, 1–25. doi:10.1016/j.scico.2017.05.004

Mirjalili, S., Mirjalili, S. M., & Yang, X.-S. (2014). Binary bat algorithm. *Neural Computing & Applications*, *25*(3-4), 663–681. doi:10.1007/s00521-013-1525-5

Mohanty, S., Mohapatra, S. K., & Meko, S. F. (2020). Ant Colony Optimization (ACO-Min) Algorithm for Test Suite Minimization. In Progress in Computing, Analytics and Networking (pp. 55-63). Springer.

Mohapatra, S. K., Mishra, A. K., & Prasad, S. (2020). Intelligent Local Search for Test Case Minimization. *Journal of The Institution of Engineers: Series B*, 1-11.

Nadeem, A., & Awais, A. (2006). *TestFilter: a statement-coverage based test case reduction technique.* Paper presented at the 2006 IEEE International Multitopic Conference.

Ngatchou, P., Zarei, A., & El-Sharkawi, A. (2005). Pareto multi objective optimization. *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*. doi:10.1109/ISAP.2005.1599245

Offutt, A. J., Lee, A., Rothermel, G., Untch, R. H., & Zapf, C. (1996). An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering, 5*(2), 99-118.

Offutt, A. J., & Lee, S. D. (1994). An empirical evaluation of weak mutation. *IEEE Transactions on Software Engineering*, *20*(5), 337–344. doi:10.1109/32.286422

Offutt, A. J., Pan, J., Tewary, K., & Zhang, T. (1996). An experimental evaluation of data flow and mutation testing. *Software, Practice & Experience*, *26*(2), 165–176. doi:10.1002/(SICI)1097-024X(199602)26:2<165::AID-SPE5>3.0.CO;2-K

Pargas, R. P., Harrold, M. J., & Peck, R. R. (1999). Test-data generation using genetic algorithms. *Software Testing, Verification & Reliability*, *9*(4), 263–282. doi:10.1002/(SICI)1099-1689(199912)9:4<263::AID-STVR190>3.0.CO;2-Y

Polo, M., Piattini, M., & García-Rodríguez, I. (2009). Decreasing the cost of mutation testing with second-order mutants. *Software Testing, Verification & Reliability*, *19*(2), 111–131. doi:10.1002/stvr.392

Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2016). 15 years of software regression testing techniques—A survey. *International Journal of Software Engineering and Knowledge Engineering*, *26*(05), 675–689. doi:10.1142/S0218194016300013

Sun, J.-z., & Wang, S.-y. (2010). *A novel chaos discrete particle swarm optimization algorithm for test suite reduction.* Paper presented at the 2nd International Conference on Information Science and Engineering.

Usaola, M. P., Mateo, P. R., & Lamancha, B. P. (2012). *Reduction of test suites using mutation.* Paper presented at the International Conference on Fundamental Approaches to Software Engineering. doi:10.1007/978-3-642-28872-2_29

Walsh, P. J. (1985). *A measure of test case completeness (software, engineering).* Academic Press.

Wang, S., Ali, S., & Gotlieb, A. (2015). Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software*, *103*, 370–391. doi:10.1016/j.jss.2014.08.024

Wei, Z., Xiaoxue, W., Xibing, Y., Shichao, C., Wenxin, L., & Jun, L. (2017). *Test suite minimization with mutation testing-based many-objective evolutionary optimization.* Paper presented at the 2017 International Conference on Software Analysis, Testing and Evolution (SATE). doi:10.1109/SATE.2017.12

Xin, J., Chen, G., & Hai, Y. (2009). *A particle swarm optimizer with multi-stage linearly-decreasing inertia weight.* Paper presented at the 2009 International Joint Conference on Computational Sciences and Optimization. doi:10.1109/CSO.2009.420

Xue, Y., & Li, Y.-F. (2020). Multi-objective Integer Programming Approaches for Solving the Multi-criteria Test-suite Minimization Problem: Towards Sound and Complete Solutions of a Particular Search-based Software-engineering Problem. *ACM Transactions on Software Engineering and Methodology*, *29*(3), 1–50. doi:10.1145/3392031

Yang, X.-S. (2010). A new metaheuristic bat-inspired algorithm. In Nature inspired cooperative strategies for optimization (NICSO 2010) (pp. 65-74). Springer. doi:10.1007/978-3-642-12538-6_6

Yang, X.-S. (2011). Bat algorithm for multi-objective optimisation. *International Journal of Bio-inspired Computation*, *3*(5), 267–274. doi:10.1504/IJBIC.2011.042259

Yoo, S., & Harman, M. (2007). Pareto efficient multi-objective test case selection. *Proceedings of the 2007 international symposium on Software testing and analysis*. doi:10.1145/1273463.1273483

ENDNOTES

[1] http://www.inf-cr.uclm.es/www/mpolo/stvr/
[2] https://ise.gmu.edu/~ofut/mujava/
[3] http://alarcos.inf-cr.uclm.es/testooj3

Nagwa R. Fisal was born in Ismailia, Egypt in 1994. She received the B.S. degree in Computer Science from Suez Canal University, Ismailia, Egypt in 2015. Since 2016 she is with the Department of Mathematics, Faculty of Science, Suez Canal University. Her research interests are Software Testing and Computational Intelligent.

Abeer Hamdy is an associate professor at the Faculty of Informatics and Computer Science, British University in Egypt (BUE) since April 2013; during (2009-2013) was a lecturer. Prior to joining BUE, she served as a Teaching Assistant (1992-1996) at the 10th of Ramadan higher technological institute in Egypt, an Assistant Researcher (1996-2003), Research Scientist (2003-2009) at the Electronics research institute in Egypt. She received her B.Sc. degree with honors, M.Sc. and Ph.D. degrees in Electronics and Electrical communications from the Faculty of Engineering, Cairo University in 1992, 1998,2003 respectively. She was awarded two fellowships to conduct post doctoral research at University of Connecticut (2005-2006) and University of Central Florida (2007-2008) at the United States. Her research interests includes search based software engineering, software design and software maintenance.

Essam Rashed received his Ph.D. (Eng.) in Computer Science from the University of Tsukuba, Tsukuba, Japan in 2010. From 2010 to 2012, he was a Research Fellow of the Japan Society for the Promotion of Science (JSPS) at the University of Tsukuba. He served as Assistant/Associate Professor of Computer Science at the Department of Mathematics, Faculty of Science, Suez Canal University from 2012. Currently, he is a Research Associate Professor at Nagoya Institute of Technology. His research interests include medical image processing, data analysis and visualization, artificial intelligence and pattern recognition. Dr. Rashed is IEEE Senior Member and Associate Editor of IEEE Access. He is a recipient of the Egyptian National Doctoral Scholarship (2006), JSPS postdoctoral fellowship (2010), JAMIT best presentation award (2008 & 2012), and Chairman Award, Department of Computer Science, University of Tsukuba (2010). He participated as a PI and CoI for several external funded projects.