GALP: A Hybrid artificial intelligence algorithm for generating covering array

SajadEsfandyari[1], Vahid Rafe[2]*
[1,2]Department of Computer Engineering, Faculty of Engineering, Arak University, Arak 38156-8-8349, Iran
Sajad.a1367@gmail.com, v-rafe@araku.ac.ir

Abstract
Today, there are a lot of useful algorithms for covering array (CA) generation, one of the branches of combinatorial testing (CT). The major CA challenge is the generation of an array with the minimum number of test cases (efficiency) in an appropriate run-time (performance), for large systems. CA generation strategies are classified into several categories: computational and meta-heuristic, to name the most important ones. Generally, computational strategies have high performance and yield poor results in terms of efficiency, in contrast, meta-heuristic strategies have good efficiency and lower performance. Among the strategies available, some are efficient strategies but suffer from low performance; conversely, some others have good performance, but is not such efficient. In general, there is not a strategy that enjoys both above-mentioned metrics. In this paper, it is tried to combine the genetic algorithm (GA) and the Augmented Lagrangian Particle Swarm Optimization with Fractional Order Velocity (ALPSOFV) to produce the appropriate test suite in terms of efficiency and performance. Also, a simple and effective minimizing function is employed to increase efficiency. The evaluation results show that the proposed strategy outperforms the existing approaches in terms of both efficiency and performance.

Keywords: ALPSOFV, GeneticAlgorithm, Combinatorial Testing, Covering Array.

1. Introduction

Software systems include different parameters. Each parameter is assigned some values. In exhaustive testing, it is required that all possible states be examined. For example, if the system contains 9 parameters and there are 5 values for each parameter, totally, this system could have59 (1,953,125) distinct inputs that must be tested. As the number of these values grows, it becomes impossible to test the whole system in practice[1]. In other words, a calculation explosion occurs in the test.

CT (Combinatorial Testing) is a good approach to prevent this explosion [2].Test suites generated for CT are of various structures, namely CA, VSCA, and CCA.One of the most common combinatorial tests is CA (Covering Array), also known as the t-way testing, where t denotes the interaction strength [3]. Here, instead of thoroughly examining the interactions of a system, all its t-arys combinations are required. Utilizing computational and artificial intelligence solutions together, CA attempts to completely cover these combinations by sampling several test cases. The less the number of samples (test cases) is, the stronger the solution gets. In general, CA usually deals with three challenges: (1) minimum test case selection (efficiency) (2) execution time reduction (performance), and (3) ability to cover high interaction strength [2]. Furthermore, the strategies could be evaluated for supporting VSCA and CCA. In general, CA generation methods can be classified into four basic categories: (1) artificial intelligence approaches, (2) greedy approaches, (3) mathematical approaches, and (4) stochastic approaches. Among these methods, greedy and AI are more popular. Many of the greedy strategies have good performance, but they are not as efficient as AI strategies. From an efficiency standpoint, the best greedy strategy is In-Parameter-Order-General (IPOG) [4], and the Classification-Tree Editor eXtended Logics (CTE-XL) [5, 6] strategy also shows the best performance compared with other Greedy strategies, but on the other hand, it lacks efficiency and can generate CA up to t = 4. Artificial intelligence (AI) algorithms, which are numerous, are in fierce competition for efficiency. The crucial problem in producing the minimum test suite for CA using meta-heuristic algorithms is getting stuck in a local optimum. To overcome this problem, Discrete Particle Swarm Optimization (DPSO) [7] provides an efficient solution that is not good enough in terms of performance. It is also capable of producing test suites up to t = 10 (taking less than one day to produce CA [2]). Of course, given that the support for the high interaction strength is directly related to performance, DPSO may also support high-strength configurations (t > 10). Different strategies with appropriate efficiency have been proposed that have similar results as DPSO, but unfortunately, since they are unavailable, their performance cannot be discussed. Genetic Strategy (GS) [2] is also another AI-based strategy that has good performance and supports up to t = 20, but is not as efficient as DPSO. In general, any solution with high performance, efficiency, and ability to support the high interaction strength has not been introduced in CA production, so far. Notable capabilities of these strategies include their support of the VSCA. Other powerful algorithms that have been proposed for CCA generation include: CASA [8], TCA [1], FastCA [9], MOCSFO [10], etc.

In this paper, we try to utilize the useful mechanisms of DPSO and GS tools and exploit the advantages of both in the form of a single solution. This solution, called GALP, employs the combination of two meta-heuristic algorithms, namely the Augmented Lagrangian Particle Swarm Optimization with Fractional Order Velocity (ALPSOFV) [11] and Genetic Algorithm (GA), the steps of which are described as follows:

- Generating the data structures needed to calculate the test sample weight and detecting the total coverage of the final test suite (data structure used in GS [2]).
- Combining GA and ALPSOFV to generate a covering array, in such a way that GA from GS is the underlying algorithm, with this difference that it employs the ALPSOFV algorithm in place of the CrossOver function. In each step, the mutation function receives the best value and 50% of that is re-initialized randomly.
- The weight of a test case equals the number of cases covered, and the heaviest one is selected. If the test cases are the same, a test case that is more different from the previous test cases is chosen by the solution used in the DPSO [7]. And this process continues until the complete coverage.
- Finally, after the test suite production, the number of test cases is reduced by a simple minimizing solution [11].

The rest of this paper is organized as follows. Section 2, titled "Background", provides an initial understanding of CA, GA, and ALPSOFV algorithms. Section 3 is devoted to the solutions presented in the field of CA. Besides, computational and artificial intelligence strategies are discussed in section 3. Section 4 describes precisely the steps of implementing the proposed strategy and defines the parameters of GA and ALPSOFV algorithms. The comparison results between the proposed solution and other available strategies are given in section 5. Section 6 is dedicated to the conclusion and future work.

2. Background

2-1. Covering array

CA is demonstrated in two forms: if the values of all parameters are identical, it is shown as CA (N; t, vp), where v and p is the value and the number of parameters, respectively, t is the t-way interaction strength (2 ≤ t ≤ p), and N is the number of selected test cases that form the complete coverage for the interaction strength t. But if the parameter values are not identical, then it is represented by CA (N; t, p, v), where v is a vector that stores the values of parameters in order. The main goal in generating a covering array is to minimize N; that is, the smaller the N, the stronger the strategy of test case selection.

Here for further explanation of CA, the example of a travel agency [8] is presented. This system consists of five components: client, travel agency, bank, airline, and hotel. The relationships between these components are depicted in Figure 1. Each of these components is known as an independent parameter and can take one or more values. Table 1 lists these parameters and their corresponding values. The 5-tuple (Sajad, Rakhsh, Aseman, Golestan, Mellat) constitutes a typical test case in this system.

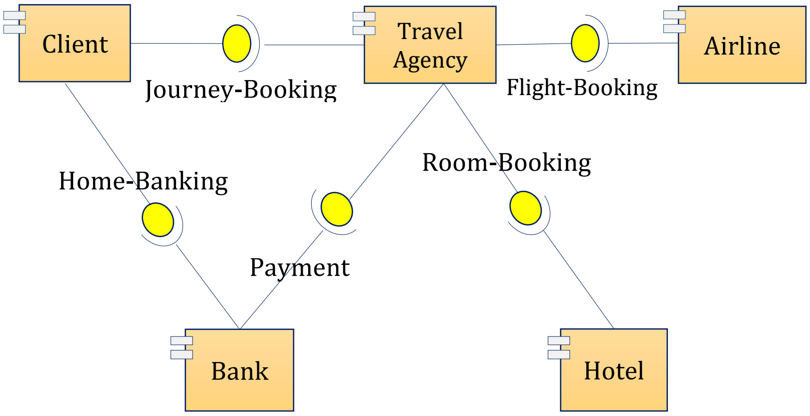


Figure 1: The travel agency components [2].

Table 1
The values of e-travel agency components [2].

Client	Travel Agency	Airline	Hotel	Bank
Vahid (0)	Rakhsh (0)	Zagros (0)	Parsian (0)	Melli (0)
Sajad (1)	Safiran (1)	Caspian (1)	Golestan (1)	Pasargad (1)
		Aseman (2)		Mellat (2)

For example, suppose Sajad (client (1)) is not allowed to perform banking transactions with Mellat Bank. So the system error is triggered when the client and bank parameters have the values "Sajad" and "Mellat", respectively. To identify this error, the state (Sajad, -, -, -, Mellat) must be covered at least once by the test suite. This state can be represented as (1, -, -, -, 2). In an exhaustive test, all possible states of the parameters must be covered. In this example, we need to produce 3 × 2 × 3 × 2 × 2 = 72 test cases. As the number of parameters and their values grow, the cost of testing increases accordingly. In addition, because the interaction of a limited number of parameters causes an error [9], the production of a full test is not cost-effective at all, and it is recommended to apply some techniques, such as CA, to solve this problem.

As indicated, all scenarios of the Travel Agency example are 72 test cases. By applying a 2-way (t = 2) test, the total number of test cases decreases to 9, as shown in

Table 2. This table contains all possible combinations of every two parameters. For example, there are 2 × 3 = 6 different binary states for the two parameters of bank and hotel: (-, -, -, Parsian, Melli), (-, -, -, Golestan, Melli), (-, -, -, Parsian, Pasargad), (-, -, -, Golestan, Pasargad), (-, -, -, Parsian, Mellat), and (-, -, -, Golestan, Mellat), all of which can be seen in

Table 2. In general, for the configurations whose parameters have equal values, the total number of coverage is calculated by equation (1), otherwise, equation (2) is used.

$$Max\_Coverage = \binom{p}{t} * v^t \qquad (1)$$

$$Max\_Coverage = \binom{p}{t} * |v_1| * |v_2| * \ldots * |v_p| \qquad (2)$$

Table 2
Test suite for CA (9; 2, $2^3 3^2$) [2].

No.	Client	Travel Agency	Airline	Hotel	Bank
1	Vahid(0)	Rakhsh (0)	Zagros (0)	Golestan (1)	Mellat (2)
2	Sajad (1)	Safiran (1)	Aseman (2)	Parsian (0)	Mellat (2)
3	Vahid(0)	Safiran (1)	Caspian (1)	Golestan (1)	Melli (0)
4	Sajad (1)	Rakhsh (0)	Caspian (1)	Parsian (0)	Pasargad (1)
5	Vahid(0)	Rakhsh (0)	Aseman (2)	Parsian (0)	Melli (0)
6	Sajad (1)	Safiran (1)	Zagros (0)	Golestan (1)	Pasargad (1)
7	Vahid(0)	Rakhsh (0)	Aseman (2)	Golestan (1)	Pasargad (1)
8	Sajad (1)	Safiran (1)	Zagros (0)	Parsian (0)	Melli (0)
9	Sajad (1)	Rakhsh (0)	Caspian (1)	Parsian (0)	Mellat (2)

2-2. Genetic algorithm

One of the successful algorithms in optimization is the Genetic Algorithm (GA) [15]. Charles Darwin has a famous quote: "It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is most adaptable to change." In fact, the whole concept of the genetic algorithm is based on this quote. For a better understanding, look at the example below:

Suppose you are a city leader and want to keep your city safe from bad events. So follow this policy:

- Select all the good people in the city and ask them to expand their generation with childbearing.
- Repeat this for several generations.
- Now, the entire city population is good people.

This example is impossible in the real world, and it is given here only to help you understand the subject. So the fundamental idea is to change the input in order to obtain better output. Since the genetic algorithm is partly related to biology, to study the relationship between these two, a few new concepts are presented.

Let's return to the primitive example and review this in summary.

- First, the initial population is defined as our compatriots.
- A function is defined to determine whether a person is good or bad.
- Good people are selected for mating and childbearing.
- At last, bad people are replaced with children (off-springs), and this process is repeated.

The steps above describe how the genetic algorithm works, and essentially tries to somehow mimic human evolution. It can be said that GA is an optimization technique and aims to find out the inputs that produce the best outputs (results).

2-3. Particle swarm optimization

One of the most well-known population-based algorithms is the particle swarm optimization algorithm (PSO) [15]. This algorithm is inspired by the behavior of animals that live in swarms (groups) and is used in optimization problems. Particles form problem solutions and compete to achieve a better position in the swarm. Since the initial version of PSO algorithm was proposed, multiple variants of this algorithm have been presented, some of which are ALPSOFV [16], Complex-Order PSO (CoPSO). [17], etc. In this paper, we utilize ALPSOFV, which is briefly described in the following.

The ALPSOFV algorithm, called FoPSO in [17], uses a fractional derivative to better traverse the search space. In order not to lengthen the paper, we do not explain it in details and only outline its steps in summary:

1. r = 0, and initialize the population at random.
2. According to equation (12), calculate L for each particle.
3. If the final criterion is satisfied, then B = B1 and terminate the algorithm.
4. If the counter is in the range of [0, kmax], then update L values.
5. Update the values of A and B using equations (14) and (15), k = 0, and J = J + 1.
6. Resume the algorithm from step 2.

3. Related work

In general, CA generation methods can be classified into four basic categories: (1) mathematical approaches, (2) greedy approaches, (3) artificial intelligence approaches, and (4) stochastic approaches. These four categories are discussed below.

3-1. Mathematical approaches

A mathematical approach exploits an orthogonal array (OA) structure to generate a CT. This technique can only be applied to generate the test suite for specific configurations. Combinatorial Test Services (CTS) [13] and Test Configuration (TConfig) [19] are two well-known mathematical approaches.

3-2. Greedy approaches

In general, computational approaches take advantage of one-test-at-a-time (OTAT) and one-parameter-at-a-time (OPAT) methods for CA generation. In the former method, one or a set of test cases are chosen then the test case that covers the most is looked for. But in OPAT, the CA is generated incrementally in two directions: horizontal and vertical. In horizontal expansion, each time one component is inserted into the CA then a coverage check is conducted and the best value for that component is taken. If after the horizontal expansion some interactions are not covered, the vertical expansion begins to cover them. The well-known strategies for OTAT method include Automatic Efficient Test Generator (AETG) [15], mAETG [16], Pairwise Independent Combinatorial Testing (PICT) [17], Deterministic Density Algorithm (DDA) [18, 19], CTE-XL [5, 6], Test Vector Generator (TVG) [20, 21], Jenny [22], GTWay [23] and Intelligent Test Case Handler (ITCH) [24].

The AETG strategy is the first OTAT-based strategy that selects test cases in a greedy manner and adds them to the test suite to complete the coverage. This strategy is later developed as mAETG [16] and mAETG-sat [9]. Another OTAT-based strategy that greedily chooses test cases is PICT [17]. This random-based strategy usually does not appear to be efficient enough, but acts well in terms of performance. TVG [20] is also a strategy from OTAT family, which is available like PICT. The strategy uses T-Reduce, plus-one, and Random Sets algorithms to produce a test suite, among which T-Reduce has better outcomes than its competitors. Jenny is another available OATA-based strategy that produces acceptable results from an efficiency standpoint and can support high interaction strength. It can be said that the most efficient and purely computational strategy is ITCH [24]. In this method, a test suite is generated by carrying out an exhaustive search. In some configurations, this strategy even outperforms AI-based strategies in terms of efficiency, but it has poor performance and can produce a test suite up to t = 4.

Among the strategies mentioned above, the only strategy that generates the test suite using the Classification-Tree Method (CTM) is CTE_XL [5]. This strategy often produces test suites up to t = 3, and is not good regarding efficiency and performance. The latest pure computational strategy based on OTAT is GTWay. GTWay has good results in efficiency and performance. It is capable of producing test suites up to t = 12.

IPOG [4], IPOG-s [25], and IPOG-D [26] are commonly used strategies that are developed based on OPAT. IPOG and IPOG-D strategies are presented in the ACTS tool [27], and are the best in CA generation, concerning the performance.

3-3. Artificial intelligence (AI)-based approaches

As the most popular methods for CA generation, AI-based algorithms often use the OTAT method to extend the test suite. A large number of AI-based algorithms are employed to generate CA, some of which found in qualified journals and conferences include Simulated Annealing (SA) [28, 16], Tabu Search (TS) [28], GA [28, 2, 29], Ant Colony Algorithm (ACA) [29], PSO [30, 7], Cuckoo Search (CS) [12],Teaching Learning-Based Optimization (TLBO) [31], and Harmony Search (HS) [23].

CA generation by AI algorithms was first proposed by Stardem [28] by the means of three algorithms SA, GA, and TS. These strategies could only produce 2-way interactions. Among them, SA outperforms both GA and TS strategies, and GA, in turn, acts better than TS. The increasing development of software systems requires the use of higher interactions. Cohen proposed SA in [16], and Shiba offered GA in [29], to support interactions up to t = 3. These strategies have far better outcomes than computational strategies. These strategies reduce the number of test cases by employing complex algorithms, which results in lower production speeds; and also they cannot support high interactions.

Particle Swarm-based t-way Test Generator (PSTG) [30] strategy can support up to t = 6 by eliminating complex algorithms, accelerating the calculation of test cases weights as well as providing a data structure to store uncovered states. It seems that PSTG considers a matrix for each combination, the index of which is used to detect the coverage or non-coverage of the test case when calculating the test case's weight. By changing the PSTG data structure, CS [12]could enhance CA generation performance, but is not superior to PSTG in terms of efficiency because it supports up to t = 6. Another robust strategy for CA generation is Harmony Search Strategy (HSS) [23], which is based on the Harmony Search algorithm. This strategy supports up to t = 15 and has far better results than PSTG in terms of efficiency, but is not evaluated for its performance.

HHH [32] is the first strategy that uses High-Level Hyper-Heuristic. Instead of a single meta-heuristic algorithm, four meta-heuristic algorithms are utilized in this strategy, namely Teaching Learning Based Optimization (TLBO), Global Neighborhood Algorithm (GNA), Particle Swarm Optimization (PSO), and Cuckoo Search (CS). Actually, this strategy uses Tabu search as its underlying algorithm, and in each step, based on the three operators including improvement, diversification, and intensification, chooses one of those four algorithms to generate the test case. HHH performs greatly from an efficiency viewpoint.

The main issue of CA generation by particle swarm-based algorithms is that applying velocity does not necessarily produce better results. DPSO strategy provides a very effective approach to overcome this problem. Among the existing strategies, DPSO is the most efficient one but lacks good performance. High interactions are supported in the implementation of this strategy, but due to the one-day time-out for each configuration, this strategy supports up to t = 10 [2]. Another strategy presented in [7] is Conventional PSO (CPSO) which is more powerful than DPSO with regard to performance, but DPSO is more efficient. These two strategies are available in [33].

The combination of Fuzzy and AI algorithms shows very good performance results in Fuzzy Self-Adaptive PSO (FSAPSO) [34], Adaptive TLBO (ATLBO) [31], and Fuzzy Inference Selection [3]. These three solutions combine fuzzy algorithm with PSO, TLBO, and Hyper-Heuristic, respectively. Applying Fuzzy has a negative impact on the performance of these strategies, but in some configurations, they produce better results than DPSO in terms of efficiency. These strategies often support up to t = 4.

GS strategy is also another strategy that supports CA. This strategy utilizes bit structure and changes GA to produce more appropriate results than AI-based strategies in terms of time, and can support up to t = 20.

The strategies presented in [35] and [36] are based on Hybrid Flower Pollination and Q-Learning Sine Cosine Algorithm (QLSCA), and they aim to increase efficiency; which provide desirable results in this regard. The results show that these strategies support up to t = 4.

Another hyper-heuristic-based algorithm is Q-EMCQ [43]. This paper presents two approaches, EMCQ and Q-EMCQ. Q-EMCQ (based on Q-Learning mechanism) is more efficient than EMCQ but EMCQ is superior in terms of performance. In general, Q-EMCQ and DPSO are very close from an efficiency standpoint.

3-4. Stochastic approaches

This method randomly selects test samples using the input distribution. TVG algorithm generates test suites by applying four methods, one of which randomly selects test samples.

4. Implementation

In this paper, the combination of ALPSOFV and GA algorithms is used for CA generation. The implementation steps are described below.

4-1. Creating the initial population (Step 1)

At this step, the parameters for the ALPSOFV and GA are initialized, and also the initial population is randomly generated. First, in GA, Chromosome encoding uses three primary methods, namely binary encoding when genes accept only 0 and 1, permutation encoding when the permutation between genes is needed, and value encoding when genes have numeric and string values. In this research, the value encoding method is employed to build chromosomes. Each chromosome is equivalent to a particle in the ALPSOFV and represents a test case.The production details of each Chromosome are described in Step 3.

4-2. Creating a covering Matrix (Step 2)

An important and effective factor in CA generation is the calculation of the test case's weight. To do this, first, all possible states required for full coverage (equations 1 and 2) should be stored. Various data structures are proposed for this purpose. In this study, the data structure of [2] is employed. The number of rows in this structure is $\mathrm{C}_{\mathrm{t}}^{\mathrm{p}}$. Each row consists of two parts: the first stores the combination number and the second contains$|v_1|*|v_2|*\ldots*|v_t|$ cells, which are 0's (i.e. no coverage) by default. For a more detailed explanation, we provide an example here. Suppose a system with 6 parameters of values 2, 2, 2, 2, 5, and 2, and t = 2. In the first part, values 1 and 2 (combinations 1 and 2) are placed in the first row; and the number of cells in the second part equals $|v_1|*|v_2| = 2*2 = 4$. The second and third rows have the same conditions except that in the first part, the values (1, 3) and (1, 4) stand for the combination of the first and third parameters and the combination of the first and fourth parameters, respectively. The first part of the fourth row is (1, 5), and the second part is $|v_1|*|v_5| = 2*5 = 10$. Other rows are also created in a similar fashion. Figure 2 (A) illustrates the coverage for the example above.

For the weight calculation, the covering matrix should be explored row by row. Using the first part of each row, the needed values are extracted from the test case, and after calculating the decimal equivalent, the corresponding cell is checked. If this value is zero, the cell's value changes to one, and one unit is added to the weight. This procedure continues until the last row. Here, we describe these steps by an example. For a system with six parameters, the test case (0, 1, 0, 1, 4, 0) is considered. In the first row, the values 1 and 2 are stored. According to these two values, the first and second cells of the test case (0, 1) are chosen (Figure 2 (B)) and their decimal equivalents are calculated; thereafter, the corresponding cell (the third cell), which is zero, changes to one, and one unit is added to the weight (Figure 2 (C)). Each row has at most one covering. In the initial state, since no state is covered, all combinations of test cases are included; so the first test case can be added randomly to the test suite. The maximum weight of a test case is equal to the number of rows in the covering matrix, thus the first test case weighs $\mathrm{C}_{\mathrm{t}}^{\mathrm{p}}$.

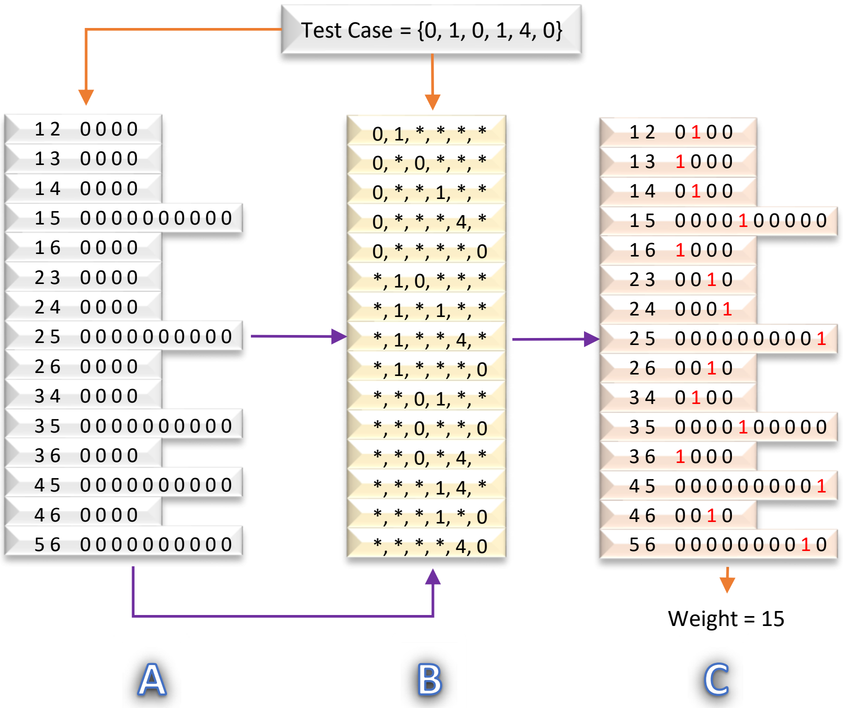


Figure 2: The covering matrix and steps to calculate the weight of test case {0, 1, 0, 1, 4, 0} [11]

4-3. Generating CA by combining ALPSOFV and GA (Step 3)

There are many different solutions, such as [44, 45], that use a combination of meta-heuristic algorithms to solve their problem. But in this study we employ a combination of two algorithms GA and ALPSOFV. After generating the initial population in the genetic algorithm, parents are selected and provided as inputs for crossover and mutation functions for childbearing. In this approach, the crossover is replaced with the ALPSOFV algorithm. The test case with the highest weight is selected (as *the best*) and given to the ALPSOFV. This test case takes the place of gbest (global best), and the full ALPSOFV algorithm is executed on this test case and returns a modified test case; if the result is improved, *the best* is updated. Then *the best* is passed to the mutation function. It receives *the best* and randomly selects half of the genes and replaces them with random values.The implementation details are presented below. The first step of the genetic algorithm is initialization. It is assumed that the chromosomes are multiple strands. Our initial population is as shown in Figure 3 (A). The values of genes are randomly selected. Each test sample is equivalent to one chromosome in the genetic algorithm, each GEN of which also represents a system parameter. For example, the chromosome of a system with 8 parameters has 8 GENs. To reduce storage and search space, we convert the values of each test sample into a number. For instance, we convert the test sample (Vahid, Rakhsh, Aseman, Golestan, Mellat) to (0, 0, 2, 1, 2) (Figure 3 (B)).

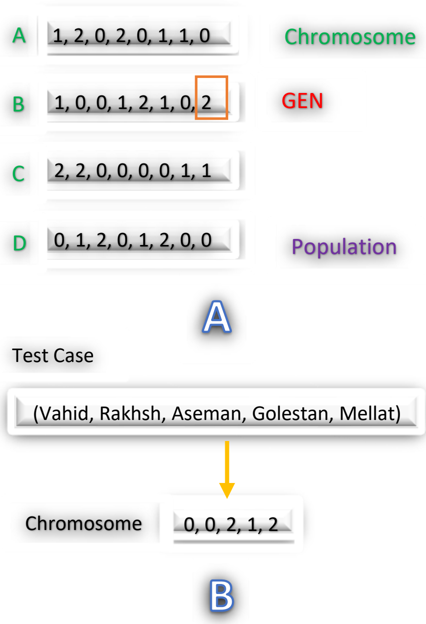


Figure 3: Initial population in GA

After generating CM, it is time to calculate the weight of the chromosome. In fact, at this stage, each chromosome is sent to a fitness function, and this function returns the value of the chromosome as the output (the thorough description of these two functions is given in step 2). The next step is the selection function. At this point, a number of chromosomes (parents) are selected for parenting. The children being born are known as offsprings. There are several methods for selection including roulette-wheel selection, truncation selection and tournament selection [2]. Here, we employ tournament selection to select test samples. In the previous step, the parents are selected. Now a new child is to be created with the crossover function. In this solution, we use the ALPSOFV algorithm instead of the crossover function. In other words, in this step, the ALPSOFV algorithm is applied to a fraction (i.e. defined by the crossover rate) of the population. If the population is not generated randomly, then manipulation of this population may lead to getting stuck in a local optimum. To overcome this problem, at each step, we compare gbest with a random test sample and update gbest in case of an improvement. Finally, the best test sample generated in this step is provided for the mutation function.

Do children always have the same characteristics as parents? Of course not. During the growth period, changes occur in the genes of the children that make them different from their parents. This process is called mutation. A simple mutation method is shown in Figure 4 (A)Figure 4. The children produced in this way are again evaluated using the fitness function and, if they are desirable, then they replace the previous ones. And this routine continues until the goal is achieved. However, since the CA generation problem strongly tends towards a local optimum, modifying just a single cell does not have a significant effect on improving the test sample. So in this approach, at each step, half of the genes are randomly selected and assigned a random value in the range of [0, v] (Figure 4 (B)).

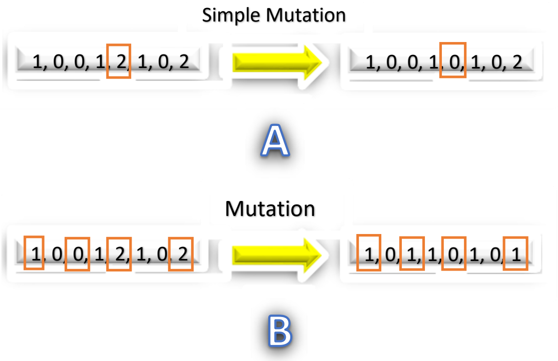


Figure 4: A simple mutation in GA

The higher the weight of a test case, the greater the value of that. However, test cases of equal weight may not have the same value. An example given in [7] confirms this fact. This example is for the configuration CA (N; 2, 3^4) and its two test suites are represented in Table 3. As can be seen, the first test suite (Sequence 1) consists of three primitive test cases of weight 6. And the three test cases in the second test suite (Sequence 2) also have the same status. But there is a difference in the selection of the next test case. In Sequence 1, it is not possible to have a new test case of weight 6, but in Sequence 2, the test case (1 0 1 2) weighs 6. In general, to select test cases of equal weight, it is advisable to select a test case that has the least difference from its previous test case.

Table 3: Two different sequence of CA(N; 2, 3^4) [7]

#	Sequence 1	Sequence 2
1	0 0 0 0	0 0 0 0
2	1 1 1 1	0 1 1 1
3	2 2 2 2	0 2 2 2
4	-	1 0 1 2

4-4. Minimizing the test suite (Step 4)

In OTAT-based solutions, in each step, a test case is added to the final test suite. Typically, this test case has the largest number of coverings compared with other test cases. In addition to the new coverings, the test case also includes some previously covered states. By increasing the number of test cases, all states covered by a test case may be covered by other test cases too; thus this test case is redundant and should be eliminated [11]. To comprehend the test suite more precisely, consider Figure 5 (A). First, the weight of the test cases should be re-calculated, but here the weight means the number of states that the test case can cover on its own. For example, row one has the value (*, *, 0, 0, *), which is also seen in row four; so no weight is considered for it. But there are (1, *, 0, *, *) and (*, 1, *, 0, *) in the same row, which do not appear in any of the test cases. As a result, the weight of this test case is 2 and it cannot be deleted. However, all six states in row four are covered by other test cases and the weight of this test case is practically zero and can be removed (Figure 5 (B)). The pseudocode in Figure 6 presents the steps of CA generation in the proposed strategy and the overall steps of the proposed algorithm are illustrated in Figure 7.

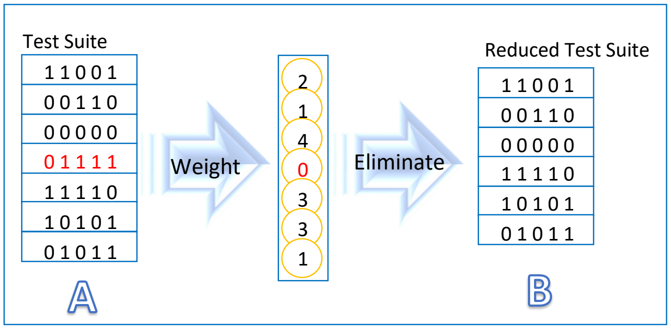


Figure 5: Elimination of redundant test cases [11].

```
Initialize the variables
Get the configuration from the input
Create the covering matrix (CM)
Select randomly a chromosome and add it to the final test suite (TS)
Update the covering matrix
While (Max_Coverage >0)
    Initialize randomly the population
    Calculate the weight of each chromosome
    Select the best value in the population (gbest)
    For (i = 0, i < crossrate)
        Apply the ALPSOFV algorithm to the population and update gbest
    For (i = 0, i < muterate)
        Apply the mutation algorithm to the selected population and update gbest
    If (fitness (gbest) > 0)
        Add gbest to TS and update CM
End While
```

Figure 6: The pseudocode for the proposed solution

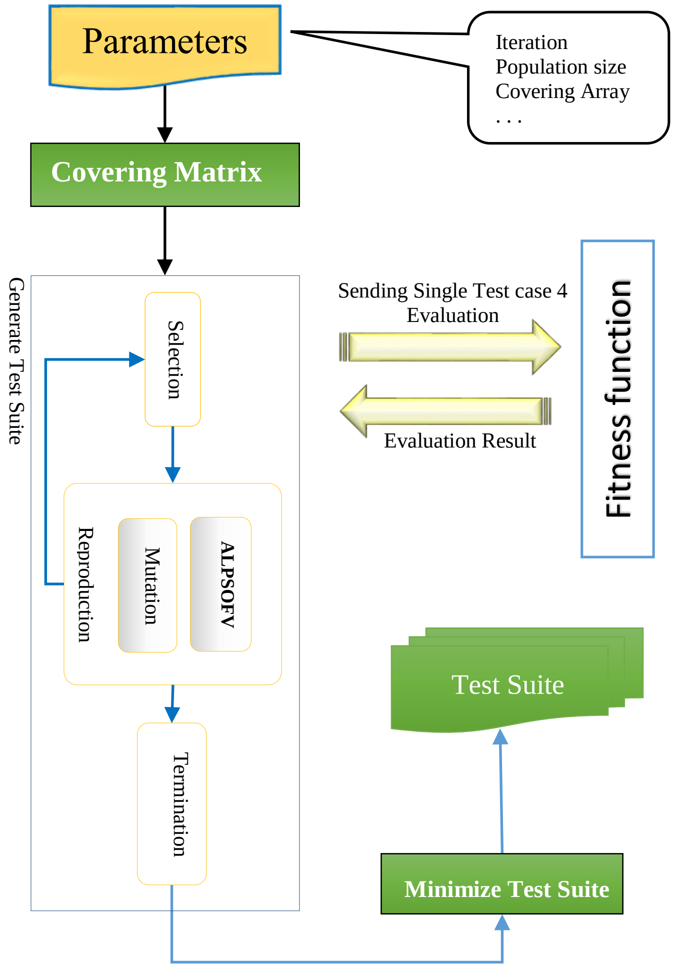


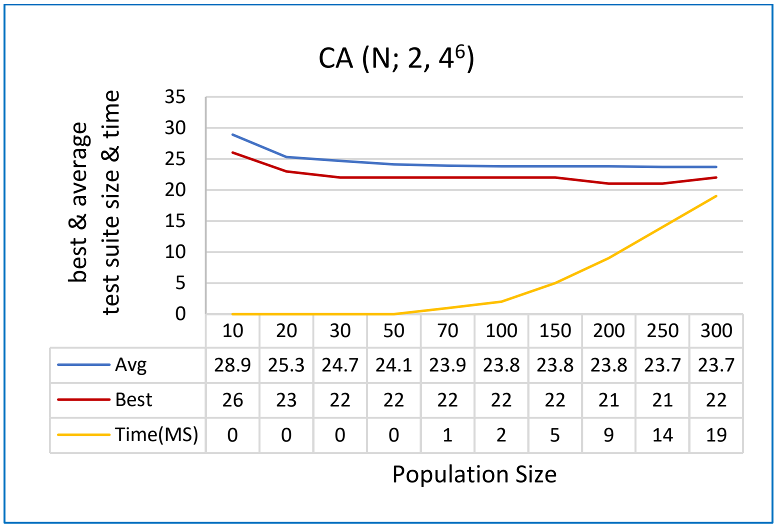
Figure 7: Flowchart of the purposed approach

4-5. Time and space complexity

Like the solutions in [46], the proposed approach requires the analysis of time and space complexity. Since the weight calculation function needs to store the test sample in the memory and this memory has $\binom{p}{t}$ rows and $v^t + t$ columns in the covering matrix, so the space complexity of the proposed solution is $O(\binom{p}{t} * v^t)$. After being completely covered, that row is removed from the memory, so gradually the number of rows in the matrix tends towards O(1), and consequently the space complexity approaches $O(v^t)$. To calculate the weight of a test sample, all rows in the matrix must be traversed, and each row requires a conversion from binary to decimal, which has an order of $O(t)$ time complexity, so as a result, the time complexity of this solution equals $O(\binom{p}{t} * t)$ and tends towards $O(t)$ by eliminating the covered rows.

4-6. Parameter setting

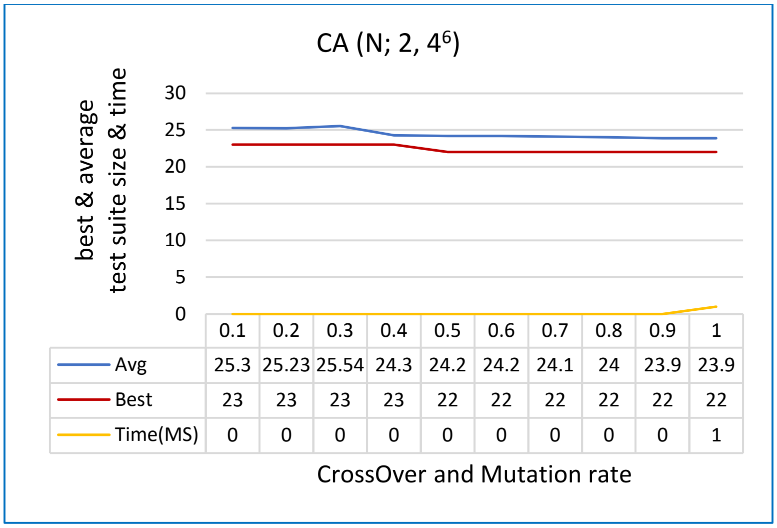
One of the obscure issues in meta-heuristic algorithms is the exact selection of parameters. In this paper, two PSO and GA algorithms are combined, so that the parameters of both should be evaluated. The PSO algorithm has 4 parameters C1, C2, W, and number of repetitions. The effect of changing the three parameters C1, C2, and W are like [35], and it is not studied and explained in detail here in order not to lengthen the paper. The main determinant parameter in the optimal covering array generation is popsize (i.e. population size). Due to the fact that the size of the search space depends mainly on the configuration size, the selection of popsize value is directly related to the configuration size; and the larger the configuration is, the higher the value of popsize should be. Also in the genetic algorithm, in addition to population size, the crossover and mutation rates, which range from 0 to 1, play significant roles, too. In the proposed solution, the crossover and mutation rates are the same. First, a minimum value of 0.1 is considered for crossover and mutation rates. By increasing popsize, its impact on the CA (N; 2, 4^6) configuration emerges. The results of this study are illustrated in Figure 8. As seen in this figure, the best value is for the test suite with 21 test cases, where popsize = 200; but the mean is still improved up to popsize = 300. Also, as the population grows, test suite production takes longer. Time is expressed in milliseconds.



	10	20	30	50	70	100	150	200	250	300
Avg	28.9	25.3	24.7	24.1	23.9	23.8	23.8	23.8	23.7	23.7
Best	26	23	22	22	22	22	22	21	21	22
Time(MS)	0	0	0	0	1	2	5	9	14	19

Figure 8: The best and average test suite size and time for CA (N; 2, 4^6)

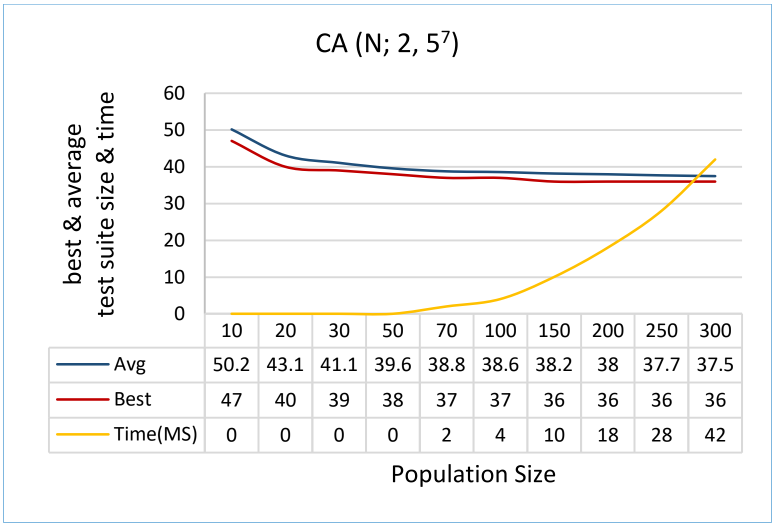
The next study focuses on the effect of increasing crossover and mutation rate, and its results are shown in Figure 9. In this evaluation, the value of popsize is considered constant (popsize = 20). The best value for these two parameters in the configuration CA (N; 2, 4^6) is 0.6. The time spent on test suite production is close to zero in this evaluation.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Avg	25.3	25.23	25.54	24.3	24.2	24.2	24.1	24	23.9	23.9
Best	23	23	23	23	22	22	22	22	22	22
Time(MS)	0	0	0	0	0	0	0	0	0	1

Figure 9: The best and average test suite size and time for CA (N; 2, 4^6)

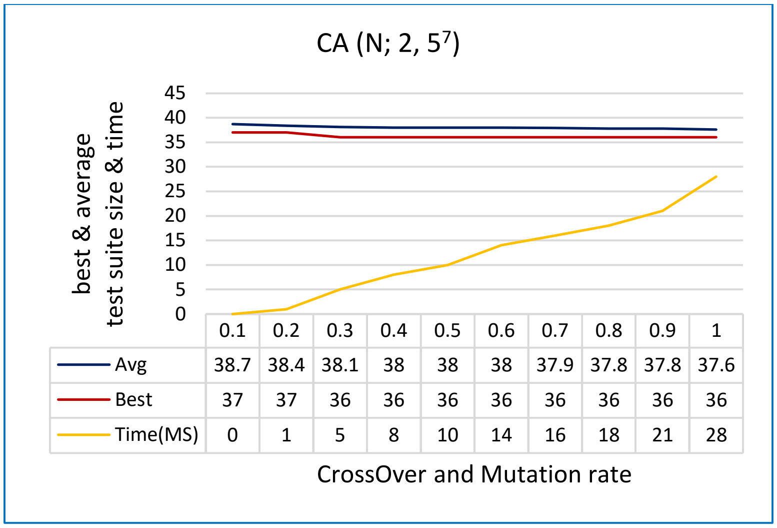
Suppose the configuration CA (N; 2, 5^7). At first, the impact of increasing popsize is evaluated. The values of the two parameters, namely crossover and mutation rates, equal 0.1, and the popsize variations range between 10 and 300. As demonstrated in Figure 10, population growth has a great influence on the execution time of the algorithm. The best value for this configuration is for popsize = 150; but for larger populations, the average is better.



	10	20	30	50	70	100	150	200	250	300
Avg	50.2	43.1	41.1	39.6	38.8	38.6	38.2	38	37.7	37.5
Best	47	40	39	38	37	37	36	36	36	36
Time(MS)	0	0	0	0	2	4	10	18	28	42

Figure 10: The best and average test suite size and time for CA (N; 2, 5^7)

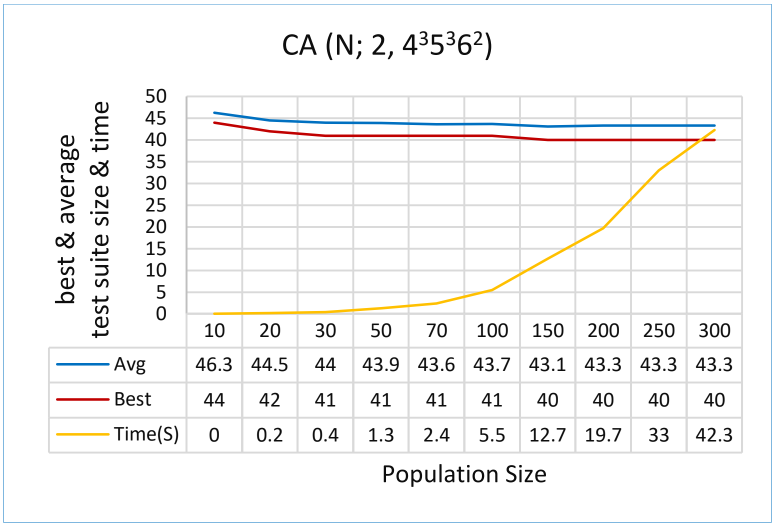
In the further study of the configuration CA (N; 2, 5^7), the effect of changing the crossover and mutation rates is investigated. Here, popsize = 70 and the crossover and mutation rates range between 0.1 and 1. Concerning Figure 11, it can be concluded that the increase in these rates affects productivity growth. As can be seen, the impact of the crossover and mutation rates on time is less than that of population increase. The appropriate value for these two rates in CA (N; 2, 5^7), given the best value, should be greater than 0.3, and if the average is important, then the maximum value can be taken into account for this value.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Avg	38.7	38.4	38.1	38	38	38	37.9	37.8	37.8	37.6
Best	37	37	36	36	36	36	36	36	36	36
Time(MS)	0	1	5	8	10	14	16	18	21	28

Figure 11: The best and average test suite size and time for CA (N; 2, 5^7)

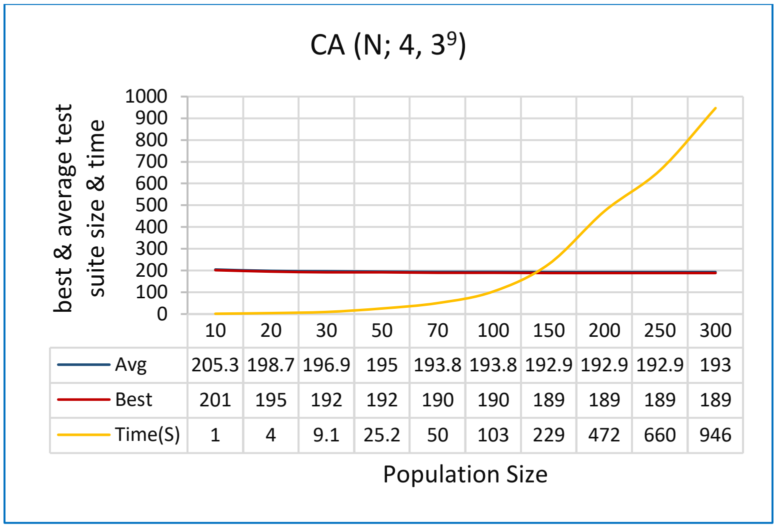
Increasing popsize and rates have a positive impact on productivity improvements, but as seen before, population growth has a more negative impact on efficiency; hence, an upper limit is set on rates (i.e. 0.8) and we try to explore the impact of increasing the popsize which is the most important parameter in the proposed strategy. The next evaluation is dedicated to the configuration CA (N; 2, $4^3 5^3 6^2$) and its results are conveyed in Figure 12. In this configuration, the power time is long, and the results are as in Figure 12 (in seconds). The best performance of the proposed solution is the test suite with 40 test cases that occurs for popsize = 150, so it can be concluded that this is satisfactory for the mentioned configuration.



	10	20	30	50	70	100	150	200	250	300
Avg	46.3	44.5	44	43.9	43.6	43.7	43.1	43.3	43.3	43.3
Best	44	42	41	41	41	41	40	40	40	40
Time(S)	0	0.2	0.4	1.3	2.4	5.5	12.7	19.7	33	42.3

Figure 12: The best and average test suite size and time for CA (N; 2, $4^3 5^3 6^2$)

The last evaluated configuration in this section is CA (N; 4, 3^9). The results of this evaluation are plotted in Figure 13. As can be seen, the best performance of the proposed solution is for the test suite with 189 test cases which is obtained with popsize = 150. The time in Figure 13 is expressed in seconds.



	10	20	30	50	70	100	150	200	250	300
Avg	205.3	198.7	196.9	195	193.8	193.8	192.9	192.9	192.9	193
Best	201	195	192	192	190	190	189	189	189	189
Time(S)	1	4	9.1	25.2	50	103	229	472	660	946

Figure 13: The best and average test suite size and time for CA (N; 4, 3^9)

5. Evaluation

The evaluation in this research is divided into three categories of efficiency (array size), performance (production time), and interaction strength [2]. Efficiency concerns the execution steps of the strategy. But performance has a direct relationship with and depends on the hardware and software platform [12]. There are many strategies for CA generation that checking all of them one by one is beyond the scope of this section. With a fair comparison, we try to evaluate a number of available strategies. Among the AI strategies, three strategies are selected including CPSO, DPSO, and GS. Among these three strategies, DPSO is the most efficient one. It should be noted that the strategies in [37] and strategies of [3] and [42] may have better results than DPSO. Also among non-artificial intelligence strategies, Jenny, TConfig, PICT, IPOG are chosen. The platform specifications for this evaluation are Windows 7 OS, 2.20GHz core ™ i7QM CPU, and 6GB RAM. Coding is done in the eclipse environment (jdk 1.8).

Non-artificial intelligence strategies are usually deterministic and running them multiple times makes no difference. In contrast, AI-based strategies are nondeterministic and the result changes with re-execution; so each configuration is executed one hundred times, and two values "best" and "average" represent respectively the best and the average of the hundred times executions for each configuration. Efficiency results for GS, CPSO, and DPSO strategies, if available, are taken directly from the related papers, otherwise, they are executed as many times as the proposed algorithm and results are acquired.

According to Section 4.6, when the configuration volume increases, the search space becomes larger, so we need more population to traverse the entire state space, and conversely, if the search space is narrow, a smaller population can give the optimal answer. As discussed in Section 4.6, for the configurations evaluated in this paper, we need a population with a number of chromosomes in the range of [10, 300], which proportionally requires crossover and mutation rates of 40% to 100% to be able to generate the minimum test suite at an appropriate speed. Table 4 lists the parameters of the proposed solution with their corresponding values.

Table 4: Parameter setting

Algorithm	Parameters	Value
GA	Population	10 - 300

	Crossover rate	40% - 100%
	Mutation rate	40% - 100%
	Selection method	tournament selection
	Crossover method	ALPSOFV
	Mutation method	Uniform
ALPSOFV	Kmax	10-20
	a	0.9
	r	4
	R1, R2	Random

5-1. Efficiency

The most significant criterion for evaluating the robustness of a strategy in CA generation is the efficiency of that strategy. This section gives a comparison among the proposed solution and other strategies, the results of which are outlined in Tables 5 - 13. The first evaluation is for t = 2 and its results are listed in Table 5. As can be seen, in this evaluation, 13 configurations are considered. Among them, Jenny in one case, IPOG in four cases, CPSO in four cases, DPSO in eight cases, and GS and GALP in eleven cases have the best outputs. However, to compare the proposed strategy with others, a statistical method, called Wilcoxon signed-rank sum, is employed. To compare efficiency between two strategies, this tool from SPSS produces two outputs regarding the given input data: test statistics and ranks. For two strategies A and B, ranks have three values: the number of samples in which A > B, A < B, and A = B. And test statistics are composed of z and Asymp. Sig. (2-tailed). The importance of z in this research is not visible. Asymp. Sig. (2-tailed) value ranges between 0 and 1; to put it simply, if this value is less than 0.05, it means that there is a semantic difference between A and B. The results of Wilcoxon test (Table 5) are given in Table 6. The first comparison is between the proposed strategy and Jenny. In the ranks section, there are three numbers 12, 0, and 1; where 12 is the number of cases in which GALP is smaller than Jenny (i.e. stronger), 0 is the number of cases in which GALP is larger than Jenny (i.e. weaker) and 1 is the number of cases in which the both are the same. In this row, Asymp. Sig. (2-tailed) value is less than 0.05, which implies a semantic difference between these two strategies. From these results, it is understood that GALP strategy is more powerful than Jenny. In this comparison, the proposed strategy does not have any semantic difference from GS and DPSO strategies, but by referring to the ranks, it can be concluded that our strategy has a slight superiority over the two others.

Table 5: Comparison of proposed solution's array size with that of other strategies at t = 2

	Jenny N	TConfig N	PICT N	IPOG N	CPSO N.Best	CPSO N.Avg	DPSO N.Best	DPSO N.Avg	GS N.Best	GS N.Avg	GALP N.Best	GALP N.Avg
CA (N; 2, 2^7)	8	7	7	7	7	7.54	7	7	6	7.33	6	7.25
CA (N; 2, 3^3)	9	10	10	9	9	9.82	9	9	9	9.52	9	9.63
CA (N; 2, 3^4)	13	10	13	9	9	10.3	9	9	9	9.92	9	10.26
CA (N; 2, 3^5)	14	14	13	15	11	13.07	11	11.53	11	12.36	11	12.79
CA (N; 2, 3^6)	15	15	14	15	14	14.93	14	14.5	13	14.48	13	14.75
CA (N; 2, 3^7)	16	15	16	15	15	15.47	15	15.17	14	15.54	14	15.26

CA (N; 2, 3^8)	17	17	16	15	15	15.93	15	16	15	16.65	15	15.61
CA (N; 2, 3^9)	18	17	17	15	16	16.63	15	16.43	15	16.99	15	16.67
CA (N; 2, 3^{10})	19	17	18	15	16	17.70	16	17.3	16	17.91	16	17.58
CA (N; 2, 3^{11})	17	20	18	17	16	17.90	17	17.7	16	18.38	16	17.60
CA (N; 2, 3^{12})	19	20	19	21	17	18.60	16	17.93	16	18.36	16	18.44
CA (N; 2, 4^7)	28	28	27	29	25	25.77	24	25.33	24	26.47	24	25.72
CA (N; 2, 5^7)	37	40	40	45	36	37.97	34	35.47	36	38.32	35	37.24

2.20GHzcore™ i7QM CPU, RAM6GB, Windows 7

Table 6: Wilcoxon signed-rank sum test for Table 5

	Ranks			Test Statistics	
	GALP<	GALP>	GALP=	Z	Asymp. Sig. (2-tailed)
Jenny - GALP	12	0	1	-3.096554937712151	0.001957835794325334
TConfig - GALP	13	0	0	-3.207328564668598	0.0013397388514189553
PICT - GALP	13	0	0	-3.2103209415265996	0.0013258684489075262
IPOG - GALP	8	1	4	-2.3925944182183168	0.016729722989272507
CPSO - GALP	7	0	6	-2.6457513110645903	0.008150971593502709
DPSO - GALP	4	1	8	-1.3416407864998738	0.1797124948789998 5
GS - GALP	1	0	12	-1.0	0.3173105078629141 5

The second evaluation investigates t = 3. For this evaluation, 14 configurations are considered, and the results are shown in Table 7. In general, IPOG in one case, CPSO in three cases, DPSO in nine cases, GS in four cases, and GALP in thirteen cases produce the best results. The output of Wilcoxon test (Table 7) is given in Table 8. The proposed strategy outperforms Jenny, TConfig, and PICT in all fourteen configurations. GALP beats IPOG in 13 configurations, and they perform alike in one configuration. CPSO appears weaker than the proposed strategy in eleven cases, and they produce the same results in three cases. Also, GS is weaker than GALP in ten cases, and their outcomes are the same in four cases. The proposed strategy has a semantic difference from all the strategies mentioned and the Asymp. Sig. (2-tailed) value confirms this claim. Like the previous evaluation, there is no semantic difference between the proposed strategy and DPSO; but regarding the ranks values, it can be concluded that the proposed strategy is provably stronger.

Table 7: Comparison of proposed solution's array size with that of other strategies at t = 3

	Jenny N	TConfig N	PICT N	IPOG N	CPSO N.Best	CPSO N.Avg	DPSO N.Best	DPSO N.Avg	GS N.Best	GS N.Avg	**GALP** N.Best	**GALP** N.Avg
CA (N; 3, 2^7)	14	16	15	16	12	14.63	15	15.06	12	14.59	12	14.59
CA (N; 3, 2^8)	14	18	17	18	16	16.12	16	16.21	14	16.62	12	15.76
CA (N; 3, 2^9)	17	20	17	20	16	16.2	16	1638	16	17.41	16	16.14
CA (N; 3, 3^4)	34	32	34	32	30	32.15	28	29.71	27	31.03	27	30.43
CA (N; 3, 3^5)	40	40	43	41	38	40.13	41	43.17	38	40.12	37	40.03
CA (N; 3, 2^{10})	18	20	18	20	16	16.56	16	16.52	16	16.9	16	16.64
CA (N; 3, 3^6)	51	48	48	46	42	45.53	33	38.3	43	44.65	40	45.5
CA (N; 3, 3^7)	51	55	51	55	49	51.13	48	50.43	49	52.71	48	50.92
CA (N; 3, 3^8)	58	58	59	56	53	55.27	52	53.83	54	57.17	52	54.3
CA (N; 3, 3^9)	62	64	63	63	58	59.40	56	57.77	58	61.36	56	58.02
CA (N; 3, 3^{10})	65	68	65	66	61	62.67	59	60.87	61	63.94	59	61.17
CA (N; 3, 3^{11})	65	72	70	70	63	65.6	63	63.97	63	65.82	62	64.26
CA (N; 3, 3^{12})	68	77	72	73	68	68.97	65	66.83	67	70.33	65	66.8
CA (N; 3, 4^7)	124	122	124	112	115	117.97	112	115.27	116	120.24	112	116.78

2.20GHzcore™ i7QM CPU, RAM6GB, Windows 7

Table 8: Wilcoxon signed-rank sum test for Table 7

	Ranks			Test Statistics	
	GALP<	GALP>	GALP=	Z	Asymp. Sig. (2-tailed)
Jenny -GALP	14	0	0	-3.3104749708770163	9.313778130529632E-4
TConfig -GALP	14	0	0	-3.301462400094654	9.618222539686696E-4
PICT -GALP	14	0	0	-3.3063692084502607	9.451346235069721E-4
IPOG -GALP	13	0	1	-3.2053382892273783	0.0013490382662570389
CPSO -GALP	11	0	3	-2.9605046065072935	0.0030713552860884468

DPSO -GALP	5	1	8	-0.9486832980505138	0.34278171114791145
GS -GALP	10	0	4	-2.8477920519317377	0.004402367998717294

Table 9 compares the proposed strategy with other strategies for t = 4. In this comparison, it is also clear that the AI-based strategies are more powerful than the computational ones. Here, the proposed strategy has the best performance in seven configurations. The results of Wilcoxon test in this comparison are shown in Table 10. As can be seen, the proposed strategy has no semantic difference just from two DPSO and GS strategies. It can be inferred from the ranks column that GALP is better than these two strategies.

Table 9: Comparison of proposed solution's array size with that of other strategies at t = 4

	Jenny N	TConfig N	PICT N	IPOG N	CPSO N.Best	CPSO N.Avg	DPSO N.Best	DPSO N.Avg	GS N.Best	GS N.Avg	**GALP** N.Best	**GALP** N.Avg
CA (N; 4, 2^{7})	31	36	32	35	24	29.64	31	32.37	27	29.70	24	30.42
CA (N; 4, 2^{8})	37	38	35	39	32	34.63	32	32.9	30	30.91	29	33.2
CA (N; 4, 2^{9})	37	41	41	41	33	36.42	34	35.33	33	34.12	25	35.71
CA (N; 4, 2^{10})	39	45	43	46	37	38.37	34	36.9	25	25.4	26	38.62
CA (N; 4, 3^{5})	109	97	100	97	94	97.62	81	83.63	88	89.5	88	100.02
CA (N; 4, 3^{6})	140	141	142	141	132	135.33	131	134.77	129	132.3	129	135.13
CA (N; 4, 3^{7})	169	166	168	167	153	157.47	150	155.23	152	154.8	152	156.72
CA (N; 4, 3^{8})	187	190	189	192	174	177.77	171	175.60	171	173.6	171	176.06
CA (N; 4, 3^{9})	206	213	211	210	191	195.47	187	192.27	187	190.4	190	193.25
CA (N; 4, 3^{10})	221	235	231	233	211	213.37	206	219.70	206	209.6	206	209.04
CA (N; 4, 3^{11})	236	258	249	251	226	229.3	221	224.7	223	226.41	220	224.13
CA (N; 4, 3^{12})	252	272	269	272	242	244.23	237	239.83	236	239.34	237	239.40

2.20GHzcore™ i7QM CPU, RAM6GB, Windows 7

Table 10: Wilcoxon signed-rank sum test for Table 9

	Ranks			Test Statistics	
	GALP<	GALP>	GALP=	Z	Asymp. Sig. (2-tailed)
Jenny - GALP	12	0	0	-3.065312218378251	0.002174428743498163

TConfig - GALP	12	0	0	-3.060589084265507	0.002209020346231392 4
PICT - GALP	12	0	0	-3.060589084265507	0.002209020346231392 4
IPOG - GALP	12	0	0	-3.0617678207175523	0.002200340526906490 5
CPSO - GALP	11	0	1	-2.942794549646885	0.003252641986305738 6
DPSO - GALP	6	3	3	-0.890870806374748	0.372998483613487 14
GS - GALP	4	3	5	-0.8574929257125441	0.391172522810139 53

The last evaluation is dedicated to t > 4, the results of which are presented in Table 11. In this comparison, TConfig can produce the test suite in less than 24 hours only in configurations CA (N; 5, 3^7) and CA (N; 6, 3^8), and it requires more time for the other cases, which are denoted by "> day" in Table 11. IPOG can produce up to t = 6 but does not support values larger than 6. DPSO also requires more than one day in five cases. But the other strategies support the higher interactions. Among these strategies, the proposed strategy emerges as the most powerful.The results of the Wilcoxon test are represented in Table 12.

Table 11: Comparison of proposed solution's array size with that of other strategies at t > 4

	Jenny N	TConfig N	PICT N	IPOG N	CPSO N.Best	CPSO N.Avg	DPSO N.Best	DPSO N.Avg	GS N.Best	GS N.Avg	**GALP** N.Best	**GALP** N.Avg
CA (N; 5, 3^7)	458	477	452	466	441	444	428	435	431	438.5	432	440
CA (N; 6, 3^8)	1466	1515	1455	1409	1397	1402	1402	1406	1398	1410.0	1392	1405
CA (N; 7, 3^9)	4746	>day	4618	NS	4422	4433	4427	4433	4437	4453.5	4425	4433
CA (N; 8, 3^{10})	14999	>day	14599	NS	13925	13944	13933	13939	13907	13943.0	13903	13923
CA (N; 9, 3^{11})	47009	>day	45521	NS	43587	43591	>day	>day	43808	43880.0	43545	43597
CA (N; 10, 3^{12})	147004	>day	141990	NS	135498	135498	>day	>day	136096	136109.5	135385	135464
CA (N; 11, 3^{12})	305797	>day	278993	NS	268173	268173	>day	>day	267630	2677690.0	267806	267852
CA (N; 12, 2^{14})	9422	>day	9112	NS	8882	8882	8972	8975	8890	8912.5	8904	8925
CA (N; 13, 2^{14})	13251	>day	12441	NS	11588	11371	>day	>day	10251	10283.5	11053	11068
CA (N; 14, 2^{15})	26579	>day	25036	NS	23889	23889	>day	>day	23377	23399.0	22645	22784
CA (N; 15, 2^{16})	53977	>day	51127	NS	45838	46423	>day	>day	46575	46598.5	41821	46095

Table 12: Wilcoxon signed-rank sum test for Table 11

	Ranks			Test Statistics	
	GALP<	GALP>	GALP=	Z	Asymp. Sig. (2-tailed)
Jenny - GALP	11	0	0	-2.9340578815309546	0.003345618115850882
TConfig - GALP	2	0	0	-1.3416407864998738	0.17971249487899985
PICT - GALP	11	0	0	-2.9340578815309546	0.003345618115850882
IPOG - GALP	2	0	0	-1.3416407864998738	0.17971249487899985
CPSO - GALP	9	2	0	-2.44625715840839	0.014434800209538572
DPSO - GALP	4	1	0	-1.4832396974191326	0.1380107375686596
GS - GALP	7	4	0	-0.9780192938436515	0.3280647817008368

5-2.Performance

The next comparison deals with the CA generation runtime. This section of the evaluation compares the proposed solution with AI-based strategies. As mentioned earlier, one of the factors influencing the CA generation in the proposed strategy is how the parameters of the algorithm are determined. For this comparison, popsize, the crossover rate, and mutation rate are set to 50, 0.8, and 0.8, respectively. Here, the production time of a test case is under consideration. For example, DPSO manages to produce a test suite with 14 test cases in 30 ms, as a result, 2.14 (i.e. 30/14) ms is spent for each test case. According to the results of this comparison shown in Table 13, the proposed solution is much stronger than AI-based strategies.

Table 13: Comparing performance between AI-based strategies available

	CPSO time	DPSO time	GS time	**GALP**
CA (2, 3^7)	0.28	2.14	0.41	0.09
CA (3, 3^7)	0.40	4.79	0.68	0.12
CA (4, 3^7)	0.82	9.86	0.66	0.12
CA (3, 3^8)	0.94	6.92	0.96	0.20
CA (3, 3^9)	1.35	9.82	1.21	0.33
CA (3, 3^{10})	1.82	13.72	2.02	0.53
CA (3, 4^7)	0.53	4.82	0.63	0.11
CA (3, 5^7)	0.57	7.96	0.57	0.11
CA (3, 6^7)	0.57	5.15	0.45	0.10
CA (3, 7^7)	0.58	3372	0.51	0.10

5-3. Interaction strength

Another measure for comparing the strategies in the field of CA generation is the interaction strength. The higher the interactions are covered by the algorithm, the more powerful the algorithm is. Among the strategies available in this paper, IPOG and TConfig are capable of generating test suits up to t= 6. Other strategies have the power to cover higher interactions. DPSO supports up to t = 12, Jenny and PICT support up to t = 15, and three CPSO, GS, and GALP strategies support over t > 15 interactions (Figure 14).

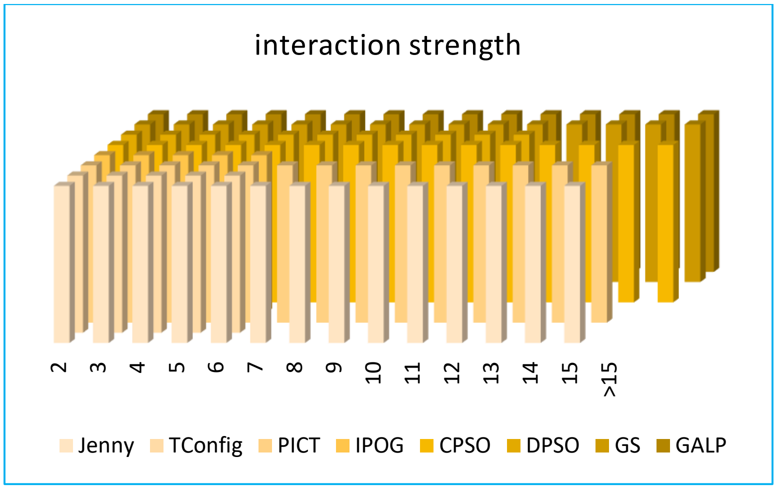


Figure 14: Maximum support for interaction strength

6. Conclusions and future work

AI-based strategies, due to their complex structure as well as their high iterations, have great results in the production of test suites. This complex structure has an inverse relationship with the production time and the interaction strength; the more complex the strategy is, the less interaction it produces in a higher time. For example, GA, SA, and ACO algorithms are often able to produce the test suite up to t = 3; PSTG and CS can support up to t =6 by reducing the complexity, changing the data structure, and increasing the speed; DPSO is capable of covering up to t = 12; HSS and CPSO can support up to t = 15. The DPSO and CPSO strategies are able to cover higher interactions but take more than one day to do this. DPSO is much more powerful than other AI-based strategies in terms of efficiency but does not have good performance. GS is more powerful than AI-based strategies in terms of time and interaction strength and covers up to t = 20, but has a weaker performance compared with DPSO. The computational strategies have a good performance as well, but they are less efficient than AI-based strategies.

In general, there is no strategy that works in both ways, that is, it covers high interactions and is robust in terms of efficiency and performance, as well. Therefore, in this research, we try to produce a test suite with good performance and efficiency by combining PSO and GA. In the proposed strategy, we take advantage of both DPSO and GS strategies, and integrate their strengths into a strategy, called GALP. Moreover, this strategy employs a simple and effective minimization function that improves efficiency. Although the GALP strategy has good results in terms of efficiency, performance, and interaction strength, this seems to be an ongoing challenge. The main issue with test suite production in CA is becoming trapped in a local optimum. Although the DPSO provides a proper solution, it seems that further research can be provided. As a final point, the OPAT method can also be utilized to further enhance the speed of AI-based strategies.

Compliance with ethical standards

Conflict of interest: Authors declare that they have no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

[1] J. Lin, C. Luo, S. Cai, K. Su, D. Hao and L. Zhang, "TCA: An Efficient Two-Mode Meta-Heuristic Algorithm for Combinatorial Test Generation (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Lincoln, NE, USA, 9-13 Nov. 2015.

[2] S. Esfandyari and V. Rafe, "A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy," *Information and Software Technology,* vol. 94, pp. 165-185, 2018.

[3] K. Z.Zamli, FakhrudDin, GrahamKendall and B. S.Ahmed, "An Experimental Study of Hyper-Heuristic Selection and Acceptance Mechanism for Combinatorial t-way Test Suite Generation," *Information Sciences,* vol. 399, pp. 121-153, 2017.

[4] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence, "IPOG: a general strategy for t-way software testing," in *4th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, IEEE Computer Society*, Tucson, AZ, 2007.

[5] E. Lehmann and J. Wegener, "Test case design by means of the CTE XL," in *8th European International Conference on Software Testing, Analysis & Review*, Copenhagen, Denmark, 2000.

[6] Y. Yu, S. Ng and E. Chan, "Generating, selecting and prioritizing test cases from specifications with tool support," in *Third International Conference on Quality Software*, Dallas, TX, USA, USA, 2003.

[7] H. Wu, C. Nie, F.-C. Kuo, H. Leung and C. J. Colbourn, "A Discrete Particle Swarm Optimization for Covering Array Generation," *IEEE Transactions on Evolutionary Computation,* vol. 19, no. 4, pp. 575-591, 2015.

[8] B. J. Garvin, M. B. Cohen and M. B. Dwyer, "An improved metaheuristic search for constrained interaction testing," in *1st International Symposium on Search Based Software Engineering*, Windsor, UK, 2009.

[9] J. Lin, S. Cai, C. Luo, Q. Lin and H. Zhang, "Towards more efficient meta-heuristic algorithms for combinatorial test generation," in *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*, 2019.

[10] P. Ramgouda and V. Chandraprakash, "Constraints handling in combinatorial interaction testing using multiobjective crow search and fruitfly optimization," *Soft Computing,* vol. 23, p. 2713–2726, 2019.

[11] S. Esfandyari and V. Rafe, "Extracting Combinatorial Test parameters and their values using model checking and evolutionary algorithms," *Applied Soft Computing,* vol. 91, pp. 1-19, 2020.

[12] V. Rafe, "Scenario-driven analysis of systems specified through graph transformations," vol. 24, p. 136–145, 2013.

[13] M. B. Cohen, M. B. Dwyer and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," in *international symposium on Software testing and analysis*, London, United Kingdom, 2007.

[14] M. Mitchell, "An introduction to genetic algorithms," *Cambridge, Massachusetts London, England, Fifth printing,* vol. 3, pp. 62-75, 1999.

[15] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *International Conference on Neural Networks*, Perth, WA, Australia, Australia, 1995.

[16] E. S. A. Shahri, A. Alfi and J. Machado, "Fractional fixed-structure H∞ controller design using Augmented Lagrangian Particle Swarm Optimization with Fractional Order Velocity," *Applied Soft Computing,* vol. 77, pp. 688-695, 2019.

[17] J. TenreiroMachado, S. M. Pahnehkolaei and A. Alfi, "Complex-order particle swarm optimization," *Communications in Nonlinear Science and Numerical Simulation,* vol. 92, 2021.

[18] A. Hartman, "Software and Hardware Testing Using Combinatorial Covering Suites," *Springer,* vol. 34, 2005.

[19] A. W. Williams, "Determination of Test Configurations for Pair-Wise Interaction Coverage," in *IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools and Techniques*, Kluwer, B.V., Deventer, The Netherlands, 2000, pp. 59–74, 2000.

[20] D. M. Cohen, S. R. Dalal, M. L. Fredman and G. C. Patton, "The AETG system: an approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering,* vol. 23, no. 7, pp. 437 - 444, 1997.

[21] M. B. Cohen, "Designing Test Suites for Software Interactions Testing," PHD Thesis, University of Auckland,Department of Computer Science ,Auckland,, 2004.

[22] J. Czerwonka, "Pairwise testing in real world: practical extensions to test case generator," in *24th Pacific Northwest Software Quality Conference, IEEE Computer Society*, Portland, OR, USA, 2006.

[23] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing: Research Articles," *Software Testing, Verification & Reliability,* vol. 17, no. 3, pp. 159 - 182 , 2007.

[24] R. Bryce and C. J. Colbourn, "A density-based greedy algorithm for higher strength covering arrays," *Software Testing Verification and Reliability,* vol. 17, no. 1, pp. 37-53, 2009.

[25] Y.-W. Tung and W. Aldiwan, "Automating test case generation for the new generation mission software system," in *2000 IEEE Aerospace Conference. Proceedings (Cat. No.00TH8484)*, Big Sky, MT, USA, USA, 2000.

[26] J. Arshem, "TVG download page," 2019. [Online]. Available: http://sourceforge.net/projects/tvg.

[27] B. Jenkins, "Jenny download web page," Bob Jenkins' Website, 2019. [Online]. Available: http://burtleburtle.net/bob/math/jenny.html.

[28] A. R. A. Alsewari and K. Z. Zamli, "Design and implementation of a harmony-search-based variable-strengtht-way testing strategy with constraints support," *Information and Software Technology,* vol. 54, no. 6, p. 553–568, 2012.

[29] A. Hartman, "IBM Intelligent Test Case Handler," IBM alphaworks, 2019. [Online]. Available: http://www.alphaworks.ibm.com/tech/whitch.

[30] A. Calvagna and A. Gargantini, "IPO-s: incremental generation of combinatorial interaction test data based on symmetries of covering arrays," in *International Conference on Software Testing, Verification, and Validation Workshops*, Denver, CO, USA, 2009.

[31] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence, "IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing, Software Testing," *Software Testing, Verification & Reliability,* vol. 18, no. 3, pp. 125-148, 2008.

[32] R. Kuhn, "ACTS page download," 2019. [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/download_tools.html.

[33] J. Stardom, "Metaheuristics and the Search for Covering and Packing Array," *Thesis (M.Sc.), Simon Fraser University, 2001,* 2001.

[34] T. Shiba, T. Tsuchiya and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *28th Annual International Computer Software and Applications Conference*, Hong Kong, China, 2004.

[35] B. S. Ahmed, K. Z. Zamli and C. PengLim, "Application of Particle Swarm Optimization to uniform and variable strength covering array construction," *Applied Soft Computing,* vol. 12, no. 4, p. 1330–1347, 2012.

[36] B. S. Ahmed, T. Sh. Abdulsamad and M. Y. Potrus, "Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the Cuckoo Search algorithm," *Information and Software Technology,* vol. 66, p. 13–29, 2015.

[37] K. Z. Zamli, F. Din, S. Baharom and B. S. Ahmed, "Fuzzy adaptive teaching learning-based optimization strategy for the problem of generating mixed strength t-way test suites," *Engineering Applications of Artificial Intelligence,* vol. 59, pp. 35-50, 2017.

[38] K. Z. Zamli, B. Y. Alkazemi and G. Kendall, "A Tabu Search hyper-heuristic strategy for t-way test suite generation," vol. 44, pp. 57-74, 2016.

[39] H. Wu, C. Nie, F.-C. Kuo, H. Leung and C. J. Colbourn, "DPSO Page Download," 2015. [Online]. Available: https://github.com/waynedd/DPSO. [Accessed 2019].

[40] T. Mahmoud and B. S. Ahmed, "An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use," vol. 42, no. 22, pp. 8753-8765, 2015.

[41] A. B. Nasser, K. Z. Zamli, A. A. Alsewari and B. S. Ahmed, "Hybrid flower pollination algorithm strategies for t-way test suite generation," *PLOS ONE,* vol. 13, no. 5, 2018.

[42] K. Z. Zamli, F. Din, B. S. Ahmed and M. Bures, "A hybrid Q-learning sine-cosine-based strategy for addressing the combinatorial test suite minimization problem," *PLoS ONE ,* vol. 13, no. 5, 2018.

[43] B. S. Ahmed, E. Enoiu, W. Afzal and K. Z. Zamli, "An evaluation of Monte Carlo-based hyper-heuristic for interaction testing of industrial embedded software applications," *Soft Computing,* vol. 24, p. 13929–13954, 2020.

[44] A. ALFI, "PSO with Adaptive Mutation and Inertia Weight and Its Application in Parameter Estimation of Dynamic Systems," *Acta Automatica Sinica,* vol. 37, no. 5, pp. 541-549, 2011.

[45] S. Mehdizadeh, F. Fathian and J. F. Adamowski, "Hybrid artificial intelligence-time series models for monthly streamflow modeling," *Applied Soft Computing,* vol. 80, pp. 873-887, 2019.

[46] C. J. Colbourn, "Covering Array Tables for t=2, 3, 4, 5, 6," 2019. [Online]. Available: http://www.public.asu.edu/~ccolbou/src/tabby/catable.html.