Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2022.019669
Article

Tech Science Press

Multi-Objective Adapted Binary Bat for Test Suite Reduction

Nagwa Reda[1], Abeer Hamdy[2,*] and Essam A. Rashed[1,3]

[1]Department of Mathematics, Faculty of Science, Suez Canal University, Egypt
[2]Faculty of Informatics and Computer Science, British University in Egypt, Egypt
[3]Department of Electrical and Mechanical Engineering, Nagoya Institute of Technology, Japan
*Corresponding Author: Abeer Hamdy. Email: abeer.hamdy@bue.edu.eg

Received: 21 April 2021; Accepted: 16 June 2021

Abstract: Regression testing is an essential quality test technique during the maintenance phase of the software. It is executed to ensure the validity of the software after any modification. As software evolves, the test suite expands and may become too large to be executed entirely within a limited testing budget and/or time. So, to reduce the cost of regression testing, it is mandatory to reduce the size of the test suite by discarding the redundant test cases and selecting the most representative ones that do not compromise the effectiveness of the test suite in terms of some predefined criteria such as its fault-detection capability. This problem is known as test suite reduction (TSR); and it is known to be as nondeterministic polynomial-time complete (NP-complete) problem. This paper formulated the TSR problem as a multi-objective optimization problem; and adapted the heuristic binary bat algorithm (BBA) to resolve it. The BBA algorithm was adapted in order to enhance its exploration capabilities during the search for Pareto-optimal solutions. The effectiveness of the proposed multi-objective adapted binary bat algorithm (MO-ABBA) was evaluated using 8 test suites of different sizes, in addition to twelve benchmark functions. Experimental results showed that, for the same fault discovery rate, the MO-ABBA is capable of reducing the test suite size more than each of the multi-objective original binary bat (MO-BBA) and the multi-objective binary particle swarm optimization (MO-BPSO) algorithms. Moreover, MO-ABBA converges to the best solutions faster than each of the MO-BBA and the MO-BPSO.

Keywords: Binary bat algorithm; mutation testing; regression testing; software testing; test suite reduction; heuristic algorithms

1 Introduction

Software testing is one of the crucial activities in the software development lifecycle. It is used to detect software defects and ensure that the software is delivered with high quality. Any modifications in one of the software components may affect one or more other components, which necessitates the re-execution of the earlier test cases in addition to the newly generated ones [1]. As a result, the test suite size expands over time and may include redundant test cases. Moreover, it may not execute in it is entirely within the testing budget and/or time. Testing the behavior of the whole system under test (SUT) after each modification is called regression testing [2]. Regression testing is expensive and it accounts for almost one half of the cost of testing [3]. Rothermel et al. [3] reported a case in the industry where executing a complete regression test suite (of a 20K LOC product) required seven weeks. In order to reduce the cost of regression testing, it is necessary to reduce the number of test cases in the test suite without compromising its effectiveness, in terms of some predefined criteria [4,5]. This problem is known as the test suite reduction (TSR) problem. 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 [6] is a white-box testing technique, which calculates a score for the test suite that indicates its capability on discovering bugs in the SUT. The TSR problem is known to be a combinational multi-objective optimization problem, that can be described as a set covering problem; which is known to be NP-complete. In practice, there is no exact solution for NP-complete problems, however, suboptimal solutions could be found using search-based optimization algorithms [7].

Researchers have proposed various test suite reduction approaches [8]. The majority of these approaches are in the form of a white-box [9]. They aim at reducing the test suite without compromising test requirements such as statement coverage, fault detection capability rate, etc. Most of these approaches utilized greedy algorithms to solve a single objective TSR (SO-TSR) problem [8]. Recent approaches used heuristic algorithms to solve the multi-objective TSR (MO-TSR) problem [8,10–15] in addition to the SO-TSR [16,17]. Some studies demonstrate a comparison between the performance of heuristic and greedy algorithms in solving the TSR problem; Sun et al. [17] showed experimentally that the Binary Particle Swarm Optimization (BPSO) algorithm outperforms greedy algorithms in solving the SO-TSR problem. Most of heuristic based approaches for solving the MO-TSR problem utilized Non-dominated Sorting Genetic Algorithm II (NSGAII) or its variants [10,14,15]. Yoo et al. [12] showed experimentally that the NSGAII is superior to greedy approaches in solving the MO-TSR problem. Wang et al. [13] used three types of weighted-sum genetic algorithm (GA) and showed experimentally the superiority of the random-weighted GA over the NSGAII and some other popular multi-objective optimization algorithms. Moreover, Mohanty et al. [11] used Ant Colony (ACO) in their proposed approach for solving the MO-TSR problem.

Aims and Contributions

The main aim of this paper is to propose a heuristic based approach to solve the multi-objective TSR problem. The proposed approach is based on the Bat algorithm (BA) [18,19]; which is a swarm intelligence search algorithm inspired from the echolocation behavior of bats. Echolocation works as a type of sonar; a bat emits a loud sound, and an echo returns when that sound hits an object. The combination of echolocation with swarm intelligence enhances the properties of swarm-based algorithms; which makes BA more effective than other swarm-based algorithms in some contexts [18,20]. Yang [18] showed empirically that BA outperforms each of the GAs [21] and Particle swarm Optimization (PSO) [22,23] over some benchmark functions. Chawla et al. [20] surveyed some applications in the areas of computer science, medical and electrical engineering where the BA surpasses GA, PSO and ACO algorithms. However, the BBA occasionally fails to discover the global best solution for some multi-modal functions.

Our contributions to accomplish the above-mentioned aim can be summarized as follows:

Proposing modifications to the original Binary Bat algorithm (BBA) to enhance its exploration and exploitation capabilities; to reduce its occasional failure to converge to global optimum solutions.

Evaluating the performance of the adapted (modified) binary bat algorithm (ABBA) against each of the BBA and the binary PSO (BPSO) [24] algorithms, over a set of different unimodal and multi-modal benchmark functions of different ranges (the unimodal function has only one minima location which is the global best solution; while the multi-modal function has more than one local minima locations but only one of them is the global best solution).

Formulating the TSR problem as an optimization problem and defining a fitness function based on two objectives which are: (i) the execution cost of the reduced test suite and (ii) the effectiveness of the reduced test suite in terms of its mutation score. The variable weighted sum method [19] was utilized to form the multi-objective fitness functions, which guides the BBA to search for the non-dominated solutions that provides an optimum balance between the cost and effectiveness of the reduced test suites.

Evaluating the performance of the multi-objective ABBA (MO-ABBA) against each of the multi-objective BBA (MO-BBA) and the multi-objective BPSO (MO-BPSO) in solving the multi-objective TSR problem over eight test suites of different sizes.

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.

2 Preliminaries

2.1 Test Suite Reduction Problem

Given: A test suite TS which includes d test cases, and a set of mutants $\{mu_1, \ldots, mu_n\}$, 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.

2.2 Pareto Optimal Concepts

In the multi-objective optimization problems, there is no single solution but a set of multiple trade-offs solutions [25]. The vector of decision variables that optimizes the considered objective functions and satisfy the problem constraints is called a Pareto front. Thus, the Pareto front is a set of Pareto solutions which are not dominated by any other solution. A solution $x = [x_1, x_2, \ldots, x_n]$ is said to dominate a solution $y = [y_1, y_2, \ldots, y_n]$, if and only if y is not better than x for any objective $i = 1, 2, \ldots, n$, and there exists at least one objective x_i in x which is better than its corresponding objective y_i in y. On the contrary, two solutions are said to be non-dominated when none of them dominates the other. Fig. 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 $f1(A) < f1(D)$ and $f2(A) < f2(D)$. 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 for the objective $f1$. The set of non-dominated 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 satisfies 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.

2.3 Bat Algorithm

Bat algorithm (BA) is one of the recent metaheuristic swarm intelligence optimization algorithms which is proposed by Yang [18]. 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 randomly generating the initial population of bats. The values of the parameters of each bat b_i are updated over the iterations according to Eqs. (1)–(3).

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

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

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

where, Gbest is the current global best location (solution), 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. The bats perform a random walk procedure which is defined by Eq. (4) for exploring the space.

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

where, ϵ is a random number in the range $[-1, 1]$, 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 (A) and the pulse emission rate (r) which are updated according to Eqs. (5) and (6).

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

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

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

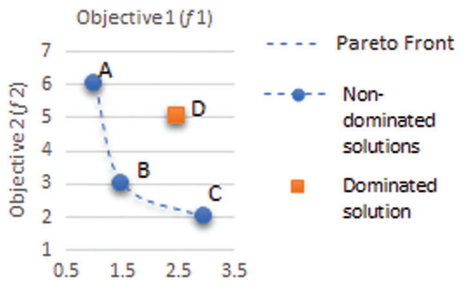

Figure 1: A sample representation of dominated non-dominated solutions and a Pareto front

Mirjalili et al. [26] proposed the BBA to solve optimization problems in the binary search space. In the BBA, the bat's position is changed from one to zero or vice versa based on the probability of the bat's velocity according to Eqs. (7) and (8).

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

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

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

3 Literature Review

Researchers proposed a significant number of approaches to minimize the size of the test suite. The majority of these approaches are based on greedy algorithms [27], while very few of them are based on clustering algorithms [28] or utilize hybrid algorithms (e.g., neuro fuzzy techniques) [29]. Greedy based approaches utilize one of the greedy algorithms to determine the reduced test suite based on the current best strategy. Over each iteration, the greedy algorithm adds to the reduced test suite the test case that has the highest greedy property, e.g., the highest statement coverage, which is a local optimal solution. It stops when the desired percentage of coverage is reached. Greedy approaches were proved empirically to be effective in solving the SO-TSR problem. 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. Recently, heuristic algorithms where utilized to solve the single and multi-objective TSR [8,10–15,17]. According to a survey study conducted by Khan et al. [8] the majority of the heuristic search based TSR approaches, 79% of them, are single-objective optimization. Some researchers showed empirically that some heuristic algorithms are superior to greedy algorithms in solving each of the single and multi-objective TSR problem [12,17]. The work of Yoo et al. [12] is considered as the first work that applies multi-objective optimization for test suite minimization. The authors used the NSGAII algorithm and showed experimentally its superiority over the greedy approaches. Geng et al. [10] and Gupta et al. [15] also utilized the NSGA-II algorithm but with different objective drivers. The objectives of Geng et al. [10] were the code coverage and test suite cost; while the objectives of Gupta et al. [15] were the code coverage and mutation score. Wang et al. [13] proposed utilizing three types of weighted-based genetic algorithms for minimizing the test suite of the product lines software. Where, the authors weighted summed the different objective drivers to form a single objective fitness function that guides the GA during the search for a Pareto front. Their experimental results showed that the Random-Weight GA algorithm outperforms seven other popular multi-objective search algorithms including: NSGA-II, strength Pareto evolutionary algorithms (SPEA) and speed-constrained multi-objective Particle Swarm Optimization (SMPSO). Wei et al. [14] compared among six evolutionary multi-objective optimization algorithms including NSGAII and several variants of the multi-objective 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 tried different combinations of objective drivers including mutation score, code coverage and test suite cost. The experiments showed that for the same statement coverage, using the "mutation score" as an objective driver guided the evolutionary algorithms to the smallest test suite.

Difference from the previous work

The approach proposed in this paper is a heuristic search-based approach. We adapted one of the recent heuristic algorithms BBA which proved its superiority over other evolutionary algorithms in different contexts [20]. However, the BBA occasionally fails to discover the global best solution for some multi modal functions; in addition, the BBA is used for solving single objective optimization problems. So, we proposed modifications to the BBA and utilized it to minimize the test suite size without loss in the fault detection quality. We used mutation testing to measure the quality of the reduced test suite, because mutation testing has been studied by numerous researchers as a method to assess the quality of a test suite [24,30–32]. Previous research proved empirically that mutation testing is more effective than code coverage in evaluating and comparing test suites [24].

4 Proposed Multi-Objective ABBA (MO-ABBA) for TSR

4.1 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 $X_i = (x_{i1}, x_{i2}, \ldots., x_{id})$, 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". This means that tc_j is included/excluded in the test suite, respectively.

4.2 Adapted Binary Bat Algorithm (ABBA)

Generally, the performance of any heuristic algorithm, including the BA, is affected by two crucial competencies 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. 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, $V_i(t+1)$ Eq. (1), includes two components. The first component is the previous velocity of the bat, $V_i(t)$, which is responsible for the global search (exploration). As, $V_i(t)$ directs the bat to keep its velocity and direction, thus it overflows the search space. While the second component, $(X_i(t) - \text{Gbest})F_i$, 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 $V_i(t+1)$ formula: Firstly, multiplying the term $V_i(t)$ 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 BAand the PSO [24,33].

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

where w_{max} and w_{min} are pre-determined constants, $iter$ 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 these 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:

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

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

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

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*.

4.3 TSR Multi-Objective Fitness Function Formulation

In this paper the fitness function is composed of two objectives. The first objective aims at minimizing the cost of the test suite; which is expressed in terms of the execution time as recommended by Yoo et al. [12]. While the second objective aims at selecting a reduced test suite that is capable of detecting the largest number of faults. The fault detection capability of the reduced test suite is expressed in terms of the mutation score. Wei et al. [14] found out that the mutation score is the most effective objective for solving the TSR problem. The two objectives are defined using Eqs. (13)–(14).

$$Objective1 = \frac{\sum_{|tc|} exec\_time_i}{\sum_{|RTS|} exec\_time_i} \tag{13}$$

$$Objective2 = \frac{|killed\ mu|}{|mu|} \tag{14}$$

where $|tc\,|$ is the size of the original test suite, $|\,RTS|$ is the size of the reduced test suite, $exec\_time_i$ is the execution time of a test case “i”, $|mu|$ is the total number of mutants of the software under test and $|killed\ mu|$ is the number of the killed mutants by the reduced test suite. The detection capability (number of killed mutants) of the reduced test suite is the cumulative sum of the detection capability of each tc_i in the reduced test suite; where each tc_i is represented using a binary vector of size n, n is the total number of mutants of a SUT. The value of a bit number j in tc_i vector is set to equal “1”/”0” if tc_i kill/do not kill the mutant number j.

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 objective functions. Yang [19] 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. [13] showed that the random 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 Eqs. (15)–(17), the best solution is the one that maximizes the *fitness*.

$$fitness = weight_1 * Objective1 + weight_2 * Objective2 \tag{15}$$

$$weight_2 = 1 + (weight_{init} - 1)\left(\frac{iter_{max} - iter}{iter_{max}}\right)^n \tag{16}$$

$$weight_1 = 1 - weight_2 \tag{17}$$

where, $weight_1$, $weight_2$ are the weights used to find the Pareto-optimal set of solutions, $weight_{init}$ denotes the initial value of $weight_2$, $iter$ is the current iteration number, and n is a modulation index. With the increase in the iteration the value of $weight_2$ increases from $weight_{init}$ to 1, whereas $weight_1$ 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. 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.

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

```
Define the objective functions $f_1(X), \ldots, f_k(X)$, $X = (x_1, x_2, \ldots, x_d)^T$
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 $V_i$ $(i = 1, 2, \ldots, N)$
    Specify the pulse frequencies $F_i$ at $X_i$
    Initialize loudness $A_i$ and pulse emission rate $r_i$
    Find the current $Gbest$ and $Rbest$
    $while(\ t \leq Maximum\ number\ of\ iterations)$
        Adjust $F_i$ and update $V_i$ $((i = 1, 2, \ldots, N))$ using Eqs. (3) and (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
```

5 Experiments and Results

5.1 Research Questions

The experiments were designed to answer the following research questions:

RQ1: Does the performance of the single-objective (SO-ABBA) surpasses the performance of the SO-BBA and the SO-BPSO?

RQ2: Does the performance of the MO-ABBA surpasses the performance of the MO-BBA and the MO-BPSO in solving the MO-TSR problem?

RQ3: How does the increase in the test suite size of the SUT and the number of mutants impact the performance of the MO-ABBA?

5.2 Data Sets

To answer RQ1, we used twelve unimodal and multimodal benchmark functions. Tab. 1 lists these functions along with their search boundaries (range). $fn_1 - fn_6$ are unimodal and $fn_7 - fn_{12}$ are multimodal benchmark functions. The global minimum values of all benchmark functions used are 0. In the experiments, 15 bits were used to represent each continuous variable in binary. Thus, the dimension of generating a bit vector for a benchmark function f_n was calculated by Eq. (18) as follows:

$$nb = D_{fn} \times 15 \tag{18}$$

where, nb is the dimension of the bats/particles.

Table 1: Benchmark functions

Function	Range	Function	Range
$fn_1(x) = \sum_{i=1}^{D} x_i^2$	[−100, 100]	$fn_7(x) = \sum_{i=1}^{D} ix_i^4 + random[0,1)$	[-1.28,1.28]
$fn_2(x) = \sum_{i=1}^{D} \lvert x_i^2 \rvert + \prod_{i=1}^{D} \lvert x_i^2 \rvert$	[−10, 10]	$fn_8(x) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i) + 10)$	[-5.12,5.12]
$fn_3(x) = \sum_{i=1}^{D} ix_i^2$	[−10, 10]	$fn_9(x) = \frac{1}{4000} \sum_{i=1}^{D} x_i^2 + \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[-600,600]
$fn_4(x) = \sum_{i=1}^{D} x_i^2 * (10^6)^{(i-1)/(D-1)}$	[−100, 100]	$fn_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\, 2\pi x_i\right)$	[-32,32]
$fn_5(x) = \sum_{i=1}^{D} (x_i + 0.5)^2$	[−100, 100]	$fn_{11}(x) = \sum_{i=1}^{D-1} \lvert x_i \sin(x_i) + 0.1\, x_i \rvert$	[-10,10]
$fn_6(x) = \sum_{i=1}^{D-1} [100\,(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	[−5, 10]	$fn_{12}(x) = -418.98*D + \sum_{i=1}^{D} [-x_i \sin(\sqrt{\lvert x_i \rvert})]$	[-500,500]

To answer RQ2 and RQ3, we used eight programs, out of which, two are C open-source programs and their characteristics retrieved from a popular repository, Software-artifact Infrastructure Repository (SIR) [34], which are: flex *v1* and make *v1*. While the other six java programs are from an available dataset(1), provided by Polo et al. [35]. The execution time of the test cases of these six programs is not available, so we assumed that all the test cases have equal execution time equal to 1 unit time. The characteristics of the eight programs are listed in Tab. 2, which are the line of code (LOC), the test suite size |tc|, number of mutants |mu|, and the execution time of the original test suite (T_{exec}).

Table 2: Experimental software under test (SUT)

	Flex_V1	Make_V1	MBisectOk	MBubCorrect	MFindOk	MFourBall	MMidOK	MTringle
LOC	15297	27879	31	54	79	47	59	61
|mu|	19	17	44	70	179	168	138	239
|tc|	567	1043	25	256	135	96	125	216
Texec	170.38	12070.23	25	256	135	96	125	216

(1) http://www.inf-cr.uclm.es/www/mpolo/stvr

5.3 Parameter Setting

We experimented with the most recommended values for the parameters in the literature [36]. Tab. 3 lists the parameter values that achieved the best performance.

Table 3: Parameter settings for ABBA, BBA, and BPSO

Parameter	ABBA	BBA	BPSO	Parameter	ABBA	BBA	BPSO
Pop. size	40	40	40	α , γ	0.9, 0.9	0.9, 0.9	–
Max iteration	100	100	100	w_{max}, w_{min}	0.8, 0.4	–	–
F_{min}, F_{max}	0,2	0,2	–	n	2	–	–
A, r	0.9, 0.1	0.9, 0.1	–	δ_{init}	0.6	–	–
ϵ	[−1, 1]	–	–	C_1, C_2	–	–	1.5, 1.2
Max velocity	–	–	6				

5.4 Performance Metrics

Two metrics were used to assess the performance of the heuristic algorithms in general which are the fitness *Mean* and standard deviation (SD), defined by Eqs. (19) and (20).

$$Mean = \left(\sum_{i=1}^{N} f_i\right) / N \tag{19}$$

$$SD = \sqrt{\frac{\sum_{i=1}^{N} (f_i - mean)^2}{N}} \tag{20}$$

where, N is the number of runs, f_i is the fitness of the best solution discovered during the run number *i*.

As the evolutionary algorithms are stochastic, for each experiment, 10 independent runs were performed; then the *Mean* and *SD* are calculated. Larger *mean* values indicate better solutions. While 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.

Extra three specific metrics were used which are: (i) Test suite size reduction rate (TSRR), (ii) Execution time reduction rate (*ETR*) and (iii) Fault detection capability rate (*FDR*); They are calculated using Eqs. (21)–(23)

$$TSRR = \frac{|tc| - |RTS|}{|tc|} \tag{21}$$

$$ETR = \frac{\sum_{|tc|} exec\_time_i - \sum_{|RTS|} exec\_time_i}{\sum_{|tc|} exec\_time_i} \tag{22}$$

$$FDR = \frac{|Killed\ mu|}{|mu|} \tag{23}$$

The higher the values of the TSRR, ETR and FDR, the better the performance of the search algorithm.

6 Results

Answer to RQ1:

Tab. 4 lists the mean and SD of the optimal solutions discovered for each function over the ten runs, also lists the mean and SD of the number of iterations executed to reach the corresponding optimal solutions; the best results are pointed out in bold style. The maximum number of iterations was set to equal 1000 across all the experiments; however, the best solutions were achieved earlier than the predetermined maximum number of iterations. It should be noted that, for any of the previously mentioned benchmark functions, the best solution is the one that has the smallest mean value (minimization problem). Also, the smaller the mean value of the executed number of iterations, the faster the convergence speed of the heuristic algorithm.

Table 4: Performance comparison among SO-ABBA, SO-BBA, SO-PSO over benchmark functions

fn	Optimal solution (Mean ± Std. dev)			Convergence speed (# of Iterations Mean ± Std. dev)		
	SO-ABBA	**SO-BBA**	**SO-BPSO**	**SO-ABBA**	**SO-BBA**	**SO-BPSO**
fn₁	**0 ± 0**	1 ± 0.81	**0** ± 0	**34** ± 7.14	339 ± 320.73	502 ± 25.36
fn₂	**0 ± 0**	1.5 ± 1.26	**0** ± 0	**32** ± 8.93	384 ± 244.02	615 ± 22.94
fn₃	**0 ± 0**	12.4 ± 13.13	7 ± 13.32	**59** ± 31.22	470 ± 295.46	622 ± 19.91
fn₄	**0 ± 0**	72.23 ± 132.40	78.44 ± 121.06	**97** ± 30.72	308 ± 302.75	667 ± 27.84
fn₅	**0 ± 0**	1.1 ± 132.40	**0** ± 0	**33** ± 53.30	620 ± 191.12	365 ± 21.19
fn₆	**0 ± 0**	4.675e2 ± 109.29	0.1 ± 0.32	**98** ± 53.16	747 ± 201.36	608 ± 25.04
fn₇	**0.0002** ± 0.0001	16.20 ± 24.02	4.100 ± 6.54	**180** ± 269.58	468 ± 274.45	656 ± 58.99
fn₈	**0 ± 0**	1.3 ± 1.25	**0** ± 0	**29** ± 10.46	488 ± 342.19	611 ± 21.59
fn₉	**0 ± 0**	0.011 ± 0.01	0.003 ± 0.01	**60** ± 38.24	402 ± 177.85	624 ± 23.41
fn₁₀	**8.8818e-16** ± 0	0.38 ± 0.21	**8.8818e-16** ± 0	**30** ± 10.23	409 ± 292.67	611 ± 29.13
fn₁₁	**0** ± 0	1.03 ± 0.82	**0** ± 0	**28** ± 13.65	424 ± 215.54	600 ± 220.89
fn₁₂	**31360.61 ± 0**	31361.28 ± 0.66	31360.69 ± 0. 66	**30** ± 14.01	244 ± 220.89	588 ± 12.51

As could be observed from Tab. 4 that the performance of the SO-ABBA algorithm is superior to the SO-BBA over all the unimodal and multimodal benchmark functions; as SO-ABBA could discover better solutions than the ones discovered by the SO-BBA, in terms of the mean values of the best solutions. In addition to, the SD values of the best solutions in case of using the SO-ABBA are smaller than when using SO-BBA over all the functions, which indicates that the SO-ABBA is more robust than the SO-BBA. Small SD values of the optimal solutions discovered by the SO-ABBA prove that the SO-ABBA is efficient in finding the best solutions without large variance among the different runs. When comparing the performance of the SO-ABBA to the SO-BPSO, it was found that the SO-ABBA was capable of discovering better solutions than the SO-BPSO for fn_3, fn_4, fn_6, fn_7, fn_9, fn_{12} functions. In addition, the SD values of the best solutions in the case of using the SO-ABBA are smaller than when using the SO-BPSO. On the other hand, both the SO-ABBA and the SO-BPSO could discover the same best solutions for fn_1, fn_2, fn_5, fn_8, fn_{10}, fn_{11}. As could be observed from the mean values of the number of iterations executed by each of the three algorithms to discover the optimum solutions that, the SO-ABBA could converge to the best solutions much faster than each of the SO-BBA and the SO-BPSO.

Answer to RQ2:

To answer RQ2 we conducted a set of experiments over the previously mentioned 8 programs. The results listed in the paper are the mean and SD of 10 independent runs over each program. The maximum number of iterations was set to equal 100 in all the experiments. Tab. 5 lists the generated values of weight1 and weight2 which were used to calculate 10 non-dominated solutions (NDS) on the Pareto surface. While Tab. 6 lists the mean and SD values of each of the fitness (F) and convergence speed of the discovered NDS. The convergence speed is measured in terms of the number of iterations executed by each algorithm to discover the best solution (#i). To simplify the visualization of the results, only three NDS (nu. 1, 5, 10) were listed in the table. Tab. 7 lists the FDR, TSRR and ETR of the three selected NDS.

Table 5: Weights used in experiments to generate 10 NDS

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	0.004

Table 6: Comparison among the fitness values (F) and the convergence speed (#i) of three NDS discovered by the MO-ABBA, MO-BBA AND MO-BPSO

NDS #			MEAN ± STD. DEV							
			Flex_V1	Make_V1	MBisectOk	MBubCorrect	MFindOk	MFourBall	MMidOK	MTringle
1	ABBA	F	**2.93 ± 0.06**	**4.06 ± 0.08**	**10.57 ± 0**	**8.34 ± 1.46**	**54.60 ± 0**	**38.62 ± 0.02**	**8.34 ± 1.46**	**11.53 ± 3.25**
		#i	95.5 ± 4.95	**98.5 ± 2.12**	**15.33 ± 8.09**	**91.60 ± 6.28**	**68.50 ± 12.25**	**60.20 ± 23.06**	**69.70 ± 19.30**	**94.22 ± 5.95**
	BBA	F	2.03 ± 0.05	2.81 ± 0.03	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	**94.5 ± 1.77**	99.5 ± 0.35	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	F	1.92 ± 0.05	2.68 ± 0.06	**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	97.5 ± 1.1	**98.5 ± 0.35**	67.60 ± 22.08	98.40 ± 1.58	98.70 ± 1.06	98.60 ± 1.96	99.50 ± 0.85	99.40 ± 0.97
5	ABBA	F	**1.53 ± 0.02**	**2.12 ± 0.0**	**4.42 ± 0**	**3.70 ± 0.61**	**20.13 ± 0.35**	**14.14 ± 0.02**	**3.70 ± 0.61**	**3.82 ± 0.72**
		#i	**96.5 ± 3.54**	**97.5 ± 0.71**	**20.44 ± 9.88**	**92.70 ± 7.63**	**69.20 ± 11.13**	**55.70 ± 23.31**	**67.40 ± 21.57**	**93.78 ± 5.80**
	BBA	F	1.25 ± 0.00	1.76 ± 0.0	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	98.5 ± 1.06	98 ± 1.41	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	F	1.23 ± 0.01	1.66 ± 0.02	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	100 ± 0	100 ± 0	81.80 ± 9.11	99.80 ± 0.42	99.00 ± 0.94	99.00 ± 1.56	99.50 ± 1.08	99.10 ± 0.74
10	ABBA	F	**0.86 ± 0.0**	**1.03 ± 0.0**	**1.05 ± 0**	**1.08 ± 0.02**	**1.54 ± 0**	**1.08 ± 0.01**	**1.08 ± 0.02**	**1.04 ± 0**
		#i	**92 ± 5.66**	100 ± 0	42.22 ± 30.97	92.50 ± 6.08	**64.30 ± 13.65**	**54.30 ± 21.68**	**76.50 ± 14.16**	**88.56 ± 11.28**
	BBA	F	0.85 ± 0.0	1.02 ± 0.0	1.04 ± 0.01	1.02 ± 0	1.03 ± 0.01	1.04 ± 0.01	1.03 ± 0.01	1.01 ± 0
		#i	98 ± 0.7	**99 ± 0.71**	**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	F	0.85 ± 0.0	1.02 ± 0.0	**1.05 ± 0.01**	1.02 ± 0	1.03 ± 0.01	1.05 ± 0.01	1.03 ± 0.01	1.01 ± 0
		#i	99.5 ± 0.35	99.5 ± 0.35	71.90 ± 16.63	99.40 ± 0.84	98.40 ± 2.22	99.50 ± 0.71	99.40 ± 0.70	99.20 ± 0.92

As could be observed from Tab. 6, that the MO-ABBA algorithm surpasses both of the MO-BPSO and MO-BBA in terms of the mean fitness values of the discovered NDS, across the 8 programs. Moreover, the fitness standard deviation values of the NDS discovered by the MO-ABBA are very small; most of them are equal to zero or approaches zero, which indicates the stability of the MO-ABBA. E.g., the fitness mean and SD values of the NDS#5 discovered by the MO-ABBA, MO-BBA and MO-BPSO are as follows: Flex_V1

(**1.53 ± 0.02**, 1.25 ± 0.00, 1.23 ± 0.01), Make_V1 (**2.12 ± 0.0**, 1.76 ± 0.0, 1.66 ± 0.02), MBisectOk (**4.42 ± 0**, 4.31 ± 0.13, 4.41 ± 0.01), MBubCorrect (**3.70 ± 0.61**, 1.61 ± 0.06, 1.61 ± 0.07), MFindOk (**20.13 ± 0.35**, 2.13 ± 0.24, 2.45 ± 0.33), MFourBall (**14.14 ± 0.02**, 3.84 ± 3.64, 3.25 ± 0.64), MMidOK (**3.70 ± 0.61**, 2.34 ± 0.25, 2.48 ± 0.23), MTringle (**3.82 ± 0.72**, 1.58 ± 0.07, 1.58 ± 0.09).

In terms of the convergence speed, the MO-ABBA was able to converge to the best solutions faster than each of the MO-BBA and the MO-BPSO across all the programs except Flex_V1, NDS 1(#i = 95.5 ± 4.95 ABBA, **94.5 ± 1.77 BBA**, 97.5 ± 1.1 BPSO), Make_V1(NDS 10) (#i = 100 ± 0 ABBA, **99 ± 0.71 BBA**, 99.5 ± 0.35 BPSO), MBisectOk (NDS 10) (#i = 42.22 ± 30.97 ABBA, **21.90 ± 18.78 BBA**, 71.90 ± 16.63 BPSO) and MbubCorrect (NDS 10) (#i = 92.5 ± 6.08 ABBA, **89.40 ± 8.37 BBA**, 99.4 ± 10.84 BPSO); where, the MO-BBA converged faster than the MO-ABBA. Nevertheless, the MO-ABBA converged to better solutions than the ones discovered by the MO-BBA. So It could be stated that the MO-BBA was prematurely converged in these cases. Furthermore, the MO-BPSO was occasionally able to discover solutions close to the ones discovered by the MO-ABBA, e.g., MBisectOk NDS 1 (F = 10.57 ± 0 ABBA, 10.57 ± 0.01 BPSO) and MBisectOk NDS 10 (F = 1.05 ± 0 ABBA, 1.05 ± 0.01 BPSO). But the MO-BPSO needed a greater number of iterations than the MO-ABBA to converge to these solutions (MBisectOk NDS 1 #i = **15.33 ± 8.09** ABBA, 67.60 ± 22.08 BPSO ; MBisectOk NDS 10 #i = **42.22 ± 30.97**ABBA, 71.90 ± 16.63 BPSO).

As could be observed from Tab. 7 that some of the solutions discovered by the three algorithms have the same FDR values, but the solutions discovered by the MO-ABBA have the highest TSRR and ETR values (e.g., Flex_V1 NDS#1 TSRR = (**82.19 ± 2.12** ABBA, 72.40 ± 4.60 BBA, 70.46 ± 7.42 BPSO) ; ETR = (**83.50 ± 0.65** ABBA, 73.78 ± 1.53 BBA, 71.71 ± 1.88 BPSO), Make NDS#1 TSRR = (**68.41 ± 1.77** ABBA, 60.55 ± 0.35 BBA, 59.16 ± 0.71BPSO); ETR = (80.77 ± 69.0 ABBA, 81.89 ± 34.27 BBA, **81.98 ± 46.38** BPSO) and MFindOk NDS#1 TSRR = (**99.3 ± 0** ABBA, 88.7 ± 3.80 BBA, 89.6 ± 1.78 BPSO); ETR = (**99.3 ± 0** ABBA, 88.7 ± 3.80 BBA, 89.6 ± 1.78 BPSO)). Which means that for the same fault detection capability rate the MO-ABBA could select from a test suite a subset of test cases with smaller size and faster execution time (less expensive) than the subsets selected by each of the MO-BBA and MO-BPSO. While, some of the solutions discovered by the MO-ABBA are more cost effective (in terms of the TSRR and the ETR values) but with less fault detection capabilities (e.g., MFourBall NDS #1 FDR = (37.4 ± 4.02 ABBA, **83.4 ± 20.99 BBA**, 61.7 ± 15.33BPSO); TSRR = (**99.0 ± 0** ABBA, 91.8 ± 2.42 BBA, 96.4 ± 1.49 BPSO); ETR = (**99.0 ± 0** ABBA, 91.8 ± 2.42 BBA,96.4 ± 1.49 BPSO) ; MFourBall NDS #2 FDR = (37.1 ± 4.06 ABBA, **97.6 ± 40.20** BBA, 85.7 ± 25.01 BPSO); TSRR = (**99.0 ± 0** ABBA, 92.5 ± 3.05 BBA, 93.8 ± 1.94 BPSO); ETR = (**99.0 ± 0** ABBA, 92.5 ± 3.05 BBA,93.8 ± 1.94 BPSO) ; MMidOk NDS #1; MTringle NDS #1 and MTringle NDS #2. So, the selection of the best algorithm depends on the testers' targets and testing budgets. Fig. 2 shows sample convergence curves of the three algorithms over the programs. As could be observed that the MO-ABBA converge faster and to better solutions than the MO-BBA and the MO-BPSO, although the parameters settings of both of the MO-ABBA and the MO-BBA are the same.

Table 7: Comparison among three NDS in terms of fault detection capability rate (FDR), the reduction percentage of test suite (TSRR) and the execution time reduction (ETR)

NDS			MEAN ± STD. DEV							
			Flex	Make	MBisectOk	MBubCorrect	MFindOk	MFourBall	MMidOK	MTringle
1	ABBA	FDR	84.21 ± 0	100 ± 0	**95.45 ± 0**	96.3 ± 0.32	**99.4 ± 0**	37.4 ± 4.02	41.6 ± 11.06	73.7 ± 30.08
		TSRR	**82.19 ± 2.12**	**68.41 ± 1.77**	**96.0 ± 0**	**94.6 ± 2.70**	**99.3 ± 0**	**99.0 ± 0**	**99.1 ± 0.31**	**96.1 ± 2.32**
		ETR	**83.50 ± 0.65**	80.77 ± 69.00	**96.0 ± 0**	**94.6 ± 2.70**	**99.3 ± 0**	**99.0 ± 0**	**99.1 ± 0.31**	**96.1 ± 2.32**
	BBA	FDR	84.21 ± 0	100 ± 0	86.4 ± 4.09	**97.1 ± 0**	99.4 ± 0	**83.4 ± 20.99**	**94.7 ± 8.62**	87.7 ± 16.29
		TSRP	72.40 ± 4.60	60.55 ± 0.35	96.0 ± 0	81.0 ± 5.19	88.7 ± 3.80	91.8 ± 2.42	89.9 ± 2.41	83.6 ± 3.66
		ETR	73.78 ± 1.53	81.89 ± 34.27	96.0 ± 0	81.0 ± 5.19	88.7 ± 3.80	91.8 ± 2.42	89.9 ± 2.41	83.6 ± 3.66

(Continued)

Table 7 (continued).

NDS			MEAN ± STD. DEV							
			Flex	Make	MBisectOk	MBubCorrect	MFindOk	MFourBall	MMidOK	MTringle
	BPSO	FDR	84.21 ± 0	100 ± 0	95.0 ± 0.6	**97.1 ± 0**	99.4 ± 0	61.7 ± 15.33	94.4 ± 6.7	**87.9 ± 8.31**
		TSRP	70.46 ± 7.42	59.16 ± 0.71	96.0 ± 0	80.9 ± 4.14	89.6 ± 1.78	96.4 ± 1.49	91.9 ± 1.52	83.1 ± 3.02
		ETR	71.71 ± 1.88	**81.98 ± 46.38**	96.0 ± 0	80.9 ± 4.14	89.6 ± 1.78	96.4 ± 1.49	91.9 ± 1.52	83.1 ± 3.02
5	ABBA	FDR	84.21 ± 0	100 ± 0	**95.45 ± 0**	96.9 ± 0.42	**79.9 ± 73.58**	37.1 ± 4.06	41.2 ± 5.47	75.0 ± 29.21
		TSRP	**81.04 ± 2.47**	**69.32 ± 3.53**	**96.0 ± 0**	**94.6 ± 2.34**	**99.3 ± 0**	**99.0 ± 0**	**99.2 ± 0**	**95.2 ± 2.31**
		ETR	**82.14 ± 0.79**	**88.28 ± 17.66**	**96.0 ± 0**	**94.6 ± 2.34**	**99.3 ± 0**	**99.0 ± 0**	**99.2 ± 0**	**95.2 ± 2.31**
	BBA	FDR	84.21 ± 0	100 ± 0	83.2 ± 6.35	**97.1 ± 0**	99.4 ± 0	**97.6 ± 40.20**	**95.8 ± 0.32**	**97.7 ± 1.26**
		TSRP	72.22 ± 1.77	60.93 ± 2.47	96.0 ± 0	81.0 ± 3.68	88.4 ± 2.90	92.5 ± 3.05	90.3 ± 2.33	80.5 ± 3.81
		ETR	73.04 ± 0.50	82.20 ± 23.57	96.0 ± 0	81.0 ± 3.68	88.4 ± 2.90	92.5 ± 3.05	90.3 ± 2.33	80.5 ± 3.81
	BPSO	FDR	84.21 ± 0	100 ± 0	95.0 ± 0.6	**97.1 ± 0**	99.4 ± 0	85.7 ± 25.01	94.7 ± 8.88	96.7 ± 2.35
		TSRP	70.99 ± 3.89	58.15 ± 3.18	96.0 ± 0	80.5 ± 4.27	90.6 ± 2.71	93.8 ± 1.94	91.3 ± 1.59	80.6 ± 4.89
		ETR	71.72 ± 1.24	80.44 ± 5.72	96.0 ± 0	80.5 ± 4.27	90.6 ± 2.71	93.8 ± 1.94	91.3 ± 1.59	80.6 ± 4.89
10	ABBA	FDR	84.21 ± 0	100 ± 0	95.0 ± 1.05	**97.1 ± 0**	**99.4 ± 0**	99.3 ± 0.42	99.1 ± 1.06	**100 ± 0**
		TSRP	**82.1 ± 1.06**	**67.79 ± 4.24**	**95.6 ± 0.3**	**94.9 ± 3.38**	**99.3 ± 0**	**95.5 ± 0.42**	**96.0 ± 0.67**	**90.5 ± 1.51**
		ETR	**83.31 ± 0.24**	**88.28 ± 17.65**	**95.6 ± 0.3**	**94.9 ± 3.38**	**99.3 ± 0**	**95.5 ± 0.42**	**96.0 ± 0.67**	**90.5 ± 1.51**
	BBA	FDR	84.21 ± 0	100 ± 0	**99.5 ± 0.6**	97.1 ± 0	99.4 ± 0	**99.9 ± 0.6**	**100.0 ± 0**	100 ± 0.0
		TSRP	73.02 ± 2.12	61.55 ± 4.24	89.6 ± 0.8	81.5 ± 4.43	87.9 ± 3.17	90.5 ± 1.52	89.0 ± 3.09	78.7 ± 4.16
		ETR	74.37 ± 0.23	81.90 ± 5.60	89.6 ± 0.8	81.5 ± 4.43	87.9 ± 3.17	90.5 ± 1.52	89.0 ± 3.09	78.7 ± 4.16
	BPSO	FDR	84.21 ± 0	100 ± 0	97.3 ± 0.68	97.1 ± 0	99.4 ± 0	99.8 ± 0.97	99.9 ± 0.32	100 ± 0
		TSRP	71.34 ± 2.47	57.05 ± 4.24	93.6 ± 0.8	80.8 ± 4.87	89.3 ± 2.06	91.9 ± 1.93	89.4 ± 2.15	75 ± 4.25
		ETR	72.37 ± 0.91	80.44 ± 5.72	93.6 ± 0.8	80.8 ± 4.87	89.3 ± 2.06	91.9 ± 1.93	89.4 ± 2.15	75 ± 4.25

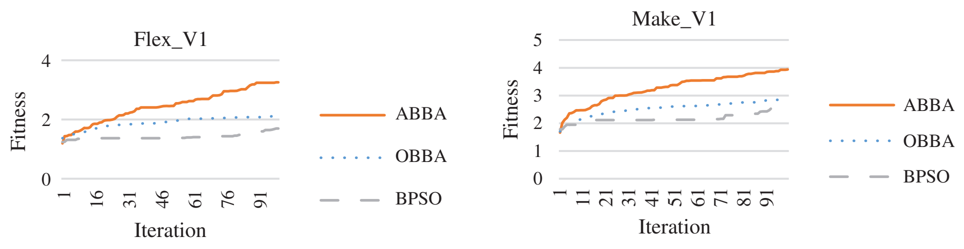

Figure 2: Sample convergence curves of the MO-ABBA, MO-BBA and MO-BPSO

Answer to RQ3:

As could be observed from the set of experiments over the benchmark functions that the performance of the SO-ABBA was not affected by the functions' ranges or types. For example, the SO-ABBA was superior to SO-BBA and SO-BPSO over unimodal functions fn_2 and fn_4, but fn_2 has a small range, while fn_4 has a wide range. SO-ABBA has the same superior performance over the multi-model functions fn_9 and fn_{11}. Moreover, the performance of the MO-ABBA was superior over the 8 programs with different test suites sizes (ranges from 25 to 1034), different execution times (ranges from 170.38 to 12070.23) and different numbers of mutants (ranges from 17 to 239). From these observations we could conclude that the ABBA is scalable.

7 Conclusion and Future Work

This paper proposed solving the multi-objective test suite reduction problem using the Binary Bat algorithm, which is reported in the literature as one of the effective swarm intelligence based algorithms. The BBA algorithm was adapted for better exploration capabilities and consequently better performance. The TSR problem was formulated as a multi-objective optimization problem. The adapted binary bat algorithm was utilized to search for the non-dominated solutions that keep the balance between the cost of the reduced test suites and their fault detection capabilities. The effectiveness of the adapted binary bat algorithm was assessed over eight programs of different test suites sizes, in addition to a set of unimodal and multi modal benchmark functions. The experimental results showed that the performance of the proposed MO-ABBA is superior to each of the MO-BBA and MO-BPSO in terms of the previously defined five metrics. Moreover, The MO-ABBA converged to the best solutions faster than each of the MO-BBA and the MO-BPSO. As a further extension for this work, different weighting mechanisms could be tried for the weighted sum multi-objective optimization. In addition, the fitness function could be redefined to include more objectives such as branch coverage. Furthermore, different inertia formulas [24] could be experimented with.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] C. Catal and D. Mishra, “Test case prioritization: A systematic mapping study,” *Software Quality Journal*, vol. 21, no. 3, pp. 445–478, 2013.

[2] R. H. Rosero, O. S. Gómez and G. Rodríguez, “15 years of software regression testing techniques—A survey,” *International Journal of Software Engineering Knowledge Engineering*, vol. 26, no. 5, pp. 675–689, 2016.

[3] G. Rothermel and M. J. Harrold, “Experience with regression test selection,” *Empirical Software Engineering*, vol. 2, no. 2, pp. 178–188, 1997.

[4] A. Nadeem and A. Awais, “TestFilter: A statement-coverage based test case reduction technique,” in *2006 IEEE Int. Multitopic Conf.*, Islamabad, Pakistan, IEEE, pp. 275–280, 2006.

[5] A. Gotlieb and D. Marijan, “FLOWER: Optimal test suite reduction as a network maximum flow,” in *Proc. of the 2014 Int. Symp. on Software Testing and Analysis*, San Jose, CA, USA, ACM, pp. 171–180, 2014.

[6] Y. Jia and M. Harman, “An analysis and survey of the development of mutation testing,” *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[7] T. Y. Chen and M. F. Lau, “A simulation study on some heuristics for test suite reduction,” *Information Software Technology*, vol. 40, no. 13, pp. 777–787, 1998.

[8] S. U. R. Khan, S. P. Lee, N. Javaid and W. Abdul, “A systematic review on test suite reduction: Approaches, experiment’s quality evaluation, and guidelines,” *IEEE Access*, vol. 6, pp. 11816–11841, 2018.

[9] R. Pan, Z. Zhang, X. Li, K. Chakrabarty and X. Gu, “Black-box test-coverage analysis and test-cost reduction based on a Bayesian network model,” in *2019 IEEE 37th VLSI Test Symp.*, Monterey, CA, USA, IEEE, pp. 1–6, 2019.

[10] J. Geng, Z. Li, R. Zhao and J. Guo, “Search based test suite minimization for fault detection and localization: A co-driven method,” in *Int. Sym. on Search Based Software Engineering*, Raleigh, USA, Springer, pp. 34–48, 2016.

[11] S. Mohanty, S. K. Mohapatra and S. F. Meko, “Ant colony optimization (ACO-Min) algorithm for test suite minimization,” in *Progress in Computing, Analytics and Networking*. Berlin, Germany: Springer, pp. 55–63, 2020.

[12] S. Yoo and M. Harman, “Pareto efficient multi-objective test case selection,” in *Proc. of the 2007 Int. Symp. on Software Testing and Analysis*, New York, NY, USA, pp. 140–150, 2007.

[13] S. Wang, S. Ali and A. Gotlieb, “Cost-effective test suite minimization in product lines using search techniques,” *Journal of Systems Software*, vol. 103, pp. 370–391, 2015.

[14] Z. Wei, W. Xiaoxue, Y. Xibing, C. Shichao, L. Wenxin *et al.,* “Test suite minimization with mutation testing-based many-objective evolutionary optimization,” in *2017 Int. Conf. on Software Analysis, Testing and Evolution*, Harbin, China, IEEE, pp. 30–36, 2017.

[15] N. Gupta, A. Sharma and M. K. Pachariya, “Multi-objective test suite optimization for detection and localization of software faults,” *Journal of King Saud University-Computer Information Sciences*, vol. 82, no. 11, pp. 1780, 2020.

[16] S. K. Mohapatra, A. K. Mishra and S. Prasad, “Intelligent local search for test case minimization,” *Journal of The Institution of Engineers: Series B*, vol. 101, no. 5, pp. 1–11, 2020.

[17] J.-Z. Sun and S.-Y. Wang, “A novel chaos discrete particle swarm optimization algorithm for test suite reduction,” in *The 2nd Int. Conf. on Information Science and Engineering*, Hangzhou, China, IEEE, pp. 1–4, 2010.

[18] X.-S. Yang, A new metaheuristic bat-inspired algorithm. in *Nature Inspired Cooperative Strategies for Optimization*. Berlin, Germany: Springer, pp. 65–74, 2010.

[19] X.-S. Yang, “Bat algorithm for multi-objective optimisation,” *International Journal of Bio-Inspired Computation*, vol. 3, no. 5, pp. 267–274, 2011.

[20] M. Chawla and M. Duhan, “Bat algorithm: A survey of the state-of-the-art,” *Applied Artificial Intelligence*, vol. 29, no. 6, pp. 617–634, 2015.

[21] A. Hamdy, “Genetic fuzzy system for enhancing software estimation models,” *International Journal of Modeling Optimization*, vol. 4, no. 3, pp. 227, 2014.

[22] A. Hamdy and A. Mohamed, “Greedy binary particle swarm optimization for multi-objective constrained next release problem,” *International Journal of Machine Learning Computing*, vol. 9, no. 5, pp. 561–568, 2019.

[23] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *Proc. of the Sixth Int. Symp. on Micro Machine and Human Science*, Nagoya, Japan, IEEE, pp. 39–43, 1995.

[24] J. C. Bansal, P. Singh, M. Saraswat, A. Verma, S. S. Jadon *et al.,* “Inertia weight strategies in particle swarm optimization,” in *2011 Third World Congress on Nature and Biologically Inspired Computing*, Salamanca, Spain, IEEE, pp. 633–640, 2011.

[25] P. Ngatchou, A. Zarei and A. El-Sharkawi, “Pareto multi objective optimization,” in *Proc. of the 13th Int. Conf. on, Intelligent Systems Application to Power Systems*, Arlington, VA, USA, IEEE, pp. 84–91, 2005.

[26] S. Mirjalili, S. M. Mirjalili and X.-S. Yang, “Binary bat algorithm,” *Neural Computing Applications*, vol. 25, no. 3–4, pp. 663–681, 2014.

[27] R. A. Assi, W. Masri and C. Trad, “Substate profiling for effective test suite reduction,” in *2018 IEEE 29th Int. Symp. on Software Reliability Engineering*, Memphis, TN, USA, IEEE, pp. 123–134, 2018.

[28] C. Coviello, S. Romano, G. Scanniello, A. Marchetto, G. Antoniol *et al.,* “Clustering support for inadequate test suite reduction,” in *2018 IEEE 25th Int. Conf. on Software Analysis, Evolution and Reengineering*, Campobasso, Italy, IEEE, pp. 95–105, 2018.

[29] Z. Anwar, H. Afzal, N. Bibi, H. Abbas, A. Mohsin *et al.,* “A hybrid-adaptive neuro-fuzzy inference system for multi-objective regression test suites optimization,” *Neural Computing Applications*, vol. 31, no. 11, pp. 7287–7301, 2019.

[30] P. G. Frankl, S. N. Weiss and C. Hu, “All-uses vs mutation testing: An experimental comparison of effectiveness,” *Journal of Systems Software*, vol. 38, no. 3, pp. 235–253, 1997.

[31] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, C. Zapf *et al.,* “An experimental determination of sufficient mutant operators,” *ACM Transactions on Software Engineering*, vol. 5, no. 2, pp. 99–118, 1996.

[32] J. H. Andrews, L. C. Briand, Y. Labiche and A. S. Namin, “Using mutation analysis for assessing and comparing testing coverage criteria,” *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 608–624, 2006.

[33] J. Xin, G. Chen and Y. Hai, “A particle swarm optimizer with multi-stage linearly-decreasing inertia weight,” in *2009 Int. Joint Conf. on Computational Sciences and Optimization*, Sanya, China, vol. 1, pp. 505–508, 2009.

[34] H. Do, S. Elbaum and G. Rothermel, “Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact,” *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[35] M. Polo, M. Piattini and I. García-Rodríguez, “Decreasing the cost of mutation testing with second-order mutants,” *Software Testing, Verification and Reliability*, vol. 19, no. 2, pp. 111–131, 2009.

[36] I. A. Carvalho, D. G. da Rocha, J. G. R. Silva, V. da Fonseca Vieira and C. R. Xavier, “Study of parameter sensitivity on bat algorithm,” in *Int. Conf. on Computational Science and Its Applications*, Trieste, Italy, Springer, pp. 494–508, 2017.