Neural Processing Letters
https://doi.org/10.1007/s11063-020-10288-7

An Improved Crow Search Algorithm for Test Data Generation Using Search-Based Mutation Testing

Nishtha Jatana[1,2] · Bharti Suri[1]

Abstract
Automation of test data generation is of prime importance in software testing because of the high cost and time incurred in manual testing. This paper proposes an Improved Crow Search Algorithm (ICSA) to automate the generation of test suites using the concept of mutation testing by simulating the intelligent behaviour of crows and Cauchy distribution. The Crow Search Algorithm suffers from the problem of search solutions getting trapped into the local search. The ICSA attempts to enhance the exploration capabilities of the metaheuristic algorithm by utilizing the concept of Cauchy random number. The concept of Mutation Sensitivity Testing has been used for defining the fitness function for the search based approach. The fitness function used, aids in finding optimal test suite which can achieve high detection score for the Program Under Test. The empirical evaluation of the proposed approach with other popular meta-heuristics, prove the effectiveness of ICSA for test suite generation using the concepts of mutation testing.

Keywords Improved Crow Search Algorithm · Cauchy random number · Mutation sensitivity testing · Mothra mutation operators

1 Introduction

Evaluation of quality of the software is a significant area of research in the field of software engineering. The correctness of a software application is evaluated using the concepts of software testing. Software testing involves the execution of the software application or a product with a set of test cases with the intention of finding bugs. The debugging process is carried out when the testing process reveals changes in the output which are identified as bugs [1, 2]. A test case can be defined as a set of inputs, and execution preconditions developed to verify whether a specific requirement is catered to or not [3]. The set of test cases which are capable of satisfying the testing objective is known as the effective test suite [4].

✉ Nishtha Jatana
nishtha.jatana@gmail.com

[1] University School of Information, Communication and Technology, Guru Gobind Singh Indraprastha University, New Delhi, Delhi 110078, India

[2] Maharaja Surajmal Institute of Technology, New Delhi, India

Mutation testing is one of the popular methods used in the software industry to verify the quality of the software under test [5]. In mutation testing, faults are added to the original program to create mutated code (or mutants). The faults are the mistakes that a programmer may commit. The process is carried out by applying the same test cases to both the original and mutated code with an aim of identifying the mutants [6]. If the test cases find a difference between the original and the mutated program, the mutant is said to be killed or detected. On the contrary, when the test case is unable to find the difference, the mutants are said to be alive [7]. The mutants generated can be too many in number and many of these can be equivalent or can be trivially killed [8]. The focus of mutation testing based test data generation is to find a test suite that is able to kill the maximum number of non-equivalent mutants. Generally, the number of killed mutants is evaluated with the term known as mutation score, which gives the ratio of mutants killed by the test cases to the total number of mutants [5]. Mutation testing can be employed in software testing at various levels, namely unit level, specification and integration level. Mutation testing has been applied to various software languages, like Fortran, C, Java, C#, SQL and AspectJ [5, 7]. The mutation operators for various languages have gradually evolved during the research on mutation testing. The five-operator set, proposed by Offutt et al. [9] provided the basic set of mutation operators. They include relational, arithmetic, logical, absolute and unary insertion operators. The Mothra mutation system consists of 22 mutation operators developed for the FORTRAN language by the members of the Georgia Institute of Technology's Software Engineering Research Centre [10]. With the evolution in technology, several tools for mutation testing have been evolved [5]. The tools like MuJava and PIT mutate the code with various types of mutation operators [11].

The manual generation of test cases is a complex as well as a costly task. The automatic test case generation is considered as an alternative to manual test case generation, since it can reduce the cost as well as time involved in the process [12–15].

Search based software engineering attempts to optimise the tasks associated with the process of software engineering by making use of the metaheuristic techniques. These techniques generate optimal solutions for the problems having huge search space. The fitness function is used to define the objective to be achieved by the metaheuristic that measures the closeness of the designed solution to the desired aim [15–18]. The usage of metaheuristics for the purpose of test case generation has yielded virtuous results [15]. Search-Based Software Testing (SBST) employs metaheuristic algorithms to generate test suites automatically or semi-automatically and they can also provide various features like optimum branch coverage, faster testing, and reduction in the size of effective test suite. Search Based Mutation Testing (SBMT) is an area which is inspired from the working of the SBST. The SBMT applies metaheuristics to optimise the process of test data generation, mutant generation and selection of effective mutant operators [18].

In the past decade, most of the research work on Mutation Testing concentrates to find the faults in the non-scientific software. The proposed approach aims to improve the process with the help of bio-inspired algorithm's to identify the mutants in the scientific codes. The scientific software depends on the accuracy than the correctness of the output. Therefore the general Mutation Testing is not sufficient to find the faults since the scientific software are sensitive even to a small round-off error where the faults may be in a masked form. Though Mutation Sensitivity Testing (MST) [19] has close relations with mutation testing but it differs in the process of dealing with the mutants. It makes use of tolerance instead of strict equality by calculating the relative error between the mutant output and the original output. In this work, we utilize the strength of the SBMT along with the concept of mutation sensitivity to obtain an effective and optimised test suite which is capable of finding faults in the software under test. The search based technique used in this work, is the

Crow Search Algorithm (CSA). We have modified the original CSA approach to overcome the limitations associated with it. In CSA, the search pattern is dependent on the independent parameter known as flight length. This may lead to the problem of search solutions getting trapped within the local area. Therefore, the proposed ICSA approach generates Cauchy random numbers [20] for each of the test cases to prevent the search from getting trapped in local search. The proposed SBMT is handled with each of the individual algorithms namely GA, PSO, ACO, and CSA with the relative error as the fitness function.

The remaining of the work is introduced as pursues. Section 2 discusses the different works related with the software testing. Section 3 gives insights into the CSA technique. Section 4 provides the problem formulation. Section 5 presents the details of Mutation Testing with the ICSA approach. Section 6 illustrates the results obtained and the experimental analysis. Section 7 ends with conclusion.

2 Related Work

In this section, we discuss the various works related to automatic test case generation, strategies and algorithms used in the area of SBMT. Baudry et al. [21] proposed the optimization of test cases with Bacteriological Algorithm (BA) on.Net programs. As BA differs from GA, only in terms of memorization of effective test cases, they applied and explored all the spectrum of algorithms between GA and BA. They found that pure BA optimizes the test cases more efficiently and converges faster. Srivastava et al. [22] presented a modified Firefly Algorithm (FA) to generate optimal test paths. The behaviour of the FA is incorporated to generate an optimal test sequences with the help of state transition diagram and graph reduction. This method can be useful for providing the best test path using other expensive testing procedures.

Lam et al. [23] presented test suite optimization technique by employing the Artificial Bee Colony algorithm (ABC) which may work well for programs with small size. Ant Colony Optimization [24] was used to reduce the cost of test data generation using mutation testing. Ant's behaviour has also been used to enhance testing effectiveness using the state transition testing commonly used in real time and embedded software system [25].

Fraser and Zeller [26] proposed GA for unit-test oracle generation and used cost functions to evaluate a test case. The test cases were generated by using the evolutionary approach, the oracle was then generated to kill the mutants which were then used to generate the java unit test suites. The approach has been proved successful to drive test data generation automatically for object oriented classes. Bueno et al. [27] proposed a diversity oriented test data generation with the help of three metaheuristics namely GA, simulated annealing and the proposed metaheuristic which they named as simulated repulsion. The strategy has successfully improved the mutation score, and data flow coverage. Binh et al. [28] presented an approach to generate test cases for Simulink models. This method consists of three phases. In the first phase, the antibody (taken as test data) is generated in order to kill the mutants. On the completion of the testing process, the best antibody is added to the memory set. In the next phase, a clonal selection algorithm is proposed to choose the test data and in the final phase, optimization is carried out. This approach killed more number of mutants than random testing, thereby proving the effectiveness of automatic test case generation over random test generation. Bottaci [29] formulated a fitness function in Evolutionary Mutation Testing (EMT) which assigns three costs for three conditions (reachability, sufficiency and necessity) to detect the mutants. Based on the observations, their method dealt with the scalability issues in Mutation Testing. The approach improved the score by storing record of all the difficult to kill mutants and equivalent mutants. Fraser and Arcuri [8] implemented an extension to Evosuite to save time and effort in test case generation using mutation testing. Their method dealt with the scalability issue in mutation testing, but improvements on reachability and propagation should be considered. Shulga et al. [30] formulated a State Machine Generator system for the automatic code generation. The program code is generated based on the finite-state automatic transition graph.

Zhang et al. [31] constructed a prediction based approach in mutation testing. This method uses an artificial intelligence approach which paved the way for automatic test generation of mutants. Simulation results show that this approach is highly successful in its prediction capacity but lags where the possibility of mutant execution against each single test case is to be considered. Panichella et al. [32] implemented a strategy to solve the problems associated with test case generation by covering the multiple test targets. Kelly et al. [33] proposed a method to seizure the maximum contradiction and varying the tolerance to simulate oracle. This works well to test the software modules independently. Further, this work needs to be extended in testing computational software to fill the research gap. Durelli et al. [34] conducted a study on Machine Learning (ML) based software testing. The study stated that the ML based approaches were found to be free from the challenges like time and cost incurred in manual mutation testing. However, the study found several issues in using ML for software testing. The huge amount of training data needed, lack of pre-processing methods to convert the data in a learnable format, poor evaluation and lack of research in the area are some of the associated disadvantages. The ML based software testing is still in its primitive stage. Metaheuristics, on the other hand have been extensively applied in the area of SBMT and yielded efficient results.

3 Crow Search Algorithm (CSA)

The CSA that makes use of the crows' foraging behaviour has been proposed by Askarzadeh [35]. Crows are extremely intelligent and are known for their astonishing communication skills and their ability to solve problems. The crows can mimic the voice of the other birds and can communicate the danger within themselves. They possess the ability to learn from other birds. The crows have the habit of stealing the food of other birds and hide their food in some locations. If another crow finds their food hiding location, they can deceive other crows by moving from one place to another, thereby saving their food hiding place. The CSA algorithm makes use of some of the salient characteristics of crows like the memory of crow's food hiding location and the crow's foraging behaviour. CSA and its modified versions or hybrid versions have been applied to many areas since its evolution. Jain et al. [36] modified the CSA, using Levy distribution to adjust the positioning of the crows and demonstrated its efficiency on the non-linear high dimensional functions. Chaotic CSA [37] has been applied for optimal feature subset selection to maximise the classification performance. CSA has also been applied to solve scheduling problems [38] and its evaluation with other optimisation approaches reveals the efficiency of the approach in terms of time calculation and optimality. Researchers have also hybridised the CSA approach with other approaches [39, 40] to apply it to solve complex optimisation problems. Some of the advantages of CSA over other metaheuritics is that, the CSA has fewer parameters and is easy to implement as compared to other well-known metaheuristics namely GA, ACO and PSO. The convergence rate of CSA has also been empirically proved to be faster than other metaheuristics [38].

The mapping of CSA to the problems of optimised test suite generation using mutation testing is described here. The test cases corresponds to the crows in the search space. The fitness function for the search based approach (ICSA) is the relative error concept used by MST [19]. At each iteration, various crows (test cases) change their position and their corresponding fitness is evaluated. The optimum solution is achieved by finding a set of test cases which exhibit higher relative error.

The CSA consists of d-dimensional environment which holds the number of crows. The crows represents each of test cases. The collection of test cases in the d-dimensional environment corresponds to the test suite which needs to be optimized in order to obtain better test cases. The position of the test case changes based on the path taken by the crow to access their food. Each of the new position is evaluated by checking their fitness function. There are M crows in the search space. The crow's i th position on a search space is given as $y^{iter}\left(i = 1, 2, \ldots M;\ iter = 1, 2, \ldots iter_{\max}\right)$ where $iter_{\max}$ is the maximum number of iterations and $y^{i,iter} = y_1^{i,iter}, y_2^{i,iter}, \ldots y_d^{i,iter}$. Initially, the crows are assigned a memory randomly without any constraints. Their initial position is considered to be the best position. The position is denoted as $m^{i,iter}$. The notations are provided in Table 1.

At the ith iteration, when a crow j wants to visit the hidden food source, the crow i decides to follow crow j to steal its food. Two conditions may arise based on the crow's behaviour.

Case 1 The first condition is when the Crow j is not aware that it is being followed by another crow i. At such an instance, the crow i will find the food hiding location of crow j. Now the position of crow i is given by the following equation:

$$y^{i,iter+1} = y^{i,iter} + r_i * fl^{i,iter} * \left(m^{j,iter} - y^{i,iter}\right) \tag{1}$$

where r_i is a random number uniformly distributed between 0 and 1. $m^{j,iter}$ is the memory of hiding place of crow j. The value of $fl^{i,iter}$ decides whether the search will be confined to local area or it will be global. The smaller the value of flight length, the search will be restricted to local area. When the values are larger, the search will be global.

Case 2 The second condition according to CSA is when the crow j is not aware that a crow i is following it. In order to protect its food from being pilfered, the crow j will shift to some other position in the search space. The position is given as

Table 1 Notation table

Notation	Explanation
i and j	Crows
fl	Flight length
$fl_{\max}$	Flight length maximum
$fl_{\min}$	Flight length minimum
A_p	Awareness probability
$y^{i,iter}$	Initial position of crow
$y^{i,iter+1}$	New position of crow
$m^{j,iter}$	Memorized position
r_i	Random number between 0 and 1
C	Cauchy random number

$$y^{i,iter+1} = random\,position \tag{2}$$

The two main states of the CSA is given as

$$y^{i,iter+1} = \begin{cases} y^{i,iter} + r_i * fl^{i,iter} * \left(m^{j,iter} - y^{i,iter}\right) & if\ r_j \geq Ap^{j,iter} \\ a\ random\ position & otherwise \end{cases} \tag{3}$$

In CSA, the intensification and diversification are controlled by the parameter called awareness probability *Ap*. With decrease in the value of *Ap*, search will be confined within the local region, while with an increase in the value of *Ap* the search space is explored beyond the local region. The small values of *Ap* will result in intensification and the large values will result in diversification.

The steps in the optimization process are given as follows.

1. The parameters like the flock size, the flight length, the awareness probability are initialized. The position of the crow is randomly initialized.
2. Initially the crows had no experience with foraging. Therefore the memory corresponds to the initial position.
3. The fitness function value is calculated for the memorized position. Thus the quality of the position of the crows are evaluated.
4. The new position of the crows are generated as follows. A random crow is selected from the flock as the chased crow and the chasing crow follows the selected random crow to access the food hiding location. The new position of the crows is generated based on Eq. 3.
5. The practicability of the position of crows are evaluated and the positions are updated.
6. The fitness function of the new positions are evaluated.
7. The fitness of new position is compared with the fitness of memorized position based on the step 2. If the value is better, the memory is updated.
8. The above steps are executed till the maximum iterations. The best memorized position is then selected as the optimal solutions. Then, another flock of crows are selected and whole process is repeated to obtain the best of the crow from the selected flock.

3.1 Problems in CSA and its Solution in ICSA

In a metaheuristic algorithm, two opposite criteria namely diversification and intensification must be taken into account for optimal results. In CSA, the balance between intensification and diversification is maintained through two factors: the flight length ($fl^{i,iter}$) and the awareness probability (*Ap*). It states that the small value of flight length *fl* leads to local search and the large values leads to global search. Likewise, with the decrease in value of *Ap*, the search is confined to local area and higher values lead the search on a global scale.

The CSA approach chooses the flight length value statically before the execution of the optimisation approach and remains fixed for all the population. In ICSA, we have chosen the value of *fl* dynamically (using Cauchy random number) and is generated for each test case value. The flight length is replaced by the Cauchy random number which is generated from the inverse of Cumulative Density Function (CDF) of Cauchy distribution [20]. The CDF of the Cauchy distribution is given by the Eq. (4)

$$F(y) = 1/2 + 1/\pi \arctan(y/t) \tag{4}$$

where t is the scaling factor, whose value here is taken as 1 and y corresponds to each of the test case. This operator helps to decrease the problem of solutions getting trapped in local minima. Further, it also helps to balance the search pattern by adjusting the scaling factor. The Cauchy random number C is obtained by inverse of the Eq. (4).

The value of awareness probability (Ap) in ICSA corresponds to the minimum threshold value that is the minimum number of mutants to be killed by the test case. It decides whether the number of mutants killed by the test case is acceptable or not for each of the new test cases (new position of crow) and it has been adjusted for each of the benchmark programs used. If the fitness of the new position of crow is greater than the minimum threshold, then it is compared with the fitness of the memorised position else a random value of test case is selected. This further helps in maintaining the balance between intensification and diversification.

Therefore, the proposed algorithm (ICSA) enhances the searching capability of the metaheuristic technique (CSA) by choosing the value of flight length dynamically and setting the value of awareness probability.

4 Fitness Function

The concept of mutation testing is based on a rule which states that mutants are detected or killed when the output of the mutated code is not equal to the output of the original code for a test case. The testing of scientific software should concentrate more on the accuracy of the output rather than the correctness. Therefore the concept of mutation sensitivity [19] is utilized to obtain the optimized test suite. The research focus is to generate the optimized test suite which has minimum number of test cases that can kill maximum number of mutants. There are possibilities when both the mutated and original code can produce the same results. The mutants may also get killed when there is a small change in the round off error. The equality concept is replaced by the tolerance concept to select the best test case which can identify the mutant. Therefore, in this work we make use of a fitness function that can optimize the process of test suite generation based on the concept of the MST. It utilizes the relative error as the fitness function to identify the test case which reveals the faults more efficiently. The fitness function is adapted from [19].

Let P_o be the original program which is mutated to produce P_m

$$\gamma(P_m, t) = (|P_m(t) - P_o(t)|)/|P_o(t)| \tag{5}$$

where γ is the relative error and t be the test case. A maximum relative error can be calculated from the above Eq. (5) as follows.

$$\tau\big(P_m(t) = \max_{t\in T}\{\gamma(P_m, t)\} \tag{6}$$

where τ is the maximum relative error for a given P_m, t is the test case and T is the set of test cases or the test suite. Relative error will exhibit changes according to the strength of the test cases. When appropriate test cases are chosen, there will be an increase in the relative error [19]. The test case which generates maximum relative error identifies the faults in the code more accurately.

5 Proposed Improved Crow Search Algorithm (ICSA)

ICSA makes use of the foraging behaviour of crows along with the Cauchy random number that enhances the exploration capabilities of the search based approach and hence the efficiency of the generated test suite. The Cauchy random number is calculated for each of the test case. The modifications may help in exploration of the wide range of solutions and to obtain the optimal results. The modified approach helps in generation of efficient test cases using the concepts of mutation testing.

The Fig. 1 shows that the crow *j* can search within the local area and go beyond the local search after applying the Cauchy random number. The figure shows that the crow can search its solution between the initial position and memorized position as well as go beyond the memorized position. While in the traditional CSA, the crow either moves beyond the memorized position or stays within the memorized position, when the flight length is greater or lesser than one.

Figure 2 shows the architecture of the proposed approach. Here, the implementation of ICSA for test suite generation using mutation testing is explained. The approach is explained through the pseudo code in the next subsection.

5.1 Implementation of ICSA on Test Suite Generation Problem

The section discusses how we implement the ICSA approach towards optimal test suite generation and detecting the mutants. Crows correspond to the test cases in the *d*-dimensional search space. The population size of the crow corresponds to number of test cases (*N*). The position and memory are initialized according to the principles of the CSA. The test cases from the memorized position are evaluated with the help of fitness function. A random test case is selected and its new position is calculated according to the equation

$$y^{i,iter+1} = y^{i,iter} + r_i * C * (m^{j,iter} - y^{i.,iter})\, r_j > AP^{j,iter}_{\max} \tag{7}$$

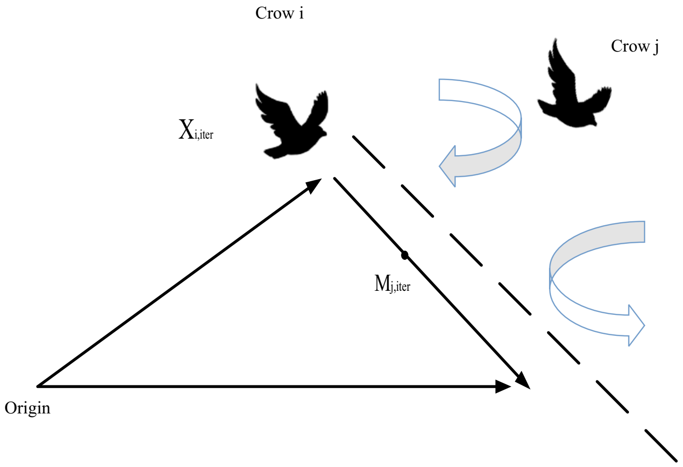

Fig. 1 Effect of Cauchy random number in ICSA

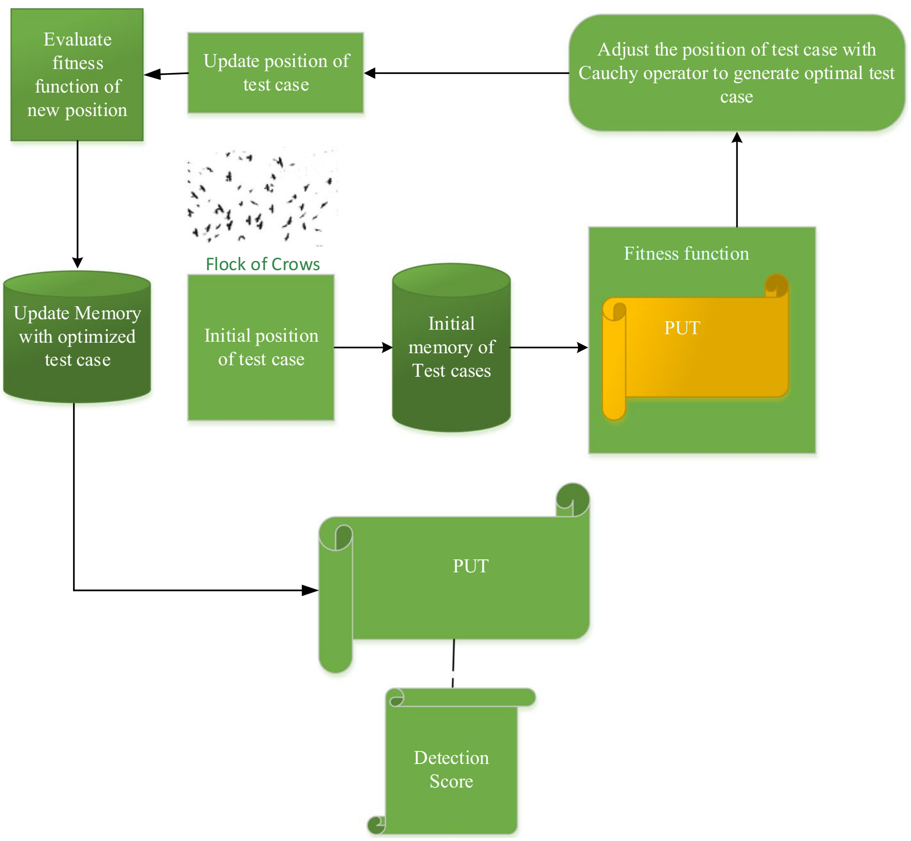
Evaluate fitness function of new position
Update position of test case
Adjust the position of test case with Cauchy operator to generate optimal test case
Flock of Crows
Update Memory with optimized test case
Initial position of test case
Initial memory of Test cases
Fitness function
PUT
PUT
Detection Score

Fig. 2 Proposed architecture of test suite generation using mutation testing with ICSA

$$y^{i,iter+1} = a\ random\ position\ otherwise \tag{8}$$

r_i and r_j are the random number with uniform distribution between 0 and 1. It is found that the fitness function of the new positions are better than the memorized ones. The process is continued till maximum iterations are met. The best of the memorized positions are selected as the optimal test suite. With optimal test suites, the detection score is calculated. The detection score [19] is given as the fraction of mutants killed by the test cases for a given mutated program and γ_d is the detection boundary.

$$Detection\ score = |D(\gamma_d, P_m, T)|/|P_m| \tag{9}$$

Pseudo code for ICSA

```
Position and memory of crow are initialized after adjusting the flock size and number of iterations.
while iter < $iter_{\max}$
for $i = 1 :::::: M$ (all m crows of the flock)
Evaluate the fitness function of the initial memory.
Generate Cauchy random number for the test cases
Choose a random test case from memory
Select the awareness probability
If $r_j \geq A_p^{j,iter}$
Update the position of the crow according to (7)
Else
update the position according to (8)
end if
end for
check the feasibility of new position
Evaluate the new position of crow
Update the memory if the fitness of new position is better than the memorized position.
Repeat the whole process with another flock of crows
end while
```

6 Experimental Result and Analysis

Simulation of the algorithm is carried out in MATLAB 2018b on a 64-bit Intel (R) I5 2.30 GHz CPU with 8 GB RAM. The maximum number of iterations $itr_{\max}$ is set to 300 for smaller programs, and 100 or 50 for larger programs due to their higher execution time. The execution is repeated up to 20 times in order to find the detection score and for the statistical ranking of the algorithms. The mutants are generated with the help of a code. The Mothra mutation operators [10] from Table 2 are used to mutate the original program. The Table 3 provides the list of benchmark programs used for empirical evaluation along with their relevant details.

6.1 Benchmark Algorithms

The simulations are carried with the various well known metaheuristics namely GA, PSO, ACO and CSA in order to compare the performance with the proposed ICSA approach. The fitness function used for GA, PSO, CSA and ICSA is the detection score. Whereas for the ICSA approach, the fitness function used is as described in Sect. 4. The details of the benchmark algorithms are also presented in this section.

Genetic Algorithm The genetic algorithm is a search based technique employed to solve many optimization problems. The method is based on the evolution phenomenon. Software engineering has employed this algorithm to solve complex problems. The key operators in the algorithm are selection, crossover, mutation and elitism. In the proposed work, GA is employed to optimize the test cases with the fitness function used in the proposed algorithm. The chromosomes of GA corresponds to the test cases. The algorithm helps to obtain optimized test cases with the help of fitness function. The parameter selection is adopted from the paper [42].

Table 2 Mothra mutations

Mutation operator	Description
ACR	Array reference for constant replacement
ABS	Absolute value insertion
AAR	Array reference for array reference replacement
AOR	Arithmetic operator replacement
ASR	Array reference for scalar variable replacement
CAR	Constant for array reference replacement
CNR	Comparable array name replacement
DER	DO statement end replacement
CSR	Constant for scalar variable replacement
CRP	Constant replacement
DSA	DATA statement alterations
ROR	Relational operator replacement
LCR	Logical connector replacement
GLR	GOTO label replacement
RSR	RETURN statement replacement
SAN	Statement analysis
SAR	Scalar variable for array reference replacement
SDL	Statement deletion
SCR	Scalar for constant replacement
SVR	Source variable replacement
SRC	Scalar constant replacement

Table 3 Benchmark programs used for analysis

Program	LOC	Description	Source
Simpson	170	Numerical Integration Method	[41]
Nearest Neighbor	300	A proximity search method	[41]
Kruskal	120	find the edge of the least possible weight	[41]
Greedy Best First search	423	Finds the minimum spanning tree for a connected weighted graph	[41]
N dimensional sparse array	3231	Creates N-dimensional sparse array object	[41]
Mason	400	Simplifies the signal flow graph with masons rule	[41]
Astar Algorithm	250	Path finding algorithm	[41]
Calendar	130	Prints the Calendar	[41]
Dijkstra's Minimum Cost Path Algorithm	331	Calculates minimum cost and path using Dijkstra's Algorithm	[41]

Particle Swarm Optimization The PSO is employed to optimize the test cases which is based on the movement of the particles [43]. The particles corresponds to test cases is optimized in order to obtain a better mutation score. The particles in the search space moves still the test cases finds a better one. If the optimized test cases are able to kill the mutants, the test case is added to the optimized test suite. The parameter selection is adopted from the research work [44].

Ant Colony Optimization The ACO algorithm [45] is based on the movement of the ants. The ACO discovers solution with the aid of trail pheromones. Mutation testing utilizes the ants moving pattern in order to search for optimized test cases from the search space. The parameters of ACO and details are followed from the research work [45].

Crow Search Algorithm The CSA algorithm [35] works by simulating the food searching pattern of the crows. The crows hide their food in certain location and can fool the other birds to protect their food location position. The parameters of the CSA are adopted from [35].

6.2 Evaluation Metric

The evaluation metric used to compare the performance of the benchmark algorithm is the detection score. Table 4 provides the detection score for each of the benchmark programs used for evaluation. These results over the benchmark algorithms have been evaluated with the fixed number of iterations.

The Table 5 gives the categorized mutants like revealed, unrevealed and terminal mutants. The revealed mutants denotes the number of mutants killed by the proposed approach. The unrevealed mutants are the equivalent mutants. The terminal mutants are the one which failed terminally.

6.3 Sensitivity Analysis

The Morris method [46] has been used for sensitivity analysis. The elementary effect method is employed to find the sensitivity. The elementary effect is calculated for the proposed method is given as

$$EE(C) = f\left(c_1, \dots c_i + \Delta, \dots c_D\right) - f(C)/\Delta \tag{10}$$

where C represents the Cauchy random number and Δ represents the offset. The analysis helps to identify the input parameter's effect on the detection score. The absolute mean σ_i is evaluated in order to find the sensitivity of input parameter. The standard deviation μ_i is also evaluated since there may be the presence of non-linear behaviour for some input parameters [46].

Table 4 Detection Score achieved for the benchmark algorithms and the proposed approach

Programs	GA	PSO	ACO	CSA	ICSA
Simpson	33.53	33.36	51.3	61.2	82.61
Nearest Neighbor	20.3	54.46	55.9	54.21	84.23
Kruskal	19.7	52.93	52.21	87.22	89.21
Greedy Best first search	23.42	54.27	54.71	88	88.20
N-dimensional sparse array	22.3	35.6	58.9	59	81.6
Mason	45.4	45.6	40.3	80	74.5
Astar algorithm	37.5	37.22	82.34	88	88.1
Calendar	40.1	39.9	68.21	95	95.23
Dijkstra's Minimum Cost Path Algorithm	38.8	38.62	59.7	81	75.21

Table 5 Mutant categories

Target program	Revealed mutant	Un-revealed mutant	Terminal mutant
Simpson	200	6	34
Nearest Neighbor	400	4	47
Krushal	190	6	43
Greedy Best first search	500	5	52
N dimensional sparse array	1000	9	120
Mason	250	4	33
Astar algorithm	300	6	28
Calendar	100	3	43
Dijkstra's Minimum Cost Path Algorithm	350	7	49

The lesser values of σ_i and μ_i denotes that the input parameter has lesser effects on the model. The lesser the value of σ_i and higher value of μ_i says that the there is a higher linear effects on the model. The lower value and high value of σ_i, μ_i implies that the input values have a higher non-linear effects. The high values of both σ_i and μ_i denotes that the input parameters have high non-linear effect. Based on the Morris sensitivity analysis, a table is formulated for the benchmark programmes used. The absolute mean and standard deviation for a set of Cauchy numbers are provide in the Table 6.

The table shows how much variation is experienced when the value of C changes for each of the test case.

6.4 Rank Summary of Statistical Assessment Result Comparison of the Metaheuristic Techniques

The performance of the metaheuristic algorithms have been compared with the help of the statistical method named Wilcoxon Signed Rank Test [47]. The analysis of the 9 benchmark programs is presented. The Table 7 provides the result of rank Wilcoxon signed rank test.

The statistical assessment of the simulation results provides the comparison between various algorithms. The research work ranks the algorithm based on the Wilcoxon signed rank test as shown in Table 7. The algorithm which has attained the best results is ranked as 1 and others are ranked as 2, 3 respectively. In some of the benchmark algorithms, there was a possibility of same rank due to similar detection score achieved. In such cases, an average value of same ranking position has been assigned. The ICSA has achieved the best performance on 7 out of 9 benchmark functions. CSA is the second algorithm which has

Table 6 Sensitivity analysis on Cauchy random number

Cauchy random number									
C_1 12.82		C_2 8.75		C_3 6.77		C_4 4.88		C_5 3.72	
σ_i	μ_i	σ_i	μ_i	σ_i	μ_i	σ_i	μ_i	σ_i	μ_i
65.6	8.1	54.6	7.3	43.5	5.4	33.7	3.7	33.4	3.6

Table 7 Pair-wise Wilcoxon signed rank test results

Benchmark programs	Wilcoxon signed rank test order
Simpson	ICSA > CSA > ACO > PSO = GA
Nearest Neighbor	ICSA > CSA = PSO > ACO > GA
Kruskal	ICSA > CSA > ACO = PSO > GA
Greedy Best first search	ICSA = CSA > ACO = PSO > GA
N dimensional sparse array	ICSA > CSA = ACO > PSO > GA
Mason	CSA > ICSA > GA = PSO > ACO
Astar algorithm	ICSA = CSA > ACO > PSO = GA
Calendar	ICSA = CSA > ACO > PSO = GA
Dijkstra's Minimum Cost Path Algorithm	CSA > ICSA > ACO > PSO = GA

achieved a better performance followed by ACO, PSO and GA. The Table 8 provides the rank summary.

7 Conclusion

In this study, the popular metaheuristics used in the area of SBMT have been applied for test suite generation. The major contribution of this work, is to propose a modified version of Crow Search Algorithm for test suite generation using mutation testing. The modified version has been shown to mitigate the problem of solutions getting trapped into the local minima. We have employed the concept of mutation sensitivity testing as the fitness function for identifying the efficient test cases. The empirical evaluation reveals that the proposed approach gives better results as compared to the other popular algorithms used in the literature of SBMT. The efficiency of results shown by the proposed approach is due to the modifications made in the working of the algorithm to fit the test suite generation problem that helps the search based approach to strike a balance between exploration and exploitation to achieve the global optimum and the fitness function that further enhances the efficiency of the test cases. The results have been statistically reported by using the Wilcoxon signed rank test.

Table 8 Rank summary of statistical assessment results

Program	ICSA	CSA	ACO	PSO	GA
Simpson	1	2	3	4.5	4.5
Nearest Neighbor	1	2.5	4	2.5	5
Kruskal	1	2	3.5	3.5	5
Greedy Best first search	1.5	1.5	3.5	3.5	5
N dimensional sparse array	1	2.5	2.5	4	5
Mason	2	1	5	3.5	3.5
Astar algorithm	1.5	1.5	3	4.5	4.5
Calendar	1.5	1.5	3	4.5	4.5
Dijkstra's Minimum Cost Path Algorithm	2	1	3	4.5	4.5

Appendix

Function: Area of a triangle; Let the **Original Program** = bh/2; and **Mutated Programme** = (b + h)/2

Step 1: Initialize the test cases and memory.

Let the initial test cases be $T = \begin{bmatrix} 2 & 3 \\ 4 & 6 \\ 8 & 9 \end{bmatrix}$; let the memory be randomly allocated as $M = \begin{bmatrix} 9 & 6 \\ 2 & 8 \\ 4 & 3 \end{bmatrix}$

Step 2: Evaluate the fitness function

$$\text{Relative error} = \begin{bmatrix} 0.722 \\ 0.375 \\ 0.416 \end{bmatrix} \tag{11}$$

Step 3: Based on the principle of CSA the new position of test case is generated from memory and original position according to the following formula.

New position of each test case = original position +0.5*flight length*|(memory-original position)|.

Now flight length is adjusted such that *fl* is less than 1. Let *fl = 0.5*

$$\text{The new position is given as} = \begin{bmatrix} 3 & 3 \\ 4 & 6 \\ 4 & 2 \end{bmatrix} \tag{12}$$

Step 4: The fitness function of new position is evaluated

$$\text{Relative error} = \begin{bmatrix} 0.325 \\ 0.166 \\ 0.255 \end{bmatrix} \tag{13}$$

It is found that there is no much improvement in the relative error when (3) is compared to (1). This shows that when flight length is less than one, search solutions are restricted to local area. When the flight length is greater than one, search will be a purely global one.

Step 5: When the Cauchy operator is applied to the initial position the following result occurs. The Cauchy random number for the each of the test cases is obtained from the inverse of the Cauchy distribution equation: equation $F(x) = 1/2 + 1/\pi \arctan(x/t)$, t > 0

$$\text{The Cauchy random number for each of the test case is } C = \begin{bmatrix} 12.82 & 8.75 \\ 6.77 & 4.88 \\ 4.00 & 3.72 \end{bmatrix} \tag{14}$$

Step 6: Replace the value of flight length with the Cauchy random number and calculate the new position and fitness function.

The new position we obtained after applying Cauchy random number is given as

$$\begin{bmatrix} 44 & 15 \\ 2 & 10 \\ 16 & 3 \end{bmatrix} \tag{15}$$

The fitness function is calculated for the new positions (5) is given as

$$\text{Relative error} = \begin{bmatrix} 0.911 \\ 0.4 \\ 0.604 \end{bmatrix} \tag{16}$$

The test cases obtained after using the Cauchy random number shows that the search yields better results. The fitness function of the new position is found to be better than the memorized position. The matrix shown by Eq. (6) is found to be better than that shown in (3). The test cases which generated the better fitness function is considered to be optimal one. The process is carried out till the highest relative error is obtained or till the termination criteria is met.

References

1. Jamil MA, Arif M, Abubakar NSA, Ahmad A (2016) Software testing techniques: a literature review. In: 2016 6th international conference on information and communication technology for the Muslim world (ICT4M), pp 177–182
2. Kasurinen J (2010) Elaborating software test processes and strategies. In: 2010 third international conference on software testing, verification and validation, IEEE, pp 355–358
3. Jovanović I (2006) Software testing methods and techniques. In: The IPSI BgD transactions on internet research, p 30
4. Chen TY, Lau MF (1998) A new heuristic for test suite reduction. Inf Softw Technol 40(5–6):347–354
5. Jia Y, Harman M (2010) An analysis and survey of the development of mutation testing. IEEE Trans Softw Eng 37(5):649–678
6. Rani S, Dhawan H, Nagpal G, Suri B (2019) Implementing time-bounded automatic test data generation approach based on search-based mutation testing. In: Progress in advanced computing and intelligent engineering. Springer, Singapore, pp 113–122
7. Papadakis M, Kintis M, Zhang J, Jia Y, Le Traon Y, Harman M (2019) Mutation testing advances: an analysis and survey. Adv Comput Elsevier 112:275–378
8. Fraser G, Arcuri A (2015) Achieving scalable mutation-based generation of whole test suites. Empir Softw Eng 20(3):783–812
9. Offutt AJ, Lee A, Rothermel G, Untch RH, Zapf C (1996) An experimental determination of sufficient mutant operators. ACM Trans Softw Eng Methodol TOSEM 5(2):99–118
10. King KN, Offutt AJ (1991) A fortran language system for mutation-based software testing. Softw Pract Exp 21(7):685–718
11. Kintis M, Papadakis M, Papadopoulos A, Valvis E, Malevris N (2016) Analysing and comparing the effectiveness of mutation testing tools: a manual study. In: 2016 IEEE 16th international working conference on source code analysis and manipulation (SCAM), IEEE, pp 147–156
12. Rani S, Suri B (2019) Adopting social group optimization algorithm using mutation testing for test suite generation: SGO-MT. In: International conference on computational science and its applications. Springer, Cham, pp 520–528

13. Bashir MB, Nadeem A (2018) An experimental tool for search-based mutation testing. In: 2018 International conference on frontiers of information technology (FIT), IEEE, pp 30–34
14. Klammer C, Ramler R (2017) Journey from manual testing to automated test generation in an industry project. In: 2017 IEEE international conference on software quality, reliability and security companion (QRS-C), IEEE, pp 591–592
15. Ali S, Briand LC, Hemmati H, Panesar-Walawege RK (2009) A systematic review of the application and empirical investigation of search-based test case generation. IEEE Trans Softw Eng 36(6):742–762
16. Harman M, Jia Y, Zhang Y (2015) Achievements, open problems and challenges for search based software testing. In: 2015 IEEE 8th international conference on software testing, verification and validation (ICST), IEEE, pp 1–12
17. McMinn P (2011) Search-based software testing: past, present and future. In: 2011 IEEE fourth international conference on software testing, verification and validation workshops, IEEE, pp 153–163
18. Jatana N, Suri B, and Rani S (2017) Systematic literature review on search based mutation testing. e Inf Softw Eng J 11(1)
19. Hook D (2009) Using code mutation to study code faults in scientific software (Doctoral dissertation)
20. Wang GG, Deb S, Gandomi AH, Alavi AH (2016) Opposition-based krill herd algorithm with Cauchy mutation and position clamping. Neurocomputing 177:147–157
21. Baudry B, Fleurey F, Jézéquel JM, Le Traon Y (2002) Automatic test case optimization using a bacteriological adaptation model: application to.net components. In: Proceedings 17th IEEE international conference on automated software engineering, IEEE, pp 253–256
22. Srivatsava PR, Mallikarjun B, Yang XS (2013) Optimal test sequence generation using firefly algorithm. Swarm Evolut Comput 8:44–53
23. Lam SSB, Raju MHP, Ch S, Srivastav PR (2012) Automated generation of independent paths and test suite optimization using artificial bee colony. Proc Eng 30:191–200
24. Ayari K, Bouktif S, Antoniol G (2007) Automatic mutation test input data generation via ant colony. In: Proceedings of the 9th annual conference on genetic and evolutionary computation, pp 1074–1081
25. Srivastava PR, Baby K (2010) Automated software testing using metahurestic technique based on an ant colony optimization. In 2010 international symposium on electronic system design, IEEE, pp 235–240
26. Fraser G, Zeller A (2011) Mutation-driven generation of unit tests and oracles. IEEE Trans Softw Eng 38(2):278–292
27. Bueno PM, Jino M, Wong WE (2014) Diversity oriented test data generation using metaheuristic search techniques. Inf Sci 259:490–509
28. Binh NT, Tung KT (2015) A novel test data generation approach based upon mutation testing by using artificial immune system for Simulink models. In: Knowledge and systems engineering. Springer, Cham, pp 169–181
29. Bottaci L (2001) A genetic algorithm fitness function for mutation testing. In: Proceedings of the SEMINALL-workshop at the 23rd international conference on software engineering, Toronto, Canada
30. Shulga TE, Ivanov EA, Slastihina MD, Vagarina NS (2016) Developing a software system for automata-based code generation. Program Comput Softw 42(3):167–173
31. Zhang J, Zhang L, Harman M, Hao D, Jia Y, Zhang L (2018) Predictive mutation testing. IEEE Trans Softw Eng 45:898–918
32. Panichella A, Kifetew FM, Tonella P (2017) Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. IEEE Trans Softw Eng 44(2):122–158
33. Kelly D, Gray R, Shao Y (2011) Examining random and designed tests to detect code mistakes in scientific software. J Comput Sci 2(1):47–56
34. Durelli VH, Durelli RS, Borges SS, Endo AT, Eler MM, Dias DR, Guimarães MP (2019) Machine learning applied to software testing: a systematic mapping study. IEEE Trans Reliab 68:1189–1212
35. Askarzadeh A (2016) A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Comput Struct 169:1–12
36. Jain M, Rani A, Singh V (2017) An improved Crow search algorithm for high-dimensional problems. J Intell Fuzzy Syst 33:3597–3614
37. Sayed A, Ahmad (2019) Feature selection via a novel chaotic crow search algorithm. Neural Comput Appl 31:171–188
38. Antono A, Santosa B, Siswanto N (2018) A meta-heuristic method for solving scheduling problem: crow search algorithm. In: IOP conference series: materials science and engineering, vol 337. no. 1. IOP Publishing
39. Allahverdipoor A, Gharehchopogh FS (2018) An improved K-nearest neighbor with crow search algorithm for feature selection in text documents classification. J Adv Comput Res 9(2):37–48

40. Hassanien AE, Rizk-Allah RM, Elhoseny M (2018) A hybrid crow search algorithm based on a rough searching scheme for solving engineering optimization problems. J Ambient Intell Humaniz Comput 1–25
41. Matlab programs. https://in.mathworks.com/matlabcentral/fileexchange/. Accessed 6 Sept 2016
42. Mishra DB, Mishra R, Acharya AA, Das KN (2019) Test data generation for mutation testing using genetic algorithm. Springer, Singapore, pp 857–867
43. Trelea IC (2003) The particle swarm optimization algorithm: convergence analysis and parameter selection. Inf Process Lett 85(6):317–325
44. Jatana N, Suri B (2019) Particle swarm and genetic algorithm applied to mutation testing for test data generation: a comparative evaluation. J King Saud Univ Comput Inf Sci
45. Dorigo M, Stützle T (2003) The ant colony optimization metaheuristic: algorithms, applications, and advances. In: Handbook of metaheuristics. Springer, Boston, pp 250–285
46. Loubière P, Jourdan A, Siarry P, Chelouah R (2016) A sensitivity analysis method for driving the Artificial Bee Colony algorithm's search process. Appl Soft Comput 41:515–531
47. Woolson RF (2007) Wilcoxon signed-rank test. Wiley Encycl Clin Trials 9:1–3