

A Hybrid Strategy Based on Asexual Reproduction Optimization and the Beta Hill-Climbing Algorithm for Generating an Optimal Test Suite

Zohreh Ghorbani¹ , Einollah Pira^{2✉} , Alireza Rouhi³ , Sajad Esfandyari⁴ 

1. Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran, Email: zohregorbani1378@gmail.com
2. Corresponding author, Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran, Email: pira@azaruniv.ac.ir
3. Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran, Email: rouhi@azaruniv.ac.ir
4. Department of Computer Engineering, Malayer University, Malayer, Iran. Email: sajad.al367@gmail.com

Article Info	ABSTRACT
Article type: Research Article Article history: Received 30 January 2026 Received in revised form 15 May 2026 Accepted 10 September 2026 Published online 1 October 2026 Keywords: asexual reproduction optimization, beta hill climbing algorithm, combinatorial explosion, t-way testing, minimum covering array	In testing software systems involving a large number of input parameters and their various combinations, the phenomenon of “combinatorial explosion” often occurs. The t-way combinatorial testing method, by generating a set of test cases, seeks to provide the maximum possible coverage of parameter combinations. Constructing a covering array with minimal size is considered an optimization problem, and so far, various metaheuristic approaches, such as Genetic Algorithms and Particle Swarm Optimization, have been applied to solve it. Although these methods have succeeded in producing smaller arrays, challenges such as computational overhead and increasing array size still remain, requiring further optimization. In this paper, to enhance testing efficiency and reduce the volume of generated test cases, we employ a hybrid strategy based on Asexual Reproduction Optimization (ARO) and the Beta Hill-Climbing (BHC) algorithm to generate an optimal test suite. By leveraging the exploration capability of ARO and the exploitative capability of BHC, this hybrid approach can not only reduce the size of the covering array but also improve the accuracy and comprehensiveness of testing. Evaluation results and performance comparisons with several well-known methods show that the proposed strategy improves functional testing efficiency.

Cite this article: Saffarinia, A. & et al, (2026), A Hybrid Strategy Based on Asexual Reproduction Optimization and the Beta Hill-Climbing Algorithm for Generating an Optimal Test Suite. *Journal of Engineering Management and Soft Computing*, 12 (4). 210-226.

DOI: <https://doi.org/10.22091/jemsc.2026.15230.1344>



© The Author(s) retains the copyright.

DOI: <https://doi.org/10.22091/jemsc.2026.15230.1344>

Publisher: University of Qom

1) Introduction

Software testing is considered one of the key stages in the software system development life cycle, with its primary goal being the identification of existing faults at the lowest possible cost. This process consists of three main steps: generating a set of test data, executing the software using these data, and analyzing the resulting outcomes. Software testing is examined from two perspectives: structural and functional. In the structural (white-box) testing, test case design is based on the code structure and the system's implementation. In contrast, the functional (black-box) testing is conducted without regard to the internal structure of the software and is based solely on its specified requirements and features (Singh & Singh, 2012). In this approach, expected inputs and their corresponding outputs can be derived from the requirements. The success of black-box testing depends on its comprehensiveness; that is, test data must cover all possible combinations of inputs. However, in large and complex software systems with a very high number of input parameters, this leads to the problem of combinatorial explosion (Nie & Leung, 2011).

The t-way testing method is one of the combinatorial testing approaches that seeks to cover the maximum possible combinations among input parameters by generating a covering array (Pira et al., 2022). In this technique, the value t , also referred to as the strength, indicates that instead of examining all possible parameter combinations of the system, the test suite considers only combinations involving t parameters. In t-way testing, if a system consists of p input parameters, each of which can take d distinct values, the total number of generated test cases is equal to $\binom{p}{t} \times d^t$. This value grows exponentially as the strength t increases. Therefore, the problem of generating Minimum Covering Array Generation (MCAG) is considered an NP-hard optimization problem (Yilmaz et al., 2006). In general, the exact minimum size of such an array is not known in advance, and reducing the number of test cases without the reduction of coverage is regarded as a desirable objective.

The approaches proposed to solve the MCAG problem can generally be classified into two main categories: exact and approximate methods. Exact methods, which are based on mathematical techniques, are capable of generating covering arrays of the minimum possible size; however, they are applicable only to small and specific configurations. In contrast, approximate methods—such as greedy and metaheuristic algorithms—can generate near-optimal covering arrays in less time than exact methods (Esfandyari & Rafe, 2018). In greedy algorithms, there are two main strategies: one-parameter-at-a-time (OPT) and one-row-at-a-time (ORT) (Nie & Leung, 2011). The OPT method incrementally completes the covering array by gradually adding each parameter to the test suite, whereas in the ORT method, the covering array is expanded step by step by progressively adding new rows (Lei et al., 2007). The most significant weakness of greedy algorithms is their inability to escape local optima. Unlike greedy methods, metaheuristic approaches are able to generate covering arrays close to the optimal solution within a reasonable time. Examples of such methods include the Genetic Algorithm (GA) (Esfandyari & Rafe, 2018) and the Teaching–Learning–Based Optimization (TLBO) strategy (Rao et al., 2011). Nevertheless, even these methods are not capable of fully solving the MCAG problem in systems with a very large number of input parameters.

In this research, to improve testing performance and reduce the size of the generated test cases, a hybrid strategy based on Asexual Reproduction Optimization (ARO) (Farasat et al., 2010) and the Beta Hill Climbing (BHC) algorithm (Al-Betar, 2017) is employed to solve the MCAG problem. By

leveraging the exploratory capabilities of the ARO algorithm and the exploitative capabilities of the BHC algorithm, this hybrid strategy can reduce the size of the covering array while also enhancing the accuracy and comprehensiveness of testing.

Section 2 of the paper is devoted to the literature review, covering the theoretical background, such as t-way combinatorial testing and the ARO and BHC algorithms, and also the literature pertinent to this paper. Section 3 presents a detailed description of the proposed strategy. In Section 4, the performance of this approach is evaluated through comparisons with several existing strategies. Finally, Section 5 concludes the paper and provides suggestions for future research.

2) Literature Review

2-1) Theoretical Background

a) T-way combinatorial testing: To fully evaluate a software system, it is necessary to generate a set of test cases that covers all possible combinations of input parameter values. In systems with a large number of parameters, this approach leads to a phenomenon known as combinatorial explosion. T-way combinatorial testing is a method that, by generating a set of test cases, covers all t-wise combinations of parameters rather than all possible parameter combinations.

Consider a hypothetical system with p input parameters, denoted as $(x_1, \dots, x_p)$, such that each input parameter x_i can take one of d_i distinct values. Each test case can be represented as a p -dimensional vector $(v_1, \dots, v_p)$, where v_i denotes one of the possible values of parameter x_i . As an example, consider a simple e-commerce system (ECSS) whose specifications are presented in Table 1 (Ahmed & Zamli, 2011). This system has five input parameters ($p = 5$); the first three parameters each have two possible values, and the last two parameters each can take three distinct values ($d_1 = d_2 = d_3 = 2$, $d_4 = d_5 = 3$). In such a system, faults typically depend on the simultaneous settings of several parameters, and each specific configuration of these values constitutes a test case. For instance, in the ECSS, a fault may occur when the “smartphone” parameter is set to iPhone and the “commercial database” parameter is set to Oracle. To detect this fault, the test case $(-, \text{iPhone}, -, -, \text{Oracle})$ must be included in the test suite (The symbol “-” indicates that the corresponding parameter can take any value). To fully cover all potential faults, all possible combinations of parameter values must be generated. In this example, the total number of required test cases is $72 = 3 \times 3 \times 2 \times 2 \times 2$. Clearly, in systems with a large number of parameters, this number increases dramatically, leading to the phenomenon of combinatorial explosion.

Table 1. E-Commerce Software System (Ahmed & Zamli, 2011)

Payment Server	Smartphone	Web Server	User Browser	Commercial Database
Master	iPhone	iPlanet	Chrome	SQL
VISA	Blackberry	Apache	Explorer	Oracle
			Firefox	Access

In *t*-way testing, the choice of the value t plays a critical role. If t is too small, some essential combinations required to detect faults may not be included in the test suite; conversely, if t is set too large, the number of test cases increases dramatically, and the problem of *combinatorial explosion* reappears. In this method, t is referred to as the *coverage strength*, indicating the depth of parameter combination testing. The resulting test suite from this process is called a *covering array* (CA). For

the hypothetical system mentioned earlier, the covering array is denoted as $CA(N; t, p, d_1, \dots, d_p)$, where N is the total number of test cases. Moreover, if all d_i values are equal to a constant d , the notation can be simplified to $CA(N; t, p, d)$ or $CA(N; t, d^p)$. Formally, the covering array $CA(N; t, p, d_1, \dots, d_p)$ is defined as a two-dimensional $N \times p$ array as follows:

1. In the i -th column of this matrix, each possible value of variable x_i ($1 \leq i \leq p$) must appear at least once.
2. For any arbitrary selection of t parameters, all possible combinations of their values must appear in a two-dimensional $N \times t$ submatrix (Esfandyari & Rafe, 2018).

Input: *maxIterations*: The maximum number of iterations; *L*: The dimension of the given problem; *lb, ub*: The lower and upper bounds of the problem's variables, respectively; and *fit*: The fitness function

Output: An individual with the highest fitness

```

1: Individual parent = Initialize (L, lb, ub); rep = 1;
2: while rep <= maxIterations do
3:   Individual larva = Mutate (parent, L, lb, ub, g);
4:   Compute the probability of  $P_c$  using Formula (1);
5:   Individual bud = Crossover (parent, larva, Pc);
6:   if fit (bud) > fit (parent) then
7:     parent = bud;
8:   end if
9:   rep ++;
10: end while
11: return parent;

```

Figure 1. The Pseudocode of the ARO Algorithm (Farasat et al., 2010)

b) Asexual Reproduction Optimization (ARO): In some organisms, such as bacteria, asexual reproduction is a common method for generating large populations (Zilberberg et al., 2006). In this type of reproduction, a single parent is responsible for transmitting the complete genetic information to the offspring. The ARO algorithm is designed based on a single individual and is inspired by the mechanism of asexual reproduction, particularly the budding process (Farasat et al., 2010). In the first step, the algorithm randomly generates an initial individual (as the parent). Then, until a stopping condition—such as finding the best individual in the population or reaching the maximum number of iterations—is met, the following steps are continuously repeated:

- ✓ A *child*, called a “larva,” is created from the parent using a mutation function. Each individual (chromosome) is represented as a vector of length L . The mutation function randomly selects a subvector of length g from the parent, where g denotes a random number in the range $[1, L]$. Then, each gene in this subvector is altered; specifically, its value is replaced with another randomly chosen value from the same gene’s domain.
- ✓ With a probability of P_a , the genetic material of the parent and the larva is combined using a uniform crossover function to produce a new individual called a “bud.” Generally, this function takes two parents, *parent1* and *parent2*, as input, generates a random binary vector, and constructs the offspring based on the genes indicated by this vector. Specifically, the i -th gene of the offspring is taken from *parent1* or *parent2* if the corresponding value in the random vector is 1 or 0, respectively. The value of P_a is calculated according to Equation (1).

$$P_c = \frac{1}{1 + \ln g} \quad (1)$$

- ✓ The parent and the bud are evaluated in terms of fitness, and the individual with the better performance is selected as the parent for the next generation.

Figure 1 illustrates the pseudocode of the ARO algorithm. In this algorithm, the input parameters lb and ub specify the lower and upper bounds of the problem variables, respectively.

Input: $maxIterations$: The maximum number of iterations, k : The dimension of the given problem, lb, ub, fit : The fitness function, $bw, beta$;

Output: An individual with the highest fitness;

```

1: Individual  $x$  = Initialize ( $k, lb, ub$ );
2:  $rep = 1$ ;
3: while  $rep \leq maxIterations$  do
4:   Individual  $x' = x$ ;
   /// The  $N$  operator
5:    $index$  = a random integer number between 1 and  $k$ ;
6:    $rnd$  = a random number between 0 and 1;
7:    $x'[index] = x[index] + (2 * rnd - 1) * bw$ ;
   /// The  $\beta$  operator
8:   for  $j = 1$  to  $k$  do
9:     if  $rand(0,1) \leq beta$  then
10:       $x'[j] = lb[j] + (ub[j] - lb[j]) * rand(0,1)$ ;
11:     end if
12:   end for
13:   if  $fit(x') > fit(x)$  then
14:      $x = x'$ ;
15:   end if
16:    $rep++$ ;
17: end while
18: return  $x$ ;

```

Figure 2. The Pseudocode of the BHC Algorithm (Al-Betar, 2017)

c) The Beta Hill Climbing (BHC) algorithm: The Beta Hill Climbing (BHC) algorithm is an enhanced version of classical hill climbing, upgraded by incorporating an advanced exploratory operator (Al-Betar, 2017). This operator, denoted by β , is inspired by the uniform mutation mechanism in genetic algorithms. A key feature of BHC is its ability to move between different regions of the search space; by randomly modifying some decision variables, the algorithm can escape local optima and reach better solutions. Unlike traditional hill climbing (Selman & Gomes, 2006), which focuses solely on exploiting existing solutions, BHC simultaneously enhances both exploration and exploitation, aiming to maintain a balance between the two and avoid getting trapped in local optima. The BHC algorithm begins by generating an initial solution $x = \{x_1, x_2, \dots, x_k\}$, randomly initialized within the search space, so that each variable x_i lies in the range $[lb_i, ub_i]$. The objective function $f(x)$ is then evaluated for this solution, and a new solution x' is generated by applying the operators N and β to x . This process continues until the number of iterations reaches the specified maximum, $maxIterations$. The N operator generates a neighbor for the current solution x , as follows: first, one of the variables in x (say the j -th variable) is randomly selected, and its value is modified according to a specific relation defined in Equation (2):

$$x'_j = x_j + rnd \times bw \quad (2)$$

Here, bw is the bandwidth between the current and new value, and rnd is a random number between 0 and 1. The values of all other variables in the new solution x' remain the same as in x . The β operator generates a new solution based on the existing solution x . In this process, each variable of the new solution is selected from the corresponding value in the current solution with probability $(1-\beta)$, or randomly initialized within the allowable range with probability β . The pseudocode of this algorithm is presented in Figure 2.

2-2) Literature Review

As mentioned in the introduction, the number of test cases in a complete (rather than minimal) covering array generated using the t -way strategy grows exponentially with t (the interaction strength). For this reason, various methods have been proposed in the literature to approximate the solution to this problem. Some exact methods, such as Combinatorial Test Services (Krishnan et al., 2007) and TConfig (Williams & Probert, 1996), utilize mathematical concepts like Orthogonal Arrays (OA). The main drawback of these approaches is that they can generate optimal covering arrays only for small and specific configurations. In contrast, approximate methods can produce nearly optimal arrays in less time. These methods are generally divided into two main categories: greedy strategies and metaheuristic approaches.

Greedy Strategies: This category of strategies uses two different approaches: *one-row-at-a-time* (ORT) and *one-parameter-at-a-time* (OPT) (Nie & Leung, 2011). In the ORT approach, the covering array is constructed row by row; in each iteration, a test case with the highest coverage (or weight) is added to the test suite. This process continues until all t -wise combinations of input parameters are covered in the generated set. The first strategy based on ORT is the *Automatic Efficient Test Generator* (AETG), which greedily selects a test case from multiple candidates (Cohen et al., 1997). Based on AETG, two other methods, called mAETG (Cohen, 2004) and mAETG-SAT (Cohen et al., 2007), have been introduced. The PICT tool (Czerwinka, 2008) also employs the ORT approach and covers all interactions by randomly selecting test cases; however, due to its random nature, the output is not always optimal. Another example that uses the ORT method is the Jenny tool (Jenkins, 2009). In this approach, single-wise interactions are first covered, and then the test suite is gradually expanded to cover two-wise, three-wise, and subsequently, up to t -wise interactions. Compared to previous methods, Jenny typically operates faster and produces smaller covering arrays in most configurations.

Metaheuristic Approaches: In this group of strategies, the goal is to find approximations close to optimal covering arrays within a reasonable amount of time. Another advantage of these strategies is their ability to escape local optima. Similar to the ORT method, these strategies select the test case with the highest weight from a set of candidate options. In the first step, a set of test cases is randomly generated. Then, using a set of operators, their weights are increased. The test case with the highest weight is selected and added to the test suite. This process continues until all t -wise combinations of input parameters are covered in the generated test suite. In general, metaheuristic algorithms can be categorized into nine main groups (Akyol & Alatas, 2017): bio-inspired, physics-inspired, socially-inspired, music-inspired, chemistry-inspired, sports-inspired, mathematics-inspired, swarm-based, and hybrid algorithms that combine the features of these groups. Given the characteristics of the MCAG problem, several metaheuristic algorithms have been applied to solve it. Below, some well-

known strategies based on these algorithms are reviewed in terms of type, maximum interaction strength, and support for variable interactions and constraints.

The *Teaching–Learning-Based Optimization* (TLBO) strategy (Abbasi et al., 2018) is a socially-inspired metaheuristic approach, where the main idea is derived from a classroom learning environment. To generate a test case with the highest weight, a set of test cases is first randomly created. This set then evolves repeatedly through two phases: *teaching* and *learning*. In the teaching phase, the best individual (i.e., the test case with the highest coverage) is selected as the teacher and transfers knowledge to the other members of the population. In the learning phase, all individuals exchange knowledge with one another.

The *Genetic Algorithm* (GA) is a bio-inspired metaheuristic based on Darwin’s principle of evolution, “survival of the fittest.” In this method, a set of the highest-weight test cases is repeatedly selected, and new test cases are generated by applying crossover and mutation operators. The next generation is then formed by combining the current and newly generated test cases. Based on this algorithm, a powerful new strategy called GS (Esfandyari & Rafe, 2018) has been introduced, in which a specified number of crossover and mutation operations are continuously applied to each test case. Empirical results on several configurations show that GS outperforms other approaches in terms of efficiency and can generate covering arrays up to $t = 20$.

The *Multi-Black-Hole* (MBH) strategy is a physics-inspired metaheuristic rooted in the observations of black hole phenomena and the behavior of stars interacting with them (Al-Sammarraie & Jawawi, 2020). This method does not require tuning multiple parameters, making it simple and practical for solving the MCAG problem. In the black hole algorithm, stars represent test cases, and the test case with the highest coverage is selected as the black hole. As the algorithm progresses, all stars (test cases) move toward the black hole based on their current positions and a random number. The MBH variant is based on the same principle but with multiple black holes, meaning black holes with higher energy (test cases with higher weight) can absorb other black holes. Evaluation results show that this approach can generate covering arrays up to $t = 4$.

Other metaheuristic strategies include the Hill Climbing–Bat hybrid (HC-BAT) (Esfandyari & Rafe, 2016), Firefly Test Generator (FATG) (Alsewari et al., 2018), PABC (Alazzawi et al., 2017), ABCS (Alazzawi et al., 2018), and HABCSm (Alazzawi et al., 2021) strategies based on the Artificial Bee Colony (ABC) algorithm (Karaboga, 2010), Whale Optimization Algorithm (WOA) (Hassan et al., 2020), Q-learning-based WOA (QLWOA) (Hassan et al., 2023), Wingsuit Flight Search Optimization (WSFSO) (Che Rose et al., 2024), and Cuckoo Search (CS) (Gandomi et al., 2013). Although these strategies have been able to generate smaller covering arrays, the MCAG problem has not been fully solved yet.

3) Methodology

Since, in the problem of generating an optimal test suite, the appropriate size of the covering array is not known in advance, metaheuristic-based methods must be executed in such a way that, at each step, a test case with the maximum possible coverage is generated. This process continues until full coverage of all possible combinations of t parameters is achieved. To apply any metaheuristic algorithm to this problem, two fundamental modifications to its structure are required:

- Each test case is modeled as a solution (chromosome or individual)
- A special evaluation function is defined to compute the coverage of a test case as the fitness function,

where the weight of each test case is determined based on the number of combinations it covers

Metaheuristic algorithms typically start with a population of randomly initialized solutions. Therefore, in strategies based on these algorithms, a set of test cases is considered as the initial population, and the optimization process is carried out accordingly. The general steps of executing metaheuristic algorithms for generating an optimal test suite include: population initialization, fitness evaluation, generation of a new population, population updating, checking the stopping condition, and continuing the iterative process.

This approach makes it possible to generate a test suite in such a way that all system conditions are covered with the minimum number of test cases. Moreover, due to the flexibility of metaheuristic algorithms, combining multiple algorithms can improve performance and reduce computational time. On the other hand, optimal tuning of parameters such as mutation rate, population size, and the number of iterations is one of the main challenges. Employing hybrid approaches or automatic parameter tuning can enhance the quality and efficiency of the process.

For this purpose, in the remainder of this section, we present a hybrid strategy based on Asexual Reproduction Optimization (ARO) and the Beta Hill Climbing (BHC) algorithm, named AROBHC, for generating an optimal test suite. In the optimal covering array generation problem, it is assumed that the number of parameters is p , and each parameter can take values in the range $[0, d - 1]$. The standard representation of such an array is denoted as $CA(N; t, p, d)$.

To generate a test case with the maximum coverage, the hybrid AROBHC strategy is employed, in which the chromosome structure and fitness function must be properly defined. In this method, each test case is considered as a chromosome and is represented by a p -dimensional vector $(v_1, \dots, v_p)$, such that the value of each gene v_i is one of the possible values in the range $[0, d - 1]$. The fitness evaluation function is defined as a mechanism for computing the weight of each test case, which determines its coverage capability. In other words, the weight of a test case is equal to the number of combinations covered by that test case.

3-1) Details of the Hybrid AROBHC Strategy

The AROBHC strategy, like other metaheuristic strategies, executes the hybrid algorithm based on ARO and BHC algorithms only once for generating each test case, with the objective of maximizing coverage weight. Since the minimum size of the covering array is unknown, this algorithm is executed repeatedly until all possible combinations of t parameters are covered. In this approach, a Coverage Matrix (CM) is used to determine the current coverage status of different combinations.

N denotes the initial population of random solutions, and the variables $AROrate$ and $BHCrate$ represent the percentage of the current population that will be optimized by the ARO and BHC algorithms, respectively. Figure 3 illustrates the flowchart of the AROBHC strategy in three phases for generating an optimal test set.

Initialization Phase: In this phase, a coverage matrix (CM) is constructed, and the variable *RemCov*, representing the number of combinations that are not yet covered, is initialized to $\binom{p}{t} \times d^t$. In addition, an empty test set *TS* is created.

The ARO and BHC Execution Phase: In this phase, the ARO and BHC algorithms are executed sequentially, as described below, to generate a test case, and the best test case (with the maximum weight or coverage) is stored in *tc*. First, a population of *N* random solutions (test cases) is generated using the *initial* function. Then, the ARO and BHC algorithms are executed sequentially for *maxIterations* iterations (computed using Equation (3)). It is worth noting that all solutions (test cases) are primarily sorted in descending order based on their fitness (weight or coverage). Then, the ARO algorithm is executed on *count1* ($= \text{round}(\text{ARORate} \times N)$) of the best solutions available in the current population (*X*). After executing the ARO algorithm, the current population is again sorted in descending order, and the BHC algorithm is applied to *count2* ($= \text{round}(\text{BHCrate} \times N)$) of the best solutions in the current population (*X*).

$$\text{maxIterations} = \frac{\text{maxNFFE}}{3 \times \text{count1} + \text{count2}} \quad (3)$$

where *maxNFFE* refers to the maximum number of fitness function evaluations. Additionally, since the fitness function is invoked three times and one time during each execution of the ARO and BHC algorithms, respectively, the variables *count1* and *count2* are multiplied by these coefficients.

Updating Phase: In this phase, the weight of the test case, generated in the previous phase (*tc*), is determined using the fitness function *calcWeight*, and the obtained value is subtracted from *RemCov*. Then, the test case *tc* is added to the test set *TS*, and the coverage matrix *CM* is updated. The process of executing the ARO and BHC algorithms and updating the coverage matrix *CM* continues until one of the following two conditions is met: either *RemCov* becomes zero, meaning that all possible combinations of *t* parameters are covered by the generated test set, or the execution time of the strategy exceeds 24 hours. Finally, the generated test set is presented to the user.

3-2) Creating a Coverage Matrix

Since, in the proposed strategy, test samples are generated incrementally through the execution of the ARO and BHC algorithms, it is necessary to design a data structure that, at any moment during the algorithm's execution, stores the coverage status of all possible *t*-parameter combinations. This data structure, referred to as the coverage matrix, consists of $\binom{p}{t}$ rows and d^t columns. Here, the variable *p* denotes the number of input parameters of the system, each of which can take *d* different values; therefore, the total number of possible combinations is $\binom{p}{t} \times d^t$. In addition to the d^t columns, each row must include *t* extra columns for storing the indices of the corresponding combinations, as well as one final column for counting the number of combinations that have not yet been covered by any test sample. Consequently, the total number of columns in the coverage matrix is $d^t + t + 1$. At the beginning of the algorithm's execution, since no combination has yet been covered, the values of all d^t columns in each row are initialized to zero, and the value of the final column in all rows is initialized to d^t . Despite the effectiveness of the coverage matrix in explicitly representing coverage information, its space complexity grows rapidly with increasing interaction strength and domain sizes; therefore, its direct application is mostly limited to problems of moderate scale. Consequently,

for very large problems, the use of memory-efficient data structures or approximate/implicit methods for tracking coverage is preferred.

To clarify the concept of the coverage matrix, suppose we have a hypothetical system consisting of $p = 4$ input parameters, each of which can take $d = 3$ possible values. Our goal is to construct the coverage matrix for the covering array $CA(N; 2, 3^4)$. In this case, the number of rows and columns of the matrix are $\binom{p}{t} = \binom{4}{2} = 6$ and $t + d^t + 1 = 2 + 3^2 + 1 = 12$, respectively. As presented in Figure 4, the first two columns (columns 0 and 1) represent the indices of these six distinct combinations (12, 13, 14, 23, 24, 34). The next nine columns (columns 2 to 10) represent the possible decimal values corresponding to the values of these two parameters (00, 01, 02, 10, 11, 12, 20, 21, 22). Finally, column 11 indicates the number of combinations that have not yet been covered. Formally, for any arbitrary combination with indices $i_0 i_1 \dots i_{t-1}$, the index of the corresponding column is computed using Equation (4).

$$c = t + \sum_{j=0}^{t-1} (i_j \times d^{t-1-j}) \quad (4)$$

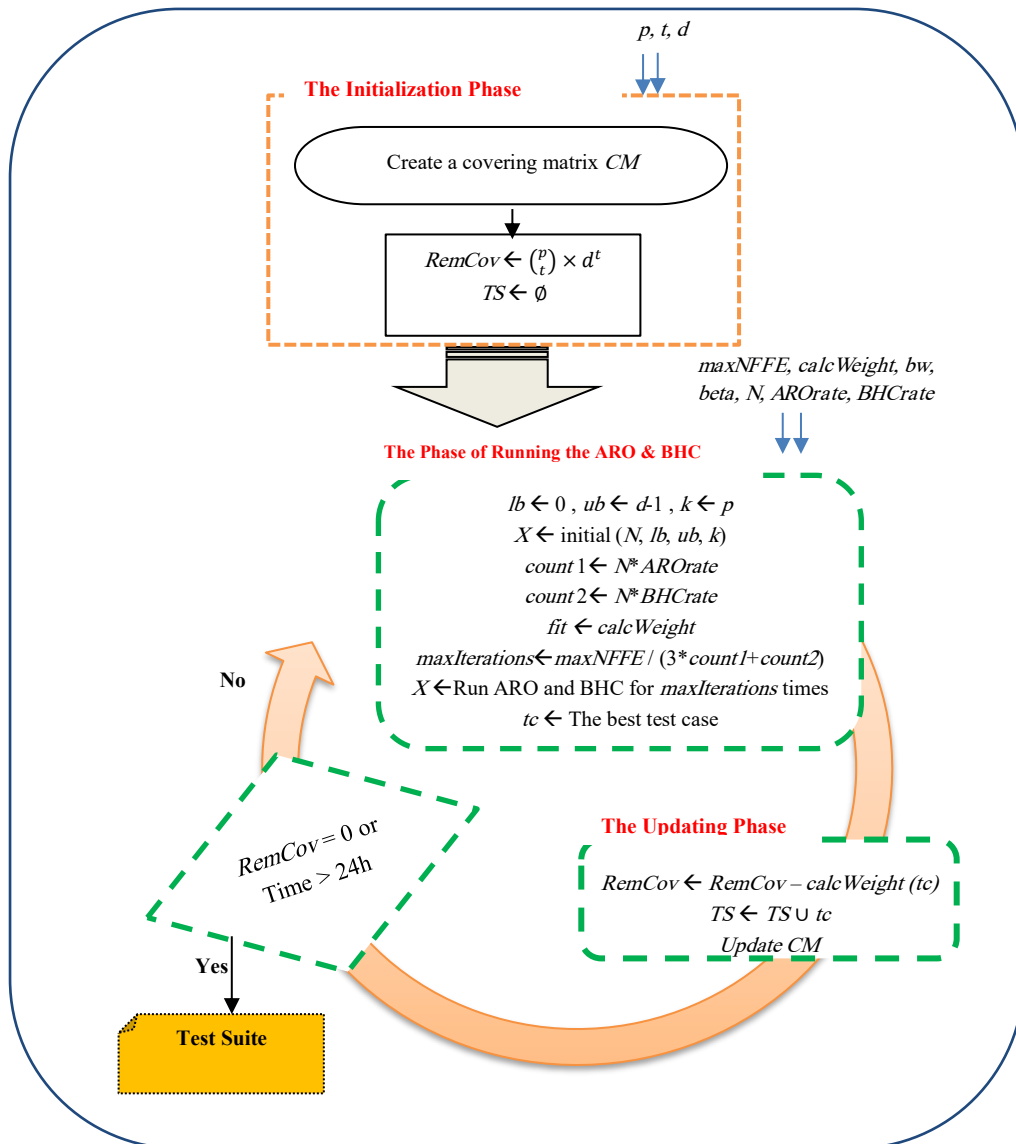


Figure 3. The Flowchart of the AROBHC Strategy

For example, in the considered hypothetical system, if we take an arbitrary combination with index 21, the column number corresponding to this combination is calculated using Equation (3-1) and is equal to $c = 2 + (2 \times 3^1 + 1 \times 3^0) = 9$. In addition, the value of RemCov (the number of uncovered combinations) is initially equal to $\binom{p}{t} \times d^t = \binom{4}{2} \times 3^2 = 54$, which can be observed in Figure 4.

1	2	0	0	0	0	0	0	0	0	0	9
1	3	0	0	0	0	0	0	0	0	0	9
1	4	0	0	0	0	0	0	0	0	0	9
2	3	0	0	0	0	0	0	0	0	0	9
2	4	0	0	0	0	0	0	0	0	0	9
3	4	0	0	0	0	0	0	0	0	0	9

0 1 2 3 4 5 6 7 8 9 10 11

$RemCov = 9+9+ \dots +9 = 54$

Figure 4. The Structure of the Coverage Matrix for the Configuration CA ($N; 2, 3^4$)

3-3) Calculation of the Weight of a Test Sample (calcWeight Function) and Updating of the Coverage Matrix

During the weight calculation process, for each test case, all corresponding t -wise combinations are generated; however, the weight increases only if the corresponding combination has not been previously marked in the coverage matrix. Similarly, the value of RemCov is reduced only for combinations that are newly covered. First, to calculate the weight, based on the values in the test sample and the values of the first t columns in each row, the corresponding column number is computed using Equation (3), and the value of the corresponding cell is obtained. If this value is equal to one, it means that this combination has already been covered by the existing test samples in the set, and there is no need to change the weight of this test sample.

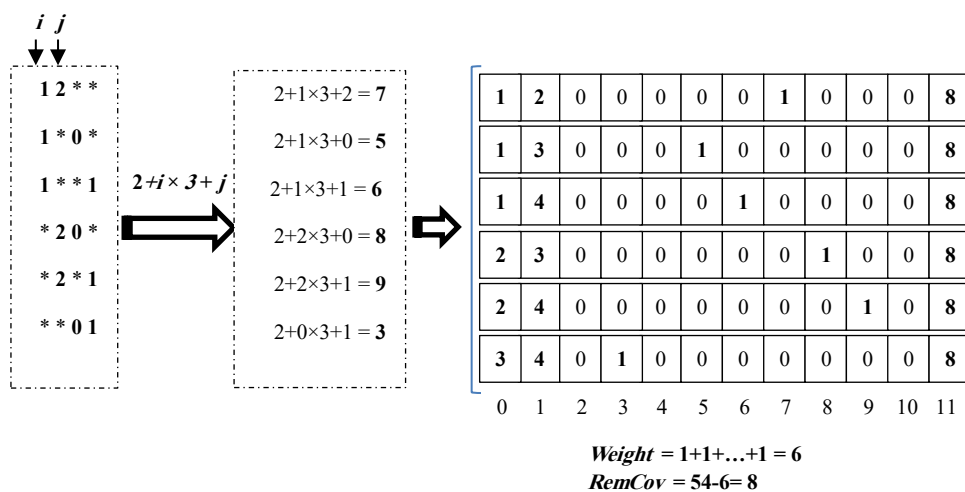


Figure 5. Calculating the Weight of the Test Sample $tc = (1, 2, 0, 1)$ According to the Coverage Matrix in Figure 4 and Its Update

However, if the value of this cell is zero, one unit is added to the weight of the test sample, and the value of this cell is changed to one, indicating that this combination will not be counted in the weight calculation for subsequent test samples. Additionally, one unit must be subtracted from the value in the last column of the corresponding row. Finally, after all rows have been examined, the calculated weight is subtracted from the value of RemCov. To better understand this process, consider the test sample $tc = (1,2,0,1)$. Figure 5 illustrates how the weight of this test sample is calculated and how the coverage matrix from Figure 4 is updated. According to the figure, first, the column numbers corresponding to the different combinations (rows) are calculated using Equation (3), and the cells with zero values are changed to one. Moreover, the values in the last column, which were initially 9, are reduced to 8, and finally, the value of RemCov decreases to 48.

4) Findings and Discussion

To evaluate the effectiveness of the AROBHC strategy and the size of the generated test set, this strategy was implemented in MATLAB version 2017, and the obtained results were compared with those of various other strategies. This comparison includes metaheuristic strategies such as BHC, ARO, GSTG, GS, BAPSO, WOA, and TLBO. Before starting the evaluation and comparison, it is necessary to first determine suitable values for the parameters bw (specific to the N operator), β (specific to the β operator), $AROrate$, and $BHCrate$. For this purpose, several experiments were conducted. According to these experiments, the AROBHC strategy achieves the best performance with the values $bw = 0.5$, $\beta = 0.2$, $AROrate = 0.2$, and $BHCrate = 0.3$. In addition, appropriate values for the parameters required by the other strategies were obtained from the related articles. These suitable parameter values are presented in Table 2. Another important parameter in algorithm execution is the maximum number of fitness function evaluations ($maxNFFE$), which is set to 10,000 for the strategies under consideration. This parameter is directly related to the number of iterations ($maxIterations$) of the algorithms, and the two parameters can be converted into each other. Specifically, if $nFFE$ represents the number of fitness function evaluations per iteration of an algorithm, the total number of iterations for that algorithm can be calculated using Equation (5).

$$maxIterations = \frac{maxNFFE}{nFFE} \quad (5)$$

For example, in the AROBHC strategy, the population size is considered to be 5. Since the ARO algorithm is applied to $count1 (= AROrate \times N = 0.2 \times 5 = 1)$ and the BHC algorithm is applied to $count2 (= BHCrate \times N = 0.3 \times 5 \cong 2)$ of the best solutions in the current population (X), the value of $nFFE$ will be 5. Using Equation 4, the number of algorithm iterations ($maxIterations$) is calculated by dividing 10,000 by 5, resulting in 2,000.

Table 2. Parameters and Appropriate Values for Strategies Based on Existing Metaheuristic Algorithms (Considering $maxNFFE$ as 10,000)

Algorithm	Parameters and Suitable Values
AROBHC	$N = 5, bw = 0.5, \beta = 0.2, AROrate = 0.2, BHCrate = 0.3$
GS	$N = 120, MutRate = 0.4, CossRate = 0.4$
WOA	$N = 180$
	$N = 100, Loudness = 0.9, PulseemissionRate = 0.9,$
BAPSO	$Minfrequency = 0, Maxfrequency = 1, Tolerance = 0.025, WeightValue = 0.4, CycWalkvalue = 1.49$
GSTG	$N = 100, ElitistCheck = 1, Rpower = 1, Rnorm = 2$

To obtain the results, each of the strategies was executed 100 times on the following five datasets: (1) CA (N; 2, p, 3) for $3 \leq p \leq 12$, (2) CA (N; 3, p, 2) for $7 \leq p \leq 12$, (3) CA (N; 3, p, 3) for $4 \leq p \leq 12$, (4) CA (N; 3, 5, d) for $2 \leq d \leq 10$, and (5) CA (N; 5, 7, d) for $2 \leq d \leq 7$. These strategies were executed on a system with an Intel® Core™ i5 processor and 6 GB of RAM. It should be noted that in the results table, the term “NA” is used when the result of executing a strategy for a specific configuration is unavailable, and the term “>day” indicates that the strategy failed to produce an optimal covering array in less than 24 hours.

Table 3 presents the size of the test sets generated by all strategies for the first dataset, namely CA (N; 2, p, 3) with $3 \leq p \leq 12$. This dataset includes 10 different configurations, in which the GS strategy successfully produced the smallest test set in 9 cases. Following this, AROBHC and BHC each achieved the smallest test set size in 8 cases. Compared to these strategies, the others performed worse and were able to generate the smallest test set in fewer configurations. These results indicate that GS, AROBHC, and BHC performed better on this dataset and were able to produce more optimal test sets compared to the other methods.

Table 4 presents the size of the test sets generated by all strategies for the second dataset, namely CA (N; 3, p, 2) in the range $7 \leq p \leq 12$. This dataset includes 6 different configurations, and the analysis of the results shows that AROBHC performed better than the other methods, successfully producing the smallest test set in most configurations.

Table 5 shows the size of the test sets generated by all strategies for the third dataset, namely CA (N; 3, p, 3) in the range $4 \leq p \leq 12$. This dataset includes 9 distinct configurations, and the results indicate that AROBHC outperformed the other methods, producing the smallest test set in the majority of configurations.

Table 6 presents the number of test cases generated by all strategies for the fourth dataset, namely CA (N; 3, 5, d) in the range $2 \leq d \leq 10$. This dataset contains 9 different configurations, and the results show that AROBHC performed better than the other methods, producing the smallest test set in most configurations.

Table 7 shows the size of the test sets generated by all strategies for the fifth dataset, namely CA (N; 5, 7, d) in the range $2 \leq d \leq 7$. This dataset includes 6 different configurations, and the analysis indicates that AROBHC outperformed the other methods, generating the smallest test set in the majority of configurations.

Table 3. Size of the Test Sets Generated by All Strategies for CA (N; 2, p, 3) for $3 \leq p \leq 12$

p	AROBHC N.Best	AROBHC N.Avg	BHC N.Best	ARO N.Best	GSTG N.Best	GS N.Best	BAPSO N.Best	WOA N.Best	TLBO N.Best
3	9	9.87	9	9	9	9	9	9	9
4	9	10.50	9	10	9	9	10	9	10
5	11	12.93	11	12	11	11	11	11	12
6	13	14.63	13	14	13	13	14	14	14
7	14	15.33	14	15	15	14	15	14	15
8	15	15.80	15	16	15	15	15	15	15
9	16	16.70	15	17	17	15	16	16	17
10	16	17.60	16	18	18	16	17	16	17
11	16	17.70	17	18	19	16	17	17	16
12	17	18.23	17	19	19	16	17	17	19

To confirm the superiority of the AROBHC strategy over others, the Friedman’s (1940) test is used. This test is a non-parametric statistical method applied for comparing multiple related samples.

In the results of this test, the strategy with the lowest average rank is assigned first place (indicating the best performance), the next lowest receives second place, and so on, with all strategies ranked accordingly. Table 8 presents the overall ranks of AROBHC and other strategies across all examined datasets. As observed, AROBHC achieved first place in generating test sets of minimal size.

Table 4. Size of the Test Sets Generated by All Strategies for CA (N; 3, p, 2) for $7 \leq p \leq 12$

p	AROBHC N.Best	AROBHC N.Avg	BHC N.Best	ARO N.Best	GSTG N.Best	GS N.Best	BAPSO N.Best	WOA N.Best	TLBO N.Best
7	12	15.03	12	12	12	12	12	12	14
8	15	16.27	12	13	14	14	14	14	16
9	15	16.20	16	15	16	16	15	15	16
10	16	16.60	16	16	16	16	16	16	16
11	17	18.43	17	19	19	17	18	18	17
12	18	21.34	19	21	19	18	18	19	18

Table 5. Size of the Test Sets Generated by All Strategies for CA (N; 3, p, 3) for $4 \leq p \leq 12$

p	AROBHC N.Best	AROBHC N.Avg	BHC N.Best	ARO N.Best	GSTG N.Best	GS N.Best	BAPSO N.Best	WOA N.Best	TLBO N.Best
4	27	31.12	27	27	28	27	27	27	30
5	37	38.41	38	39	40	38	38	38	38
6	39	45.10	39	43	43	43	43	42	44
7	48	50.80	48	51	48	49	48	48	51
8	53	54.91	52	56	53	54	52	53	54
9	56	58.92	56	60	56	58	57	57	58
10	61	61.78	60	65	61	61	59	59	61
11	63	64.81	63	67	63	63	63	64	65
12	65	67.51	65	71	65	67	65	65	67

Table 6. Size of the Test Sets Generated by All Strategies for CA (N; 3, 4, d) for $2 \leq d \leq 10$

d	AROBHC N.Best	AROBHC N.Avg	BHC N.Best	ARO N.Best	GSTG N.Best	GS N.Best	BAPSO N.Best	WOA N.Best	TLBO N.Best
2	12	12.00	12	12	12	12	12	12	12
3	38	40.30	39	39	39	40	40	40	39
4	90	92.20	90	94	90	90	91	93	91
5	173	176.20	173	184	176	178	178	178	176
6	294	298.20	295	316	298	301	302	305	296
7	462	467.30	466	500	468	472	468	478	464
8	688	691.90	687	741	687	697	710	701	692
9	971	975.20	961	1044	973	987	1001	961	973
10	1321	1328.10	1323	1444	1325	1334	1376	1353	1321

Table 7. Size of the Test Sets Generated by All Strategies for CA (N; 5, 7, d) for $2 \leq d \leq 7$

d	AROBHC N.Best	AROBHC N.Avg	BHC N.Best	ARO N.Best	GSTG N.Best	GS N.Best	BAPSO N.Best	WOA N.Best	TLBO N.Best
2	51	55.60	53	54	57	51	57	54	55
3	438	440.52	436	452	441	438	438	439	435
4	1819	1830.21	1819	1920	1843	1819	1819	1823	1836
5	5454	5472.43	5467	5820	5465	5457	5458	5458	5467
6	13339	13365.93	13356	14406	13353	13342	13351	13431	13641
7	28428	28431.42	28437	30855	28531	28512	28573	28512	28514

Table 8. Overall Ranks of AROBHC and Other Strategies Across All Considered Datasets

		AROBHC	BHC	ARO	GSTG	GS	BAPSO	WOA	TLBO
Friedman Test	Mean Rank	2.79	3.14	6.73	4.91	4.03	4.63	4.47	5.31
	Rank	1	2	8	6	3	5	4	7

To enable a more precise comparison among the obtained results, the Wilcoxon signed-rank nonparametric test (Wilcoxon, 1992) is also employed. This test is used to evaluate performance differences between two related samples and to determine in how many cases one algorithm performs better, worse, or similarly to another. The output of this test includes three statistics, R^+ , R^- , and $R=$, which respectively represent the number of cases in which the first algorithm performs better than, worse than, or equal to the second algorithm. Figure 6 presents the results of this test for the pairwise comparison of the AROBHC strategy with each of the BHC, ARO, GSTG, GS, BAPSO, WOA, and TLBO strategies. From this chart, it can be observed that, in most comparisons, the value of R^- is greater than R^+ , indicating that AROBHC produces smaller test suites than other strategies.

Considering that the focus of this study is the empirical comparison of the proposed strategy with commonly used strategies that have reported results on benchmark problems, the claim of AROBHC's superiority should be interpreted within this scope. In other words, the results presented in the tables of this paper show that AROBHC demonstrates competitive—and, in many cases, superior—performance on the set of test problems examined. However, this does not imply general superiority for all values of t and all possible scenarios. Therefore, extending the comparison to methods specifically designed for larger values of t can be considered a direction for future research.

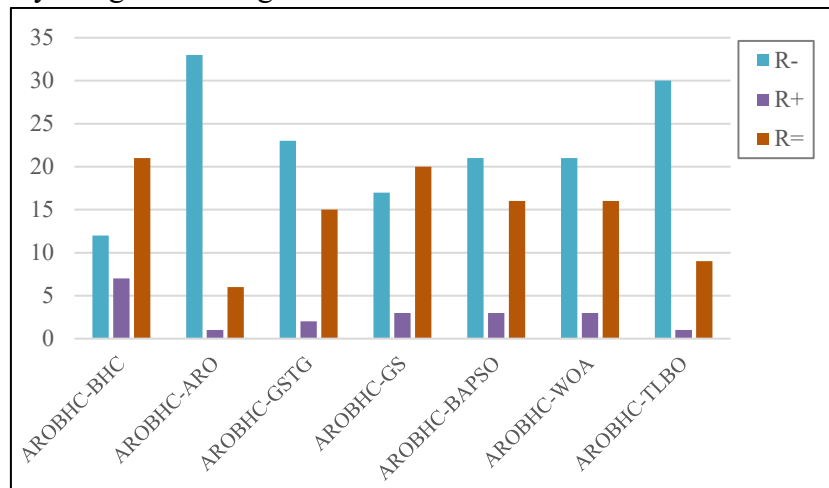


Figure 6. Results of the Wilcoxon Signed-Rank Test for Comparing the AROBHC Strategy with Other Strategies

5) Conclusion

In this paper, a novel hybrid strategy, named AROBHC, based on Asexual Reproduction Optimization (ARO) and the Beta Hill-Climbing (BHC) algorithm, was proposed to address the problem of generating minimal covering arrays. The primary goal of this approach was to reduce the size of the test set while ensuring complete coverage of t -way parameter combinations. Experimental results on various configurations demonstrated that the proposed strategy can, in most cases, generate smaller test sets compared to existing methods, while also being computationally more efficient. Based on statistical analyses, AROBHC achieved the top rank overall, confirming its superiority over other strategies. These results indicate that combining the exploratory capabilities of ARO with the exploitative strengths of BHC can provide an effective approach for solving complex combinatorial testing problems. As a future direction, the development of this strategy could be explored to support

tests with more complex constraints, enhance scalability for systems with a large number of parameters, or implement adaptive versions for automatic parameter tuning.

References

- Abbasi, Z., Esfandyari, S., & Rafe, V. (2018). Covering array generation using teaching learning base optimization algorithm. *Tabriz Journal of Electrical Engineering*, 48(1), 161-171. <https://doi.org/10.1016/j.jss.2011.06.004>
- Ahmed, B. S., & Zamli, K. Z. (2011). A variable strength interaction test suites generation strategy using particle swarm optimization. *Journal of Systems and Software*, 84(12), 2171-2185. <https://doi.org/10.1016/j.jss.2011.06.004>
- Akyol, S., & Alatas, B. (2017). Plant intelligence based metaheuristic optimization algorithms. *Artificial Intelligence Review*, 47(4), 417-462. <https://doi.org/10.1016/j.jss.2011.06.004>
- Al-Betar, M. A. (2017). β -hill climbing: An exploratory local search. *Neural Computing and Applications*, 28(Suppl 1), 153-168.
- Al-Sammarraie, H. N. N., & Jawawi, D. N. (2020). Multiple black hole inspired meta-heuristic searching optimization for combinatorial testing. *Ieee Access*, 8, 33406-33418. <https://doi.org/10.1016/j.jss.2011.06.004>
- Alazzawi, A. K., Homaid, A. A. B., Alomoush, A. A., & Alsewari, A. A. (2017). Artificial bee colony algorithm for pairwise test generation. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9(1-2), 103-108. <https://doi.org/10.1016/j.jss.2011.06.004>
- Alazzawi, A. K., Rais, H. M., & Basri, S. (2018). Artificial bee colony algorithm for t-way test suite generation. In *2018 4th International Conference on Computer and Information Sciences (ICCOINS)* (pp. 1-6). IEEE. <https://doi.org/10.1016/j.jss.2011.06.004>
- Alazzawi, A. K., Rais, H. M., Basri, S., Alsariera, Y. A., Capretz, L. F., Balogun, A. O., & Imam, A. A. (2021). HABCSm: A hamming based t-way strategy based on hybrid artificial bee colony for variable strength test sets generation. *arXiv preprint arXiv:2110.03728*.
- Alsewari, A. A., Xuan, L. M., & Zamli, K. Z. (2018). Firefly combinatorial testing strategy. In *Science and Information Conference* (pp. 936-944). Springer International Publishing. <https://doi.org/10.1016/j.jss.2011.06.004>
- Che Rose, N. H., Othman, R. R., Zakaria, H. L., Suali, A. J., & Ahmad, Z. (2024). Wingsuit flying search optimization algorithm strategy for combinatorial T-Way test suite generation. *International Journal of Advances in Soft Computing & Its Applications*, 16(3). <https://doi.org/10.1016/j.jss.2011.06.004>
- Cohen, D. M., Dalal, S. R., Fredman, M. L., & Patton, G. C. (1997). The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7), 437-444. <https://doi.org/10.1016/j.jss.2011.06.004>
- Cohen, M. B. (2004). *Designing test suites for software interactions testing* [Doctoral dissertation, University of Auckland].
- Cohen, M. B., Dwyer, M. B., & Shi, J. (2007). Interaction testing of highly-configurable systems in the presence of constraints. In *Proceedings of the 2007 international symposium on Software testing and analysis* (pp. 129-139). <https://doi.org/10.1145/1273463.1273482>. <https://doi.org/10.1016/j.jss.2011.06.004>
- Czerwonka, J. (2008). Pairwise testing in the real world: Practical extensions to test-case scenarios. *Microsoft Corporation, Software Testing Technical Articles*. <https://doi.org/10.1016/j.jss.2011.06.004>
- Esfandyari, S., & Rafe, V. (2016). A Hybrid solution for Software testing to minimum test suite generation using hill climbing and bat search algorithms. *Tabriz Journal of Electrical Engineering*, 46(3), 25-35.
- Esfandyari, S., & Rafe, V. (2018). A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy. *Information and Software Technology*, 94, 165-185. <https://doi.org/10.1016/j.jss.2011.06.004>
- Farasat, A., Menhaj, M. B., Mansouri, T., & Moghadam, M. R. S. (2010). ARO: A new model-free optimization algorithm inspired from asexual reproduction. *Applied Soft Computing*, 10(4), 1284-1292. <https://doi.org/10.1016/j.jss.2011.06.004>
- Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1), 86-92. <https://doi.org/10.1016/j.jss.2011.06.004>

- Gandomi, A. H., Yang, X.-S., & Alavi, A. H. (2013). Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems. *Engineering with computers*, 29(1), 17-35. <https://doi.org/10.1016/j.jss.2011.06.004>
- Hassan, A. A., Abdullah, S., Zamli, K. Z., & Razali, R. (2020). Combinatorial test suites generation strategy utilizing the whale optimization algorithm. *IEEE Access*, 8, 192288-192303. <https://doi.org/10.1016/j.jss.2011.06.004>
- Hassan, A. A., Abdullah, S., Zamli, K. Z., & Razali, R. (2023). Q-learning whale optimization algorithm for test suite generation with constraints support. *Neural Computing and Applications*, 35(34), 24069-24090.
- Jenkins, B. (2009). Jenny test tool. In. <https://doi.org/10.1016/j.jss.2011.06.004>
- Karaboga, D. (2010). Artificial bee colony algorithm. *Scholarpedia*, 5(3), 6915.
- Krishnan, R., Krishna, S. M., & Nandhan, P. S. (2007). Combinatorial testing: learnings from our experience. *ACM SIGSOFT Software Engineering Notes*, 32(3), 1-8. <https://doi.org/10.1016/j.jss.2011.06.004>
- Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., & Lawrence, J. (2007). IPOG: A general strategy for t-way software testing. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)* (pp. 549-556). IEEE. <https://doi.org/10.1016/j.jss.2011.06.004>
- Nie, C., & Leung, H. (2011). A survey of combinatorial testing. *ACM Computing Surveys (CSUR)*, 43(2), 1-29.
- Pira, E., Rafe, V., & Esfandyari, S. (2022). Minimum covering array generation using success-history and linear population size reduction based adaptive differential evolution algorithm. *Tabriz Journal of Electrical Engineering*, 52(2), 77-89.
- Rao, R. V., Savsani, V. J., & Vakharia, D. (2011). Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3), 303-315. <https://doi.org/10.1016/j.jss.2011.06.004>
- Selman, B., & Gomes, C. P. (2006). Hill-climbing search. *Encyclopedia of Cognitive Science*, 81(333-335), 10.
- Singh, S. K., & Singh, A. (2012). *Software Testing*. Vandana Publications. <https://doi.org/10.1016/j.jss.2011.06.004>
- Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in Statistics* (pp. 196-202). Springer.
- Williams, A. W., & Probert, R. L. (1996). A practical strategy for testing pair-wise coverage of network interfaces. In *Proceedings of ISSRE'96: 7th International Symposium on Software Reliability Engineering* (pp. 246-254). IEEE.
- Yilmaz, C., Cohen, M. B., & Porter, A. A. (2006). Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering*, 32(1), 20-34. <https://doi.org/10.1016/j.jss.2011.06.004>
- Zilberberg, C., Solé-Cava, A., & Klautau, M. (2006). The extent of asexual reproduction in sponges of the genus *Chondrilla* (Demospongiae: Chondrosida) from the Caribbean and the Brazilian coasts. *Journal of Experimental Marine Biology and Ecology*, 336(2), 211-220.