ZANCO Journal of Pure and Applied Sciences
The official scientific journal of Salahaddin University-Erbil
https://zancojournals.su.edu.krd/index.php/JPAS

ISSN (print):2218-0230**, ISSN (online):** 2412-3986**, DOI: http://dx.doi.org/10.21271/zjpas**

RESEARCH PAPER

Improved Mixed Neighborhood Tabu Search by Random Selection for Combinatorial Interaction Testing

Imad H. Hasan , Moayad Y. Potrus

Department of Software and Informatics Engineering, College of Engineering , Salahaddin University-Erbil, Kurdistan Region, Iraq.

A B S T R A C T:

Combinatorial interaction testing (CIT) is a technique used to find minimal test suite among configuration options of a System Under Test (SUT) that uses a Covering Array (CA) as a combinatorial structure. CIT is very effective for reducing the costs of the testing process that uses a sampling technique instead of exhaustive testing. This paper proposes the modification of Mixed Neighborhood Tabu Search (RMiTS) algorithm using the random selection strategy. The base MiTS algorithm is originally used for generating t-way Mixed Covering Array (MCA). The modification improves the algorithm performance (running time) to cover all possible input configuration combinations to produce the optimal or near-optimal test suites. The modified algorithm is evaluated through a comparison against the base MiTS algorithm to confirm the performance improvements. Also, it is compared to a state-of-the-art algorithm known as Advanced Combinatorial Test Tool (ACTS) to confirm its efficiency. The experimental results confirm the effectiveness of the modifications that improved the performance for all applied benchmarks, and also it shows that RMiTS is more efficient than ACTS.

KEY WORDS: Combinatorial Interaction Testing, Covering Array, Combinatorial Optimization, Software Testing
DOI: http://dx.doi.org/10.21271/ZJPAS.32.5.1
ZJPAS (2020) , 32(5);1-19 .

1. INTRODUCTION

Software Testing is a very important stage in the process of Software Development Life Cycle (SDLC). A high-quality software needs an exhaustive testing process and large test suites to detect most of the bugs in the developed software. However, exhaustive testing may require higher cost budget and longer development time (Ahmed, 2016).

Instead of exhaustive testing, a sampling technique is used to minimize the size of the test suites. This means, generating a high-quality test suite is a game-changer, that reduces the testing time and development costs without affecting the quality of the software. Eventually, it can detect most of the bugs in less effort. Recently, the demand for high customizable software systems has increased. These type of systems reuses the ready components from a set that belongs to one core to produce newly developed software. Thus, the highly-configurable systems are becoming the direction and the future of software development. Significant reusability of highly-configurable systems notably reduced the development cost (Lin *et al.*, 2016). The highly-configurable systems comprise many ready components that

* **Corresponding Author:**
Imad H. Hasan
E-mail: imad.hadi@su.edu.krd
Article History:
Received: 07/09/2019
Accepted: 07/01/2020
Published: 13/10 /2020

can interact with each other. Each component has many input configurations options. Evidence revealed that software bugs occur during the small number of the interaction of these input configuration options (Kuhn *et al.*, 2013). As mentioned by Cohen *et al.* (2007), the high-configurable systems made the software testing a threat to the quality of software than before, which forces the software tester to exhaustively test the combination of all input options. Therefore, this leads to more time consuming on the testing process that increases the development costs too. Testing all the combinations of parameter values is not always feasible. Instead, a sampling technique is used to detect faults from parameter combinations. This technique called Combinatorial Interaction Testing (CIT) (Cohen *et al.*, 1997, Cohen et al., 2007). It reduces the size of the exhaustive test suite dramatically, in such a way, that cover all possible value combinations at least once.

The CIT techniques are considered as an NP-hard problem (Nie and Leung, 2011). Thus, many approaches have been proposed in the literature to solve this problem. Most two popular approaches are meta-heuristic and greedy, the approaches are either constrained or unconstrained, the systematic literature studies including Nie and Leung (2011) that summarizes an overview of CIT approaches and applications. The book by Kuhn et al. (2013) covers a practical aspect and definitions of CIT.

The above-mentioned studies categories the CIT approaches based on generation strategies into greedy algorithms, meta-heuristic algorithms and mathematical methods. The upcoming paragraph highlights the most relevant works of the first two popular categories.

First, greedy algorithms are widely used algorithms for solving CIT problems. Basically, there are two generation strategies in greedy algorithm one-parameter-at-a-time (OPTAT) and one-test-at-a-time (OTAT). An example of OPTAT is in-parameter-order IPO algorithm that starts by generating CA with only t parameters, then the algorithm extends the CA by adding one parameter at each iteration, there are two types of CA extension of horizontally and vertically consequently, this process is repeated until all t-tuples are covered. The IPO algorithm originally proposed by Lei and Tai (1998) for generating 2-way CA, but it was generalized for t-way CA generator as in-parameter-order-general (IPOG) by Lei *et al.* (2007).

A Combinatorial Test Generation Tool (ACTS) is a tool that presented by Yu *et al.* (2013) that composed of IPOG, IPOG-D, IPOG-F, and IPOG-F2 algorithms; this tool can handle large SUT models including large constraint sets with high interaction strength $t \in \{2-6\}$ that known as the state-of-the-art greedy algorithm.

Another OTAT greedy strategy that constructs CA by generating one test at each iteration until all t-tuple are covered, the purpose of using such a strategy is to cover more t-tuples at each iteration. A popular tool of this strategy is Automatic Efficient Test Generator (AETG) that firstly proposed by Cohen et al. (1997) which was the first greedy algorithm that adopted OTAT strategy and a general framework of OTAT algorithm initiated by Bryce *et al.* (2005). Also, the proposed method uses a greedy OTAT strategy in the initial solution and test case generation as described section in 3.2.1.

Second, the meta-heuristic algorithms which use random guess and evolving solutions with each cycle to reach a solution or best solution (Potrus, 2016), such as Genetic Algorithm (GA) (Ghazi and Ahmed, McCaffrey), Simulating Annealing (SA) (Cohen *et al.*, 2003b, Cohen *et al.*, 2003a, Cohen *et al.*, 2003c), Tabu Search (TS) (Gonzalez-Hernandez and Torres-Jimenez) and Particle Swarm Optimization (PSO) (Ahmed and Zamli, 2011, Ahmed *et al.*, 2012, Kalaee and Rafe, 2016, Ahmed *et al.*, 2017), that have a similar strategy for generating CA, usually starts by constructing a partial or incomplete CA then applies modifications or transformations to it, until it covers all the t-tuples. At each iteration, the algorithm moves toward an unexplored region so that it tries to cover all possible missed t-tuples as much as possible.

Cohen et al. (2003b) used the meta-heuristic Simulating Annealing (SA) algorithm to construct covering arrays for $t \in \{2-3\}$. The SA algorithm first generates an initial solution randomly as an N by k matrix. Then the temperature reduced by a constant value close to one, the cost of the evaluation function is the number of uncovered t-tuples. Within that year, Cohen et al. (2003a) integrated an algebraic construction technique to SA, this approach called Augmented Simulated Annealing.

Gonzalez-Hernandez and Torres-Jimenez (2010) used a tabu search algorithm for generating MCA and their contribution was using the mixture of three neighborhood functions called MiTS, that selects the functions based on a random probability. However this work is improved later in (Gonzalez-Hernandez, 2015), they changed a way of using MiTS to generate smaller MCAs than the best found so far, for interaction strength $t \in \{2-6\}$. The improvements included a parameter tuning based on statistical tests identified the values that significantly affect the performance of MiTS. To verify the efficiency of the proposed improvements, the results of MiTS was compared and analyzed statistically against popular approaches included the best bound MCAs for interaction strength $t \in \{2-6\}$ that have been reported to date in the literature for SA and IPOG algorithms, and others. The improved MiTS showed that there were notable differences between the obtained solutions and the best previously reported bounds.

Besides, Avila-George *et al.* (2012) used the mixed neighborhood functions with SA to construct MCA with $t \in \{2-3\}$. The MCA generation process begins with an initial solution that uses the maximum Hamming-distance among many candidate test cases, in the algorithm uses two neighborhood functions to optimize the MCA matrix.

Ahmed et al. (2012) proposed a Particle Swarm Test Generator (PSTG) to construct a covering array for $t \in \{2-6\}$. PSTG randomly generates test cases and chooses the one that has a maximum interaction coverage that utilizing the PSO with a greedy strategy for identifying the test cases. The main problem of the PSO based algorithms is related to PSO that needs many parameter tunings.

However, the improved MiTS in (Gonzalez-Hernandez, 2015) reported the best bounds (minimum possible test suite sizes). But it still suffers from low performance (long running time). Thus, in this work, we modified the MiTS using random selection strategy into random MiTS (RMiTS), the major modifications targeted the neighborhood functions N_2 and N_3 to improve the performance (reducing running time) of the neighborhood functions with random selection techniques instead of normal sequential selection. However, one of the downsides of the tabu search algorithm is suffering from local minima (Ahmed et al., 2012). To overcome this problem, one of the neighborhood functions is optimized in which the behavior of the evaluation function is modified from local best evaluation into a global best evaluation, that helped the tabu search to diversify the search.

The reminder sections of this paper are structured as follows: Section 2 starts with an example to introduce the combinatorial interaction testing and shows the mathematical notion of CA. Section 3 presents the proposed modifications of the base MiTS algorithm. Section 4 shows the design of experiments and discusses the efficiency and performance evaluation of the proposed method. The final section concludes this paper.

2. MATERIALS AND METHODS

This section presents an overview of the theoretical backgrounds on CIT and how to model the system under test. Alongside that, it explains the general concepts and mathematical notation of combinatorial data structures.

2.1. Combinatorial Interaction Testing

To illustrate the concept of CIT assume there is a System Under Test (SUT) that has a set of k input parameters or configurations such that $P = \{p_1, p_2, .., p_k\}$ and each parameter has a set of n values or options such that $Vp_i = \{v_1, .., v_n\}$ equivalents to the domain of p_i. For this purpose, let us consider Figure 1 as a SUT which is a user interface of Tesla car autopilot settings.

For example, Table 1 shows a model of the SUT, that takes eight configuration parameters, the first five parameters have two values or options, the seventh parameter has three options, and the last parameter has four options. If all the parameter options of the current SUT are exhaustively tested together it will produce $26 \times 31 \times 41 = 768$ test cases, imagine if the only parameter with four options is added to the SUT, the number of test cases will increase to $768 \times 4^1 = 3072$ the test suite size will growth exponentially, which is impossible to apply due to time and budget constraints. Though, a sampling technique such as CIT should be applied to overcome this problem.

CIT is a combination of parameter values, with a specific interaction strength usually denoted as t and the value of $t \in \{2-6\}$. CIT sampling techniques will reduce the number of test cases dramatically for example in the current SUT instead of applying all 768 test cases to cover all possible value combinations it needs only 12 test cases to

cover all possible t-tuple value combinations at least once as shown in Table 2. To understand the interaction strength *t*, let's consider given SUT that receives three parameters each with two values, such that $P = \{P_1, P_2, P_3\}$, and *X* be a function that takes *t* parameter combinations, then computes the cartesian product of their possible values that produces t-tuple sets, for example when $t = 2$ it means the $2 - way$ combination of parameters which is pairwise combination such that $X(P_1,P_2)$, $X(P_1,P_3)$, $X(P_2,P_3)$ as demonstrated in Figure 2.

Table 2 shows a test suite that equivalents to $2-way$ CIT of the SUT in Figure 1 which means the combination of parameters is pairwise, where each column represents an input parameter that contains only values from its domain, and each row represents a test case. This combination structure is called Covering Array (CA) as defined in the next subsection.

2.2. Covering Array (CA)

A Covering array is a mathematical object denoted by $CA(N;t,k,v)$ that is *N* rows by *k* columns (parameters) array. The key feature of a CA is that for every $N \times t$ sub-array, all possible value combination t-tuple sets appear at least once and they considered as covered tuples. where *N* represents the number of tests, *k* is the number of parameters, and each parameter has *v* values, *t* is the interaction strength

A Mixed Covering Array (MCA) denoted by $MCA(N;t,k,Vp_1 \ldots Vp_k)$ is an extended version of CA, the only difference with CA is that domains of the parameters in MCA are non-unified since the domains of the parameters in CA are unified. The values in the i^{th} column belong to the V_{pi} set. for example according to MCA definitions the test suite in Table 2 can be represented as $MCA(12,2,8,2^6 3^1 4^1)$.

3. THE PROPOSED METHOD

This section presents the details and implementation of the proposed method and highlights the improvements and modification to the original mixed neighborhood tabu search. In the upcoming subsections, first, the concept of tabu search algorithm and its properties are explained briefly. Then, it illustrates the steps of the test case generation. Finally, the modifications for the neighborhood functions are implemented.

Algorithm 1: A pseudo-code of a typical Tabu Search algorithm

```
1  T ← φ      // empty tabu list
2  s ∈ S      // candidate solution
3  s_best ← s // assume s is the best soliution
4  T ← s
5  while f(s_best) > 0 do
6      s' ← N(s, T)
7      if f(s') ≤ f(s_best) then
8          s_best ← s'
9      T ← s'
10     s ← s'
```

3.1. Tabu Search

The Tabu Search (TS) algorithm originally proposed by Glover (1986). The TS is a meta-heuristic local search algorithm, the principle idea of TS is the combination of memory to the search algorithm called tabu list (Glover and Laguna, 1999). The tabu list has a queue data structure that keeps a given number of the latest moves carried out to change a current solution s to a new solution s'. When a new solution is obtained, the TS algorithm avoids the moves in the tabu list, then the current move is queued to the tabu list and the very old move is dequeued from the tabu list so that the current move is banned as it lives in the tabu list. However, sometimes the TS allows the moves that live in the tabu list which can produce better solutions than the current best, this is another feature of the TS algorithm known as aspiration criteria. Other two important features of the TS algorithm are intensification and diversification for search strategies, the details can be found here (Glover, 1998). The intensification is a strategy that intensively searches a region where the best solutions found so far to find better solutions. Whereas, the diversification strategy is exploring the unvisited regions from the search space looking forward to a new solution that may vary from those solutions seen before .

The pseudo-code of a very basic TS algorithm is shown in Algorithm 1 that includes the main components of the TS including: 1) initialization s is subset of search space; 2) tabu list queue T; 3) neighborhood function $N(s,T)$ that by which TS moves from current solution s to another new solution s'; 4) the evaluation function $f(s)$; 5) the stop criteria $f(s_{best}) > 0$, in this case, is minimization function. Note that the words move and transformation are used interchangeably in the following sections.

Algorithm 2: An overview pseudo-code of the proposed test generation algorithm

```
Input: SUT(P, V_p, C), strength t, initial CMCA size N, iteration I
Output: complete CMCA M
1 S ← search_space_contruction(SUT(P,V_p,C),t);
2 M ← initialization(N);
3 while not all t-tuple in S are covored do
4     for i ← I to 0 do
5         M ← Tabu_Search(M,I);
6         if all t-tuple in S are covored then
7             stop;
8     if M is not optimized for some iterations then
9         M ← new_row_with_hamming_distance(M);
```

3.2. Test case generation algorithm

An overview of the proposed algorithm can be summarized in Figure 3, at first the algorithm receives the SUT model files, the coverage strength t and initial size of the test suite N, the SUT models files are parameters model file which refers to P and V_p.

Then, the algorithm constructs the search space S which is a two-level hash-table data structure, the first level is the parameters combinations and the second level is the combination or cross-product of their values in the form of t-tuple sets.

Next, the algorithm initializes the partial MCA M matrix of size N from Algorithm 2 see line 6 that calls the initialization function that explained in detail in section 3.2.1. Finally, the tabu

search will start to optimize the initial *M* for *I* iterations and after each tabu search call the algorithm checks for t-tuple coverage, if all t-tuple are covered then the algorithm will stop, otherwise the algorithm will add new random row to *M*, the new row is generated based on Hamming distance to diversify the search see the line 13 and section 3.3.3 for the details, this process is repeated until it covers all the t-tuples in *S* then the algorithm will stop and produces the complete MCA, see lines 7-13. It is worth to mention that the tabu search algorithm uses two neighborhood function to optimize the matrix *M*, the neighborhood functions are selected based on a random probability to explore the search space see section 3.2for the internal steps of each neighborhood function and how the algorithm optimizes the initial *M*.

3.2.1. Initialization

Initialization function takes an initial size of test suite *N* and initializes an empty partial MCA *M* matrix, As the proposed method uses one-test-at-a-time (OTAT) as test generation strategy, the initialization function generates test cases based on the two modes **random generation** and **random t-tuple selection** until the size of the initialized MCA *M* reaches *N* rows. Lastly, it passes the generated MCA *M* for optimization. the following paragraphs describe the two modes of initial generation in detail.

$$hd(r_s) = \sum_{i=1}^{s-1} \sum_{j=1}^{k} g(m_{i,j}, m_{s,j}) \quad (1)$$

$$where \quad g(m_{i,j}, m_{s,j}) = \begin{cases} 1 & m_{i,j} \neq m_{s,j} \\ 0 & otherwise \end{cases}$$

In **random generation** mode, the very first row of the matrix *M* is randomly generated in a way that each value for the parameters p_i is randomly chosen from the V_{pi} set, after that the new row is added to the matrix *M*. Then from the second row, the initialization function generates two new random rows known as candidate rows using the same way as for the first row, but the only difference here is the two candidate rows will not directly be added to the matrix *M* instead the initialization function relies on the hamming distance to choose one of the candidate rows, as demonstrated in Table 3, hamming distance can be computed using Equation 1, which is the summation of a number of different symbols between the candidate row and all current rows in the matrix *M*, so the initialization function computes a hamming-distance for the candidate rows, then a candidate row that has maximum hamming-distance is inserted to matrix *M* with the purpose of diversifying the values in the newly added row from current rows in the matrix, so that to cover as much t-tuples as

Algorithm 3: Initialization function of the proposed Algorithm

```
   Input: N
   Output: partial MCA M
1  let CTuples() = No. of covered t-tuples in S;
2  let UTuples() = No. of uncovered t-tuples in S;
3  let HD(R) = hamming distance from R to M;
4  let random_row() = creates a valid random row;
5  M ← random_row();
6  for i ← 0 to N do
       // random mode
7      if UTuples() < CTuples() then
8          C1 ← random_row();
9          C2 ← random_row();
10         if HD(C1) > HD(C2) then
11             M ← C1;
12         else
13             M ← C2;
14     else
           // selection mode
15         temp_row ← {−1} ∗ size(P);
16         let param_combinations = {set of paramter combinations};
17         while temp_row contatins -1 do
18             foreach combination ∈ param_combinations do
19                 t_tuple ← select_t_tuple(combination);
20                 temp_row[combination] ← t_tuple;
21         M ← temp_row;
```

possible to visit the unexplored regions in the search space, the initialization function repeats this process for a specific number of iterations based on some heuristics such as number of covered and uncovered t-tuples, from the Algorithm 3 see the lines 7-13 that refers to this random mode and the rest of iterations will be generated based on the second mode.

In **random t-tuple selection** mode, this mode in initial solution generation is very simple, first, it initializes a row of -1s, then the selection mode replaces the current -1s from the row with the selected uncovered t-tuples from search space *S*, each time a t-tuple is selected and it will be replaced with *t* -1s in the new row until no -1 will remain in it. This process is repeated until the size of the matrix *M* reaches to *N* rows, see the lines between 15-21.

Algorithm 4: Th second neighborhood function N_2 of base MiTS Algorithm

Input: partial M
Output: optimized M

1 $R_{best} \leftarrow +\infty$;
2 $c \leftarrow select\ random\ column\ 0\ to\ k$;
3 $r \leftarrow 0$;
4 **while** $r < size_of(M)$ **do**
5 $\quad R \leftarrow M[r]$;
6 $\quad R[c] \leftarrow randomly\ choose\ a\ value\ from\ V_{p_c} \setminus R[c]$;
7 $\quad$ **if** $F(R) \leq F(R_{best})$ **then**
8 $\quad\quad R_{best} \leftarrow f(R)$;
9 $\quad\quad$ **if** $F(R_{best}) = 0$ **then**
10 $\quad\quad\quad$ stop the algorithm;
11 $\quad r \leftarrow r + 1$;

3.2.2. The base neighborhood functions

The mixed neighborhood tabu search MiTS is first proposed by Gonzalez-Hernandez and Torres-Jimenez (2010). The original algorithm uses three neighborhood functions each one has a specific moving strategy as follows: the first one N_1 randomly changes only a value of one cell of the matrix M at each call; the second function N_2 as shown in Algorithm 4, changes the values of randomly selected column for the entire rows at each call that helps the algorithm to move toward a better solution; the third function N_3, firstly it searches for uncovered t-tuples from the search space and selects it, then it replaces the selected t-tuple with the corresponding t-tuple in the matrix M for the entire rows.

The man goal of using such neighborhood functions in tabu search algorithm is to explore the search space that at each function call, it moves from current best solution to a better solution that navigates the search space to find unexplored regions.

In general, N_2 and N_3 takes the partial MCA M as input and tries to transform the rows in M based on their strategy, after each transformation the algorithm evaluates the movement if the search goes toward better solution it is accepted otherwise is rejected. In this case, the evaluation function $F(R)$ is a minimization function. The evaluation is based on the number of covered t-tuple the tabu search continues this process until all t-tuple are covered. The most effective neighborhood functions for exploring the search space and convergence to the final solution are the second N_2 and the third one N_3, but the main problem of them is that they require to mutate the entire rows in the matrix M at each call. However, based on statistical analysis changing the all the rows in matrix M at each call will not guarantee that all the changed rows will produce a better solution see line 7 in Algorithm 4, as a result, this will lead to more time consuming and slow convergence that affects the performance of the algorithm.

To boost the convergence and reducing the running time, naturally, we asked what if randomly a specific number of rows in M are selected to transform instead of the changing entire rows sequentially as shown in Algorithm 4. The analysis of the results showed a notable boost in performance as compared with the base algorithm. The next subsection explains the modifications in detail.

3.3. The Modification

In the proposed method RMiTS the tabu search algorithm only uses the two effective neighborhood function N_2 and N_3 that are modified with random selection strategy, to explore the search space and to move from one state to another state, the tabu search selects one of the neighborhood functions based on the random probability during the search. The next subsection explains the modifications for each neighborhood

function in detail.

Algorithm 5: The improved second neighborhood function N_2 of the proposed RMiTS Algorithm

Input: partial M, number of rows I_{rows}
Output: optimized M

```
1  c ← select random column 0 to k;
2  r ← select random row 0 to N;
3  R_global_best ← the numer of uncovered t − tuples;
4  i ← 0;
5  while i < I_rows do
6      R ← M[r];
7      R[c] ← randomly choose a value from V_pc \ R[c];
8      if F(R) ≤ F(R_global_best) then
9          R_global_best ← f(R);
10         if F(R_best) = 0 then
11             stop the algorithm;
12         i ← i + 1;
13     r ← (r + 1) mod N;
```

3.3.1. Improved second neighborhood function N_2

The N_2 function in RMiTS receives the partial MCA matrix M and the number of rows I_{rows} for modifications (movements). Firstly, The N_2 initializes random row index r and random column index c then the function starts with the r^{th} row R and changes the value of c^{th} column $R[c]$ in a way that randomly chooses a value from V_{pc} except for the current value. Later, the movement R is evaluated if the fitness value is less than or equal to the best global solution, the movement R is accepted and it will be a new global best solution $R_{gloabl\ best}$ otherwise the row index r is sequentially incremented but the column index c is fixed. Finally, the process is repeated for I_{rows} iterations see the Algorithm 5, sometimes the function may converge to zero then it will stop and returns the complete MCA M see the lines (9 to 10).

It is necessary to highlight the modifications here, The first improvement is that the function to starts from a random row and modifies I_{rows} rows see the lines 2,5,6, and 13, instead of starting from the first row and modifying all the rows as shown in Algorithm 4, using this random selection strategy boosted the performance of this function.

The second contribution is that we modified the locality property of the tabu search that allows the search return to the previous or explored regions which means it accepts the moves that worse than R_{best} found from previous function calls see lines 1 and 7 in Algorithm 4, but we modified the N_2 that does not allow the search to return to the worse moves which means a global search and the evaluation function is based globally found $R_{global\text{-}best}$ the total number of uncovered t-tuples among every function calls see the modification at lines 3,8 and 9 in Algorithm 5.

Algorithm 6: The improved third neighborhood function N_3 of the proposed RMiTS Algorithm

Input: partial M, number of rows I_{rows}
Output: optimized M

```
1  T ← {empty tabu list};
2  r ← 0;
3  R_best ← +∞;
4  i ← 0;
5  while i < I_rows do
6      r ← select random row 0 to N \ {T};
7      R ← M[r];
8      R[(t − tuple)] ← randomly choose uncovered t − tuple from S;
9      if F(R) ≤ F(R_best) then
10         R_best ← f(R);
11         if F(R_best) = 0 then
12             stop the algorithm;
13         i ← i + 1;
14     T ← T ∪ {r};
```

3.3.2. Improved third neighborhood function N_3

The third neighborhood function N_2 in RMiTS takes the partial MCA matrix M and the number of rows I_{rows} for movements, Algorithm 6 shows the steps of this function, first N_2 selects a random index or row r except for the indices currently in the tabu list, then the function N_2 starts with the r^{th} row and replaces the randomly selected uncovered t-tuple from the search space S with the corresponded t-tuple indices from R see the line 7, later the new move R is evaluated if the fitness value is less than or equal to the local best solution the move R is accepted and it will be a new local best solution R_{best} and the current row index r is added to the tabu list and it will be a tabu for this function call, for the next iteration a new random row is selected and the process is repeated for I_{rows} iterations, the function N_2 may converge to zero then it will stop and returns the complete MCA M.

The main modification done on this function is based the random selection (random sampling) strategy, the function N_3 randomly select I_{row} rows in matrix M for modifications see the lines 5, 6, 7 and 14 in Algorithm 6, instead of modifying all the rows sequentially. This improvement is very effective to diversify the search process, as a result, the tabu search converged faster than the base algorithm.

4. EVALUATION OF RESULTS AND DISCUSSION

This section is first, describes the benchmarks used to evaluate the efficiency of the proposed method. Second, specifies the tools and approaches that the proposed method is compared against. Then it shows the settings of the experiments. Next, it presents the evaluation of the experiment results. Finally, it presents the effectiveness of the generated test suites through an imperial case study.

4.1. Benchmarks

In the experiments of evaluating the proposed method, we use a specific number of real-world system benchmarks some of them are the exact benchmarks used by Gonzalez-Hernandez (2015) for evaluating the MiTS that found the best bounds so far for unconstrained CIT problem. The most popular benchmarks that are the model of real-world systems including:

1. **Bugzilla** a bug-tracking system.
2. **SPINV** a verifier for SPIN model checker.
3. **SPINS** a simulator for SPIN.
4. **GNU Gzip** which is a popular data compression program in the GNU project.
5. **Android** is an open-source mobile phone operating system.
6. **RFID** is a radio-frequency identification uses electromagnetic fields to automatically identify and track tags attached to objects.
7. **Tesla,** we modelled this benchmark which is taken from autopilot settings of Tesla electric car.
8. **TCAS** a traffic collision avoidance system which is an aircraft collision avoidance system designed to reduce the incidence of mid-air collisions between aircraft.

4.2. Experiment Settings

To perform a fair evaluation, the modified MiTS algorithm and the original MiTS algorithm are both implemented in C# language and compiled with .NET Core version 2.0. We conducted exhaustive experiments for t-way testing, for most of the benchmarks in the experiment we set different values for the control parameters of the tabu search including initial test suite (MCA) size N and the number of rows I_{rows} for the neighborhood functions.

To evaluate the proposed method RMiTS we conduct two set of experiments, in the first set experiment the RMiTS is compared the original MiTS to confirm the efficiency (test suite size) and performance (running time) improvements based on the proposed modifications, the second set experiments the RMiTS is compared to a state-of-the-art t-way CIT generator called Advanced Combinatorial Test Tool (ACTS) proposed by Yu et al. (2013) that implemented in Java language to prove the efficiency (test suite size) of the RMiTS.

All the experiments are executed 20 times for both RMiTS and MiTS but ACTS only executed once as it is a deterministic algorithm that produces the same result in every run. The experiments are executed under macOS environment on a machine with 2.6 GHz Intel Core i7 process and with 16 GB memory.

It is important to note that, for all the experiments, the iteration I of the tabu search algorithm is fixed to 100 for both MiTS and RMiTS. However, the best bounds that found by MiTS as reported in (Gonzalez-Hernandez, 2015), the iteration of the tabu search was set to 1000, as iteration increases the algorithm takes longer running and smaller results. Some of the results of the original MiTS that reported in this paper may not compatible with those reported in (Gonzalez-Hernandez, 2015) because due to lack of similar running environment as the original MiTS was executed in a hybrid cluster with 256 processing nodes with 1056 CPU cores, 2112 GB of memory RAM, but the experiments in this work as mentions above are executed in a personal computer, which this shows another strength of work as achieved many similar results with much more lower cost.

4.3. Results and discussion

This section presents the results of both sets of experiments including the performance and efficiency assessments and discusses the outcomes to evaluate the modifications done in the new algorithm. Table 4 shows the results of the first set of experiments which is an efficiency and performance comparisons between original MiTS and the improved RMiTS, the first column in the table indicates the identification (ID) of the benchmark average execution with a specific strength for both tools MiTS and RMiTS. The second column represents the SUT model and configurations of the real-word benchmarks, the third column is k which is a number of input parameters for each corresponding configuration. The values in column 5 are the average of test suite sizes N of 20 runs of the original MiTS algorithm and column 6 contains the average values of the running times (in seconds) required by the MiTS algorithm to

generate the test suites sizes on column 5. Similarly, the columns 7 and 8 are the average of the test suite size *N* of 20 runs and running time (in seconds) values of the improved algorithm RMiTS. The last two columns represent differences in the size and time respectively between MiTS and RMiTS. In general, the values in the table that are in bold indicate the improvements either in size or in time, the values with a symbol (*) indicate the theoretical best bound and can't be optimized further which is the optimal test suite size.

As mentioned before, this study aimed to improve the performance (reducing running time) of the original MiTS. So, according to the results reported in Table 4, the modified algorithm RMiTS outperforms MiTS in performance for all the benchmarks with different interaction strength *t* without exception as all the values in column 8 in bold, however when the strength $t = 2$ for the benchmarks Mobile Phone, SPINS, Android, RFID, gzip and Tesla the differences of the average running times between MiTS and RMiTS were very small amount of time because for such interaction strength the search space size is very small if compared with the search spaces for higher strengths, thus both algorithms were very fast in generating test suites. As interaction strength goes higher the difference amount of time goes higher too as shown in the last column, especially when interaction strength $t \geq 4$ the percentage of performance improvements between 50% to 75% and the average performance improvements for all the interaction strengths is 52%.

Usually improving the performance of a combinatorial problems costs the efficiency degradation, but the test suites generated with RMiTS as shown in the Table 4 confirm that the efficiency of RMiTS is enhanced for 9 benchmark instances out of 34 as compared with the original MiTS with better performance see the benchmark instances 9, 15, 16, 20, 25, 27, 29 and 34, the positive values in column 9 indicates the difference of test suites sizes between MiTS and RMiTS for the benchmark instances for which the efficiency was enhanced and the enhanced test suite sizes are in bold text in column 7. Remember, RMiTS matched MiTS for 12 benchmark instances that generated the same results but with higher performance especially for benchmark instances 23,24 and 28. Also, both MiTS and RMiTS generated the theoretical best bound test suite sizes for 10 instances that marked with (*) symbol. For the remaining tests, the RMiTS showed poor efficiency as compared with MiTS for 13 benchmark instances but the differences in average test suite sizes between the MiTS and RMiTS is very small, as the differences of 8 benchmark instances out of the 13 benchmark instances are less than one see the column 9 the negative values indicate the poor results, however, the differences are relatively small but don't forget that the RMiTS results have almost higher performance.

The second set of experiments which is a comparison of efficiency only between the improved algorithm RMiTS and a state-of-the-art greedy algorithm ACTS to evaluate the improvements in the efficiency of the modified algorithm. Table 4 shows the comparison of the average test suite sizes generated by RMiTS with test suite sizes generated by the greedy algorithm ACTS for the same benchmarks used in the first set of experiments, the first column in the table represents the benchmark instance identification, the second column is SUT model, the column 3 and 4 indicate the number of inputs parameters and interaction strength respectively, column 5 contains the size *N* of test suite generated by one run of ACTS tool, column 6 represents the average of test suite sizes *N* generated by 20 runs of the improved algorithm RMiTS, and the last column is the differences in size between the two competitors.

The observed results in Table 4 show that the improved RMiTS algorithm generated smaller test suites than ACTS for 23 benchmark instances out of 34 as in bold text in column 6, surprisingly the differences of test sizes between ACTS and RMiTS are incredibly high for interaction strengths $t \geq 4$ see the benchmarks instances 14, 15, 16, 19, 20, 25, 28, 29, 32, 33 and 34 in which the

RMiTS generated significantly smaller test suites. ACST generated smaller test suites only for benchmark instances 8 and 22, but the difference in the size of benchmark instance 22 is very small it is less than 0.5, thus only the benchmark instance 8 can be considerable. The RMiTS and ACTS generated similar results for 9 benchmark instances see the last column the value 0 indicates the similarity of test suite sizes generated by both tools. It is important to be noted, 8 instances of the 9 similar results generated by both tools are the theoretically best bounds that can't be optimized further, thus if we ignore these 8 similar instances, we can say that the RMiTS efficient than ACTS for the 88% of applied benchmark instances.

7. CONCLUSION

In this paper, we present an improvement for mixed neighborhood functions tabu search which is a meta-heuristic search algorithm, for generating t-way mixed covering array (MCA), the main contributions of this research are using random selection strategy for the two neighborhood functions N_2 and N_3 to explore the search space more thoroughly and rapidly.

For evaluating the proposed algorithm, in the experiments, we used different real-world systems with t-way testing to measure the quality of generated test cases. The results showed that RMiTS improved in performance for all the interaction strengths for about 52% as compared with original MiTS. Where running time is reduced without significant effects in the efficiency, also the efficiency of RMiTS still competitive and the test suites generated by RMiTS are significantly higher-quality than one state-of-the-art tool such as ACTS for about 88% of the applied benchmark instances. The result of both sets of experiments showed that the improvements in performance and efficiency are mainly targeted the higher interaction strength $t \geq 4$. As a result, we can say that the aim of this work is met.

As technology growth alongside many real-world system growths too that includes constraints among input configuration, therefore CIT is shifting toward constraints. The main future challenges for this research are investigating the search space and time optimizations with the presence of constraints. It is worth to mention that our approach is unable to generate the MCA for benchmarks that have more than 100 parameters for interaction strengths $t \geq 4$ due to the full memory footprint, many strategies that can be used to reduce the memory occupation of the MCA data structure, such as compression techniques or using different strategy for storing the t-tuples sets. However, this might affect the running time but there is a possibility to optimize the running time by implementing our approach in a parallel computing environment such as Compute Unified Device Architecture (CUDA) (Ploskas *et al.*, 2016) which is C/C++ language based framework runs on NVIDA Graphics Processing Unit (GPU), or using GPU programming with Matlab (Schmidt *et al.*, 2018).

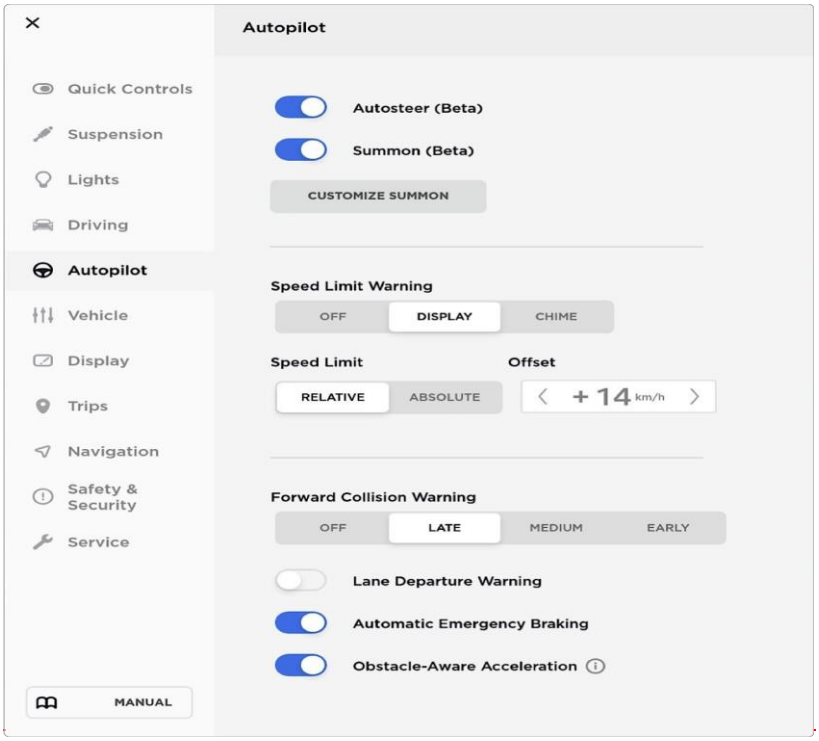

Figure 1: GUI of Tesla electric car autopilot settings

Table 1: Input parameters and their values of Tesla electric car autopilot settings

#	**Parameter**	**Values**
1	Autosteer	off, on
2	Summon	off, on
3	Lane Departure Warning (LDW)	off, on
4	Automatic Emergency Breaking (AEB)	off, on
5	Obstacle-Aware Acceleration (OAA)	off, on
6	Speed Limit (SL)	relative, absolute
7	Speed Limit Warning (SLW)	off, display, chime
8	Forward Collision Warning (FCW)	off, late, medium, early

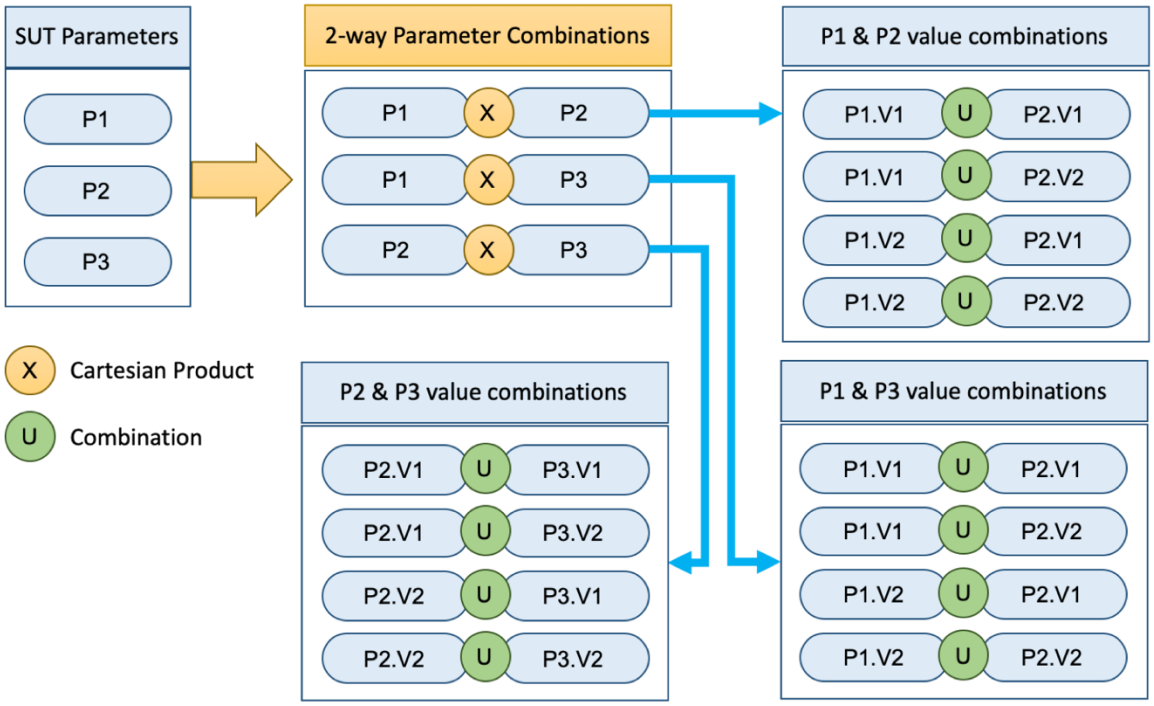

Figure 2: Modeling a 2-way CIT

Table 2: 2-way test suite of Tesla electric car autopilot settings

#	Autosteer	Summon	LDW	AEB	OAA	SL	SLW	FCW
1	on	on	off	off	on	absolute	chime	Medium
2	off	off	on	on	on	absolute	display	Late
3	on	on	on	on	on	absolute	off	Off
4	on	on	off	off	off	relative	chime	Late
5	off	off	on	on	on	absolute	chime	Early
6	off	on	off	on	off	relative	display	Early
7	off	off	on	off	off	relative	off	Medium
8	off	off	off	on	off	relative	chime	Off
9	off	off	on	on	off	relative	display	Medium
10	on	off	off	off	off	absolute	off	Early
11	on	on	on	off	on	relative	display	Off
12	on	on	off	off	off	relative	off	Late

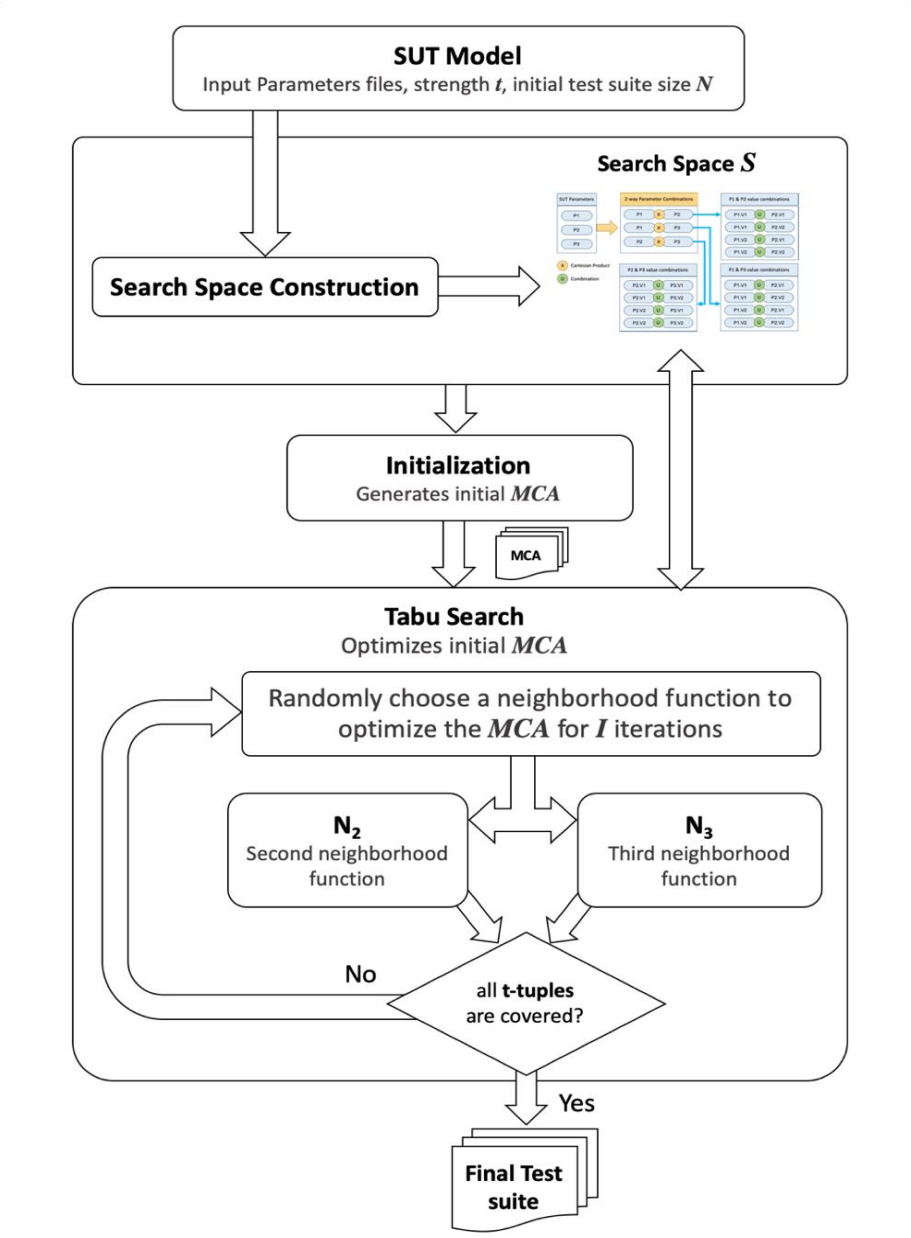

Figure 3: An overview block diagram of the proposed method

Table 3: Computing Hamming Distance for candidate rows

Test	P1	P2	P3	
R1	0	1	0	
C1	1	1	0	$hd(C1) = 1$
C2	0	0	1	$hd(C1) = 2$

R2	0	0	1	$C2$ is selected as a next row

Table 4: Efficiency(size) and performance(time in seconds) comparison between original MiTS and modified RMiTS, ***k*** refres to a number of paarameters in the benchmarks and ***t*** is an interaciton strength.

#	Benchmarks	k	t	Base MiTS		Improved RMiTS		Δ size	Δ time(s)
				Avg. size	Avg. time(s)	Avg. size	Avg. time(s)		
1	Mobile Phone	5	2	9*	0.04	9*	**0.03**	0	0.01
2	$3^3 2^2$		3	27*	0.40	27*	**0.22**	0	0.18
3			4	54*	0.34	54*	**0.17**	0	0.17
4			5	108*	1.20	108*	**0.41**	0	0.79
5	Bugzilla	52	2	17.50	4.86	**16.75**	**1.43**	0.75	3.43
6	$4^2 3^1 2^{49}$		3	59.80	2415.98	60.40	**1092.64**	-0.60	1323.34
7	SPIN simulator	18	2	18.60	0.89	19	**0.53**	-0.40	0.36
8	$4^5 2^{13}$		3	87.25	150.10	88.60	**62.57**	-1.35	87.53
9			4	345	14712.20	**337**	**4797.25**	8	9914.95
10	SPIN verifier	55	2	28	19.17	28.25	**12.05**	-0.25	7.12
11	$4^1 13^2 2^4 2$		3	151	4820.82	154.50	**2967.75**	-3.50	1853.07
12	Android	9	2	25.40	0.45	25.60	**0.25**	-0.20	0.20
13	$5^2 4^4 3^3$		3	131.20	54.48	132	**22.41**	-0.80	32.07
14			4	593.50	910.12	596.40	**472.50**	-2.90	437.62
15			5	2449.50	16087.85	**2426**	**7813.74**	23.50	8274.12
16			6	8308	144976.62	**8236**	**46652.35**	72	98324.27
17	RFID	11	2	33.60	1.14	34.80	**0.68**	-1.20	0.46
18	$5^7 2^4$		3	212	103.49	216	**62.29**	-4	41.20
19			4	1124.67	5649.59	1132.50	**4080.29**	-7.83	1569.30
20			5	5332	239766.95	**5316**	**119330**	16	120436.95
21	gzip	8	2	90*	3.48	90*	**2.67**	0	0.80
22	$10^1 9^1 2^6$		3	180*	21.87	180.20	**8.78**	-0.20	13.09
23			4	540	378.72	540	**209.86**	0	168.86
24			5	1080	1164.93	1080	**418.59**	0	746.34
25			6	1924.50	1601.92	**1911.67**	**400.85**	12.83	1201.07
26	TCAS	12	2	100*	6.36	100*	**1.48**	0	4.88
27	$10^1 4^1 3^2 2^7$		3	401.8	150.33	**400***	**32.81**	1.80	117.52
28			4	1206	7744.95	1206	**3221.52**	0	4523.43
29			5	3800	216248.35	**3754**	**58242.54**	46	158005.81
30	Tesla	8	2	12*	0.13	12*	**0.09**	0	0.04
31	$4^1 3^1 2^6$		3	28*	1.50	28*	**1.18**	0	0.32
32			4	72*	21.50	72*	**11.71**	0	9.79
33			5	144	30.47	144.50	**11.23**	-0.50	19.24
34			6	257.25	31.93	**257.20**	**9.43**	0.05	22.51

Table 5: Efficiency(size) comparison between RMiTS and state-of-the-art tool ACTS, k refers to a number of parameters in the benchmarks and t is an interaction strength

#	Benchmarks	k	t	ACTS	RMiTS	Δ size
				size	Avg. size	
1	Mobile Phone	5	2	9*	9*	0
2	$3^3 2^2$		3	27*	27*	0
3			4	54*	54*	0
4			5	108*	108*	0
5	Bugzilla	52	2	18	**16.75**	1.25
6	$4^2 3^1 2^{49}$		3	67	**60.40**	6.60
7	SPIN simulator	18	2	24	**19**	5
8	$4^5 2^{13}$		3	**79**	88.60	-9.60
9			4	343	**337**	6
10	SPIN verifier	55	2	33	**28.25**	4.75
11	$4^1 13^2 2^4 2$		3	159	**154.50**	4.50
12	Android	9	2	29	**25.60**	3.40
13	$5^2 4^4 3^3$		3	139	**132**	7
14			4	631	**596.40**	34.60
15			5	2582	**2426**	156
16			6	9107	**8236**	871
17	RFID	11	2	45	**34.80**	10.20
18	$5^7 2^4$		3	239	**216**	23
19			4	1292	**1132.50**	159.50
20			5	6030	**5316**	714
21	gzip	8	2	90*	90*	0
22	$10^1 9^1 2^6$		3	**180***	180.20	-0.20
23			4	634	**540**	94
24			5	1080	1080	0
25			6	2520	**1911.67**	608.33
26	TCAS	12	2	100*	100*	0
27	$10^1 4^1 3^2 2^7$		3	400*	400*	0
28			4	1359	**1206**	153
29			5	4233	**3754**	479
30	Tesla	8	2	12*	12*	0
31	$4^1 3^1 2^6$		3	32	**28***	4
32			4	92	**72***	20
33			5	172	**144.50**	27.50
34			6	336	**257.20**	78.80

References

AHMED, B. S. 2016. Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. *Engineering Science and Technology, an International Journal*, 19**,** 737-753.

AHMED, B. S., GAMBARDELLA, L. M., AFZAL, W. & ZAMLI, K. Z. 2017. Handling constraints in combinatorial interaction testing in the presence of multi objective particle swarm and multithreading. *Information and Software Technology*, 86**,** 20-36.

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**,** 2171-2185.

AHMED, B. S., ZAMLI, K. Z. & LIM, C. P. 2012. Constructing a t-way interaction test suite using the particle swarm optimization approach. *International Journal of Innovative Computing, Information and Control*, 8**,** 431-452.

AVILA-GEORGE, H., TORRES-JIMENEZ, J., HERNÁNDEZ, V. & GONZALEZ-HERNANDEZ, L. 2012. Simulated Annealing for Constructing Mixed Covering Arrays. Springer, Berlin, Heidelberg.

BRYCE, R. C., COLBOURN, C. J. & COHEN, M. B. A framework of greedy methods for constructing interaction test suites. 2005 New York, New York, USA. ACM Press, 146-146.

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**,** 437-444.

COHEN, M. B., COLBOURN, C. J. & LING, A. C. H. Augmenting simulated annealing to build interaction test suites. 2003 2003a. IEEE, 394-405.

COHEN, M. B., DWYER, M. B. & SHI, J. Interaction testing of highly-configurable systems in the presence of constraints. 2007 2007 New York, New York, USA. ACM Press, 129-129.

COHEN, M. B., GIBBONS, P. B., MUGRIDGE, W. B. & COLBOURN, C. J. Constructing test suites for interaction testing. 2003 2003b. IEEE, 38-48.

COHEN, M. B., GIBBONS, P. B., MUGRIDGE, W. B., COLBOURN, C. J. & COLLOFELLO, J. S. A variable strength interaction testing of components. 2003 2003c. IEEE Comput. Soc, 413-418.

GHAZI, S. A. & AHMED, M. A. Pair-wise test coverage using genetic algorithms. 2003. IEEE, 1420-1424.

GLOVER, F. 1986. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13**,** 533-549.

GLOVER, F. 1998. Tabu Search: A Tutorial. *Interfaces*, 20**,** 74-94.

GLOVER, F. & LAGUNA, M. 1999. *TABU search*, Kluwer Academic Publishers.

GONZALEZ-HERNANDEZ, L. 2015. New bounds for mixed covering arrays in t-way testing with uniform strength. *Information and Software Technology*, 59**,** 17-32.

GONZALEZ-HERNANDEZ, L. & TORRES-JIMENEZ, J. 2010. MiTS: A new approach of tabu search for constructing mixed covering arrays. 2010. Springer, Berlin, Heidelberg, 382-393.

KALAEE, A. & RAFE, V. 2016. An Optimal Solution for Test Case Generation Using ROBDD Graph and PSO Algorithm. *Quality and Reliability Engineering International*, 32**,** 2263-2279.

KUHN, D. R., KACKER, R. N. & LEI, Y. 2013. *Introduction to combinatorial testing*, CRC Press.

LEI, Y., KACKER, R., KUHN, D. R., OKUN, V. & LAWRENCE, J. 2007. IPOG: A general strategy for T-way software testing. 2007/03//. IEEE, 549-556.

LEI, Y. & TAI, K. C. 1998. In-parameter-order: A test generation strategy for pairwise testing. 1998/11//. IEEE Comput. Soc, 254-261.

LIN, J., LUO, C., CAI, S., SU, K., HAO, D. & ZHANG, L. TCA: An efficient two-mode meta-heuristic algorithm for combinatorial test generation. 2016/11// 2016. IEEE, 494-505.

MCCAFFREY, J. D. Generation of Pairwise Test Sets Using a Genetic Algorithm. 2009. IEEE, 626-631.

NIE, C. & LEUNG, H. 2011. A survey of combinatorial testing. *ACM Computing Surveys*, 43**,** 1-29.

PLOSKAS, N., SAMARAS, N., PLOSKAS, N. & SAMARAS, N. 2016. Introduction to GPU programming in MATLAB. *GPU Programming in MATLAB***,** 71-107.

POTRUS, Y. M. 2016. Maintenance Scheduling Optimization for Electrical Grid Using Binary Gray Wolf Optimization Technique. *ZANCO Journal of Pure and Applied Sciences*, 28.

SCHMIDT, B., GONZÁLEZ-DOMÍNGUEZ, J., HUNDT, C., SCHLARB, M., SCHMIDT, B., GONZÁLEZ-DOMÍNGUEZ, J., HUNDT, C. & SCHLARB, M. 2018. Compute Unified Device Architecture. *Parallel Programming***,** 225-285.

YU, L., LEI, Y., KACKER, R. N. & KUHN, D. R. ACTS: A combinatorial test generation tool. 2013/03//. IEEE, 370-375.