A Tabu Search Hyper-Heuristic Strategy for *t-way* Test Suite Generation

KAMAL Z. ZAMLI
IBM Centre of Excellence
Faculty of Computer Systems and Software Engineering
Universiti Malaysia Pahang
Lebuhraya Tun Razak, 26300 Kuantan, Pahang Darul Makmur, Malaysia
Email: kamalz@ump.edu.my

BASEM Y. ALKAZEMI
College of Computer and Information Systems
Umm Al-Qura University, Kingdom of Saudi Arabia
Email: bykazemi@uqu.edu.sa

GRAHAM KENDALL
School of Computer Science
University of Nottingham Malaysia Campus
Jalan Broga, 43500 Semenyih, Selangor Darul Ehsan, Malaysia
Email: Graham.Kendall@nottingham.edu.my

Abstract

This paper proposes a novel hybrid *t*-way test generation strategy (where *t* indicates interaction strength), called High Level Hyper-Heuristic (HHH). HHH adopts Tabu Search as its high level meta-heuristic and leverages on the strength of four low level meta-heuristics, comprising of Teaching Learning Based Optimization, Global Neighborhood Algorithm, Particle Swarm Optimization, and Cuckoo Search Algorithm. HHH is able to capitalize on the strengths and limit the deficiencies of each individual algorithm in a collective and synergistic manner. Unlike existing hyper-heuristics, HHH relies on three defined operators, based on improvement, intensification and diversification, to adaptively select the most suitable meta-heuristic at any particular time. Our results are promising as HHH manages to outperform existing *t-way* strategies on many of the benchmarks.

***Keywords*:** Software Testing; *t-way* Testing; Hyper-Heuristic; Particle Swarm Optimization, Cuckoo Search Algorithm, Teaching Learning based Optimization, Global Neighborhood Algorithm

---

1. Introduction

Interaction (*t-way*) testing is a methodology to generate a test suite for detecting interaction faults. The generation of a *t-way* test suite is a n *NP* hard problem [1]. Many *t-way* strategies have been presented in the scientific literature. Some early algebraic *t-way* strategies exploit exact mathematical properties of orthogonal arrays. These *t-way* strategies are often fast and produce optimal solutions, yet they impose restrictions on the supported configurations and interaction strength. Computational *t-way* strategies remove such restrictions, allowing for the support of arbitrary configurations at the expense of producing (potentially) non-optimal solution.

By formulating interaction testing as an optimization problem, recent efforts have focused on the adoption of meta-heuristic algorithms as the basis for *t-way* strategies. Search Based Software Engineering (SBSE) [2-4], is a relatively new field that has proposed meta-heuristic based *t-way* strategies (e.g. Genetic Algorithms (GA) [5], Particle Swarm Optimization (PSO) [6, 7], Harmony Search Algorithm (HS) [8], Ant Colony Algorithm (ACO) [5], Simulated Annealing (SA) [9, 10] and Cuckoo Search (CS) [11]). The adoption of these meta-heuristic based strategies appears to be effective for obtaining good quality solutions, as reported in benchmarking experiments related to *t-way* testing [7, 8]. Nevertheless, as suggested by the *No Free Lunch theorem* [12], no single meta-heuristic can outperform all others even over different instances of the same problem. For this reason, hybridization of meta-heuristics can be the key to further enhance the performance of *t-way* strategies. Since each meta-heuristic has its own advantages, meta-heuristic hybridization is beneficial for compensating the limitation of one with the strengths of another. In fact, the best results of many optimization problems are often obtained by hybridization [13].

In this paper we explore the hybridization of meta-heuristics based on a hyper-heuristic approach. We present a new *t-way* testing strategy. Specifically, our contributions can be summarized as follows:

- A novel hyper-heuristic based strategy, which we have termed High Level Hyper-Heuristic (HHH), for general combinatorial *t-way* test suite generation. HHH employs Tabu Search (TS) as its high level meta-heuristic (HLH) and leverages on the strength of four low level meta-heuristics (LLH), comprising Teaching Learning based Optimization (TLBO) [14], Global Neighborhood Algorithm (GNA) [15], Particle Swarm Optimization (PSO) [16], and Cuckoo Search Algorithm (CS) [17]. To the best of our knowledge, HHH is the first hyper-heuristic based strategy that addresses the problem of *t-way* test suite generation.

- A new hyper-heuristic approach for the meta-heuristic selection and acceptance mechanism based on three operators (i.e. improvement, diversification and intensification) that are integrated into the tabu search HLH. As the name suggests, the improvement operator checks for improvements in the objective function. The diversification operator measures how diverse the current and the previously generated solution are against the population of potential candidate solutions. Finally, the intensification operator evaluates how close the current and the previously generated solution are against the population of solutions.

The rest of the paper is organized as follows. Section 2 presents an overview of hyper-heuristics and the mathematical and theoretical foundation for *t-way* testing. Section 3 reviews the state-of-the-art for *t-way* test case generation strategies. Section 4 presents the design and implementation of HHH. Section 5 describes the calibration of HHH. Section 6 evaluates HHH against existing strategies and section 7 debates the usefulness of HHH. Section 8 elaborates on threats to validity. Finally, section 9 presents our conclusion.

2. Theoretical Framework

2.1. *Overview of Hyper-Heuristics*

To put hyper-heuristics into perspective, consider the different possible options for utilizing and combining meta-heuristic algorithms (see Figure 1). The first option, shown for completeness, is a standard meta-heuristic algorithm and can be ignored from our discussion as we want to focus on hybridization methodologies. These are shown in remaining figures, that is a hybrid meta-heuristic and a hyper-heuristic.

Hybrid meta-heuristics can be low-level or high level hybridizations [13]. Low-level hybridization combines two or more algorithms. High-level hybridization retains the original meta-heuristic, which can run independently (e.g. either in sequence or parallel) without any connection amongst the meta-heuristics involved (i.e. it operates as a black box).

Hyper-heuristics could be seen as a hybrid meta-heuristic owing to the integration of more than one meta-heuristic algorithm (refer to Figure 1). However, unlike a typical hybrid meta-heuristic, hyper-heuristics (or *(meta)-heuristic to choose (meta)-heuristics* [18-20]) adopts a high level meta-heuristic (HLH) to adaptively select from a set of low level meta-heuristics (LLHs), which are applied to the problem at hand. The LLHs communicate with the HLH through a domain barrier to relay the feedback of the quality of the current solution. Only the LLHs have domain knowledge, meaning that the HLH is a general algorithm that can be utilized for different problems without any algorithmic changes. It is only required to supply a different set of LLHs. In fact, the LLHs can also be formed from (low-level or high-level) hybrid meta-heuristics themselves.

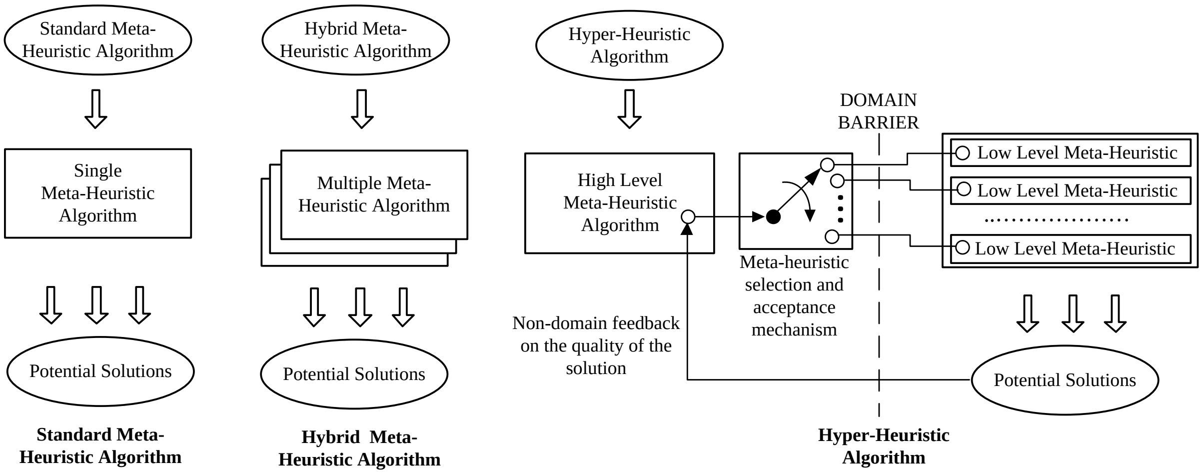
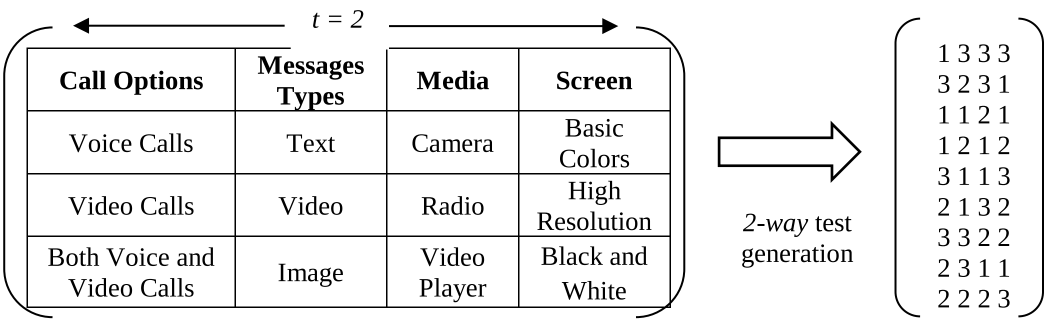


Figure 1. Standard Meta-Heuristic, Hybrid Meta-Heuristic and Hyper-Heuristic

Recent developments have introduced hyper-heuristics that are able to automatically generate the LLHs, whereby the end user does not have to implement a set of LLHs for each problem domain. Moreover, the HLH is also able to evolve its own selection and acceptance criteria [21, 22].

2.2. *The t-way Test Generation Problem*

Consider a hypothetical example of a Mobile Phone Product Configuration. The product configuration has four features (or parameters): Call Options, Message Types, Media, and Screen. Each parameter takes three possible values (e.g. Call Options = {Voice Calls, Video Calls, Both Voice and Video Calls}, Message Types = {Text, Video, Image}, Media = {Camera, Radio, Video Player}, and Screen = {Basic Colors, High Resolution, Black and White}). The pairwise (*2-way*) test generation for the Mobile Product Configuration can be seen in Figure 2 with nine test cases. The mapping of the corresponding tests can be achieved (row-wise) from the *2-way* representation based on the defined parameter values (column-wise) as depicted in Table 1. It should be noted that all the *2-way* interaction tuples between parameters are covered at-least once.



Call Options	**Messages Types**	**Media**	**Screen**
Voice Calls	Text	Camera	Basic Colors
Video Calls	Video	Radio	High Resolution
Both Voice and Video Calls	Image	Video Player	Black and White

Figure 2. Pairwise Test Suite for Mobile Phone Product Configuration

Table 1. Mapping of the Pairwise Test Suite for Mobile Phone Product Configuration

Test ID	**Call Options**	**Message Types**	**Media**	**Screen**
1	Voice Calls	Image	Video Player	Black and White
2	Both Voice and Video Calls	Video	Video Player	Basic Colors
3	Voice Calls	Text	Radio	Basic Colors
4	Voice Calls	Video	Camera	High Resolution
5	Both Voice and Video Calls	Text	Camera	Black and White
6	Video Calls	Text	Video Player	High Resolution
7	Both Voice and Video Calls	Image	Radio	High Resolution
8	Video Calls	Image	Camera	Basic Colors
9	Video Calls	Video	Radio	Black and White

Mathematically, the *t-way* test generation problem can be expressed by Equation 1.

$$f(Z) = |\{I \ in \ VIL: Z \ covers \ I\}| \qquad (1)$$

$$Subject \ to \ Z = Z_1, Z_2, \ldots Z_i \ in \ P_1, P_2, \ldots \ldots P_i; \ i = 1, 2, \ldots N$$

where, *f*(*Z*) is an objective functions (or the fitness evaluation), *Z* (i.e., the test case candidate) is the set of decision variables Z_i, *VIL* is the set of non-covered interaction tuples (*I*), the vertical bars | · | represent the cardinality of the set and the objective value is the number of non-covered interaction tuples covered by *Z*, P_i is the set of possible range of values for each decision variable, that is, P_i = discrete decision variables ($Z_i(1)<Z_i(2)<\ldots\ldots<Z_i(K)$); N is the number of decision variables (i.e. parameters); and K is the number of possible values for the discrete variables.

2.3. *The Covering Array Notation*

In general, *t-way* testing has strong associations with the mathematical concept of Covering Arrays (CA). For this reason, *t-way* testing often adopts CA notation for representing *t-way* tests [23]. The notation CA_λ ($N;t,k,v$) represents an array of size N with v values, such that every $N\times t$ sub-array contains all ordered subsets from the v values of size t at least λ times [24, 25], and k is the number of components. To cover all *t*-interactions of the components, it is normally sufficient for each component to occur once in the CA. Therefore, with λ=1, the notation becomes CA ($N;t,k,v$). When the CA contains a minimum number of rows (N), it can be considered an optimal CA according to the definition in Equation 2 [26].

$$\text{CAN}\ (t,k,v)= \min\{\ \text{N}: \exists\ CA_\lambda\ (N;t,k,v)\} \qquad (2)$$

To improve readability, it is customary to represent the covering array as CA ($N;t,k,v$) or simply CA($N;t,v^k$). Using our earlier example of the mobile phone product configuration in Figure 2, the test suite can be represented as CA (9; 2, 3^4). In the case when the number of component values varies, this can be handled by Mixed Covering Array (MCA) ($N;t,k,(v_1,v_2,\ldots v_k)$) [27]. Similar to covering array, the notation can also be represented by MCA ($N;t,k,v^k$). For example, MCA (9; 2, 3^2 2^2) represents a test suite of size nine for a system with four components (two components having three values and two components having two values) covering two-way interactions. Figure 3 illustrates the two aforementioned CA and MCA arrangements respectively.

CA (9; 2, 3^4)			
k_1	**k_2**	**k_3**	**k_4**
1	3	3	3
3	2	3	1
1	1	2	1
1	2	1	2
3	1	1	3
2	1	3	2
3	3	2	2
2	3	1	1
2	2	2	3

MCA (9; 2, 3^2 2^2)			
k_1	**k_2**	**k_3**	**k_4**
2	1	1	2
2	2	2	1
3	3	2	2
1	3	1	1
1	1	2	1
1	2	1	2
3	2	1	1
3	1	1	1
2	3	1	2

Figure 3. Illustration of CA (9; 2, 3^4) and MCA (9; 2, 3^2 2^2)

Having described the theoretical framework, the following section surveys the existing studies on *t-way* strategies in order to reflect the current progress and achievements in the scientific literature.

3. Existing Literature on *t-way* Strategies

Generally, *t-way* strategies can be classified as algebraic or computational approaches [28, 29]. Algebraic approaches are often based on the extensions of the mathematical methods for constructing Orthogonal Arrays (OAs) [30, 31]. Examples of strategies that originate from the extension of OA include Combinatorial Test Services and TConfig. The main limitation of the OA solutions is the fact that not all solutions can be found for t>2, thus, limiting its applicability for small scale system configuration. Empirical evidence [32] suggest the need to support up to at least t=6 in order to sufficiently cater for interaction faults.

3.1. *General Computational-based Strategies*

Much existing work has placed emphasis on the computational-based approaches that provide support for very large configurations. Specifically, there are two competing approaches for constructing *t-way* test suites. That is, the one-test-at-a-time (OTAT) approach and the one-parameter-at-a-time (OPAT) approach. In the first case, the strategy iteratively traverses the required interaction and generates a complete test case per iteration. During each iteration, the strategy greedily checks whether or not the generated test case is the best fit value (i.e. covering the most uncovered interactions) to be selected in the final test suite. In the second case, the strategy constructs the test case incrementally by horizontal extension until completion. This is followed by vertical extension, if necessary, to cover the remaining uncovered interactions.

One-test-at-a-time based strategies were pioneered by AETG [33]. AETG first constructs all the required interactions then generates one final test case for every cycle, for each iteration. For each cycle, AETG generates a number of test case candidates, and from these candidates, one is greedily selected as the final test case (i.e., covering the most uncovered interactions). Over the years, a number of variations of AETG have emerged including mAETG [27] and mAETG_SAT [34]. Similar to AETG, GTWay [35, 36] also adopts the one-test-at-a-time approach to generate the final test suite. Unlike AETG, GTWay permits the use of actual parameter values as a symbolic string and supports automated execution of test cases.

Claiming to be an AETG variant, the test vector generator (TVG) [37] generates test suites based on three algorithms: T-reduced, Plus-one, and Random sets. Due to limited literature, the details' concerning the implementation for each algorithm remains unclear. However, based on our experience with TVG implementation, T-reduced often produces the greater number of optimal results compared with other algorithms.

Jenny [38] adopts the one-test-at-a-time approach by first generating a test suite that covers the 1-way interaction. Later, the test suite was extended to cover 2-way interactions and the process was repeated until all *t-way* interactions (where t is specified by the user) are covered. At around the same time, Hartman developed the Intelligent Test Case Handler (ITCH) [39] as an Eclipse Java plug-in tool. ITCH relies on exhaustive search to construct the test suites for *t-way* testing. Owing to its exhaustive search algorithm, ITCH's execution time typically takes a long time and results are often not optimal.

PICT [40, 41] generates all specified interactions, and randomly selects their corresponding interaction combinations to form the test cases as part of the complete test suite. Due to its random behavior, PICT tends to give poor test sizes as compared to other strategies.

Classification-Tree Editor eXtended Logics (CTE-XL) [42, 43] is a *t-way* strategy based on the Classification-Tree Method (CTM). The idea is that CTM abstracts and separates the test object's input domain into different subsets according to features that the test engineer considers relevant to the test. Then, test cases are produced by combining subsets from different classifications as one-test-at-a-time.

Complementing the one-test-at-a-time approach, the in-parameter-order (IPO) strategy [44] is a strategy that adopts the one-parameter-at-a-time approach. IPO generates a pairwise test set for the first two parameters, and then extends the test set by generating the pair for the first three parameters and so on, until all the system parameters are covered. This is followed by a vertical extension to cover the uncovered interactions, if necessary. The IPO strategy was later generalized into a number of variants; IPOG [45], IPOG-D [46] and IPOF [47]. Owing to its simplicity, IPO has been adopted by other researchers, notably in the development of MIPOG [48-50]. Unlike IPO and its family, MIPOG removes inherent dependencies between horizontal and vertical extensions in order to permit parallel *t-way* test suite generation on multiple-core machines.

3.2. *Meta-heuristic-based Strategies*

Recently, efforts have been focused on the use of meta-heuristic algorithms as part of the computational approach for *t-way* test generation. Strategies adopting meta-heuristic algorithms as the basis of *t-way* strategies appear to be superior to other computational approaches. Its popularity has increased due to the interest in Search based Software Engineering [4].

Generally, meta-heuristic algorithms start with a random set of solutions. These solutions undergo a series of transformations in an attempt to improve them. One best candidate is selected at each iteration until all the required interactions are covered. Concerning *t-way* test generation, a number of meta-heuristic algorithms have been explored as the basis for *t-way* strategies including Genetic Algorithms (GA), Ant Colony Optimization (ACO), Simulated Annealing (SA), Particle Swarm Optimization (PSO) and Cuckoo Search (CS).

GA, ACO, and SA represent early attempts to utilize meta-heuristic algorithms for constructing *t-way* strategies. A GA [5] mimics the natural selection processes. It begins with randomly created test cases, referred to as chromosomes. These chromosomes undergo crossover and mutation until a termination criteria is met. In each cycle, the best chromosomes are (probabilistically) selected and added to the final test suite. Unlike GAs, ACO [5] mimic the behavior of ants in their search for food. SA [51] relies on a large random search space and probability-based transformation equations for generating a *t-way* test suite.

Although useful for addressing small values of uniform interaction strength t (i.e., $t \leq 3$), strategies based on GA, ACO, and SA are not without their limitations. GA and ACO have been criticized for their complex algorithm structure as well as potentially requiring large computational resources. SA, being a single solution meta-heuristic, can be sensitive to its initial starting point in the search space, hence, prone to suffer from early convergence. For these reasons, these algorithms have been limited to small interaction strengths (i.e., $t \leq 3$). Addressing earlier limitations of SA, an improved variant of SA, called CASA [52], has been developed to address the *t-way* test generation for software product lines testing.

PSTG [53-56] is a meta-heuristic *t-way* strategy based on Particle Swarm Optimization, which mimics the swarm behavior of birds. Internally, PSTG iteratively performs local and global searches to find the candidate solution to be added to the final suite until all the interaction tuples are covered. Unlike other AI-based strategies, that address small values of t (i.e., $2 \leq t \leq 3$), the most notable feature of PSTG is the fact that it can support up to $t = 6$.

Complementary to PSTG, HSS [8] is a meta-heuristic strategy based on the Harmony Search Algorithm (HSS). HSS mimics musicians trying to compose good music from improvisations to create the best tune from their memory or from random sampling. In doing so, HSS iteratively exploits the Harmony memory to store the best found solution through a number of defined improvisations within its local and global search process. In each improvisation, one test case will be selected to be the final test suite until all the required interactions are covered. Unlike PSTG, HSS addresses the support for forbidden combinations (or constraints).

Cuckoo Search (CS) [11] is a recent strategy for *t-way* test generation. At the start, the algorithm generates random initial nests. Each egg in a nest represents a vector solution (i.e. a test case). At each generation, two operations are performed. Firstly, a new nest is generated (typically through Levy Flight path) and evaluated against the existing nests. The new nest will replace the current nest, if it has a better objective function. Secondly, CS has probabilistic elitism in order to maintain elite solutions for the next generation.

Existing meta-heuristic based strategies have been successful as the basis of *t-way* strategies. Extending and complementing existing works, this paper proposes combining more than one meta-heuristic as part of a *t-way* strategy. Instead of taking one meta-heuristic algorithm, our approach takes four algorithms to form the basis of our strategy, called High Level Hyper-Heuristic (HHH). We utilize four recently developed meta-heuristic algorithms, these being Teaching Learning based Optimization (TLBO), Global Neighborhood Algorithm (GNA), Particle Swarm Optimization (PSO) and Cuckoo Search Algorithm (CS).

4. The Proposed HHH Strategy

The proposed HHH strategy utilizes Tabu Search as the high level meta-heuristic (HLH), incorporating a selection and acceptance mechanism based on three defined operators (i.e. improvement, diversification and intensification operator). The HHH strategy is illustrated in Figure 4. The algorithmic details are provided in the next sections.

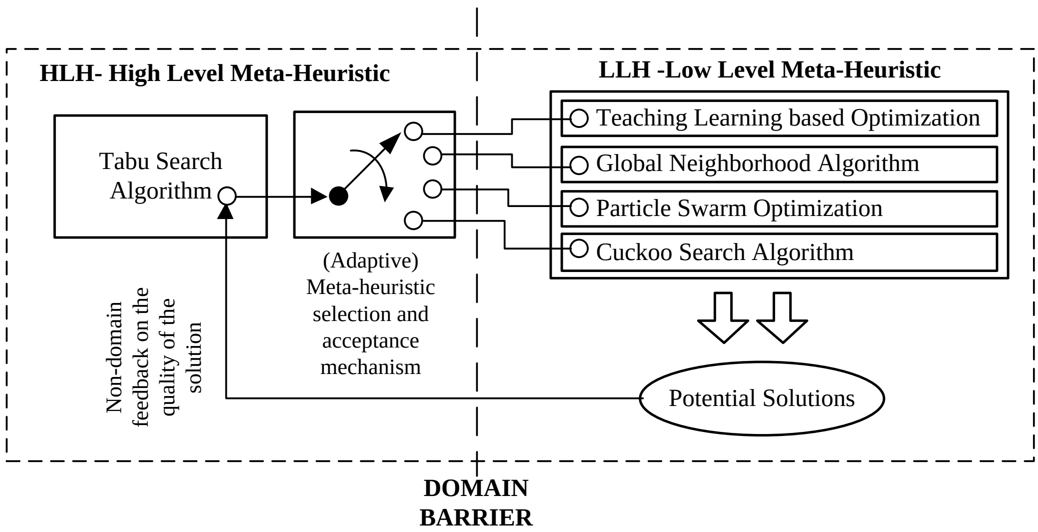


Figure 4. The HHH Strategy

4.1. *The Tabu Search HLH*

The pseudo code for the Tabu Search HLH is shown in Figure 5.

Input*: interaction strength (t), parameter (k) and its corresponding value (v)*
Output*: final suite, TS*
1: Initialize the population of the required t-way interaction tuples, $I = \{I_1, I_2 \ldots I_M\}$
2: Initialize $Tabu_{max}$, F_{max} *fitness evaluation limit ,* Θ_{max} *iteration, and population size S*
3: Initialize the population of solutions, $Z = \{Z_1, Z_2 \ldots Z_S\}$
4: Select a meta-heuristic algorithm H_i *from the pool of meta-heuristics*
5: While all interaction tuples (I) are not covered and F_{max} *fitness evaluation limit is not reached*
6: {
7: Update the population of solutions Z by the selected heuristic algorithm H_i *with* Θ_{max} *iteration*
8: Obtain Z_{best} *from the population Z and add to the final suite TS*
9: F_1 *= Improvement_Operator(Z)*
10: F_2 *= Diversify_Operator(Z)*
11: F_3 *= Intensify_Operator(Z)*
12: If the meta-heuristic selection and acceptance mechanism evaluation, $\psi(H_i,F_1,F_2,F_3)$*, improves*
13: Keep H_i
14: Else
15: {
16: Add H_i *to Tabu List*
*17: If (Tabu List is Full) // * $Tabu_{max}$
18: Randomly select a new meta-heuristic algorithm H_{i^*} *where* $i^* \neq i$ *from Tabu List*
19: Else
20: Select a new meta-heuristic algorithm H_i *from the pool of meta-heuristics*
21: }
22: }

Figure 5. The Tabu Search HLH

Line 1 initializes the population of the *t-way* interaction tuples, $I = \{I_1, I_2 \ldots I_M\}$. The value of M depends on the given input interaction strength (t), parameter (k) and its corresponding value (v). Specifically, M captures the number of required interactions that needs to be covered in the final test suite. M can be obtained as the sum of products of each individual's *t-way* interaction. For example, for CA (9; 2, 3^4), M takes the value of 3x3+3x3+3x3+3x3+3x3+3x3 = 54. If MCA (9; 2, 3^2 2^2) is considered, then M takes the value of 3x3+ 3x2+3x2+3x2+3x2+2x2= 37. Line 2 defines the maximum iteration Θ_{max} and population size, N. Line 3 randomly initializes the initial population of solutions $Z = \{Z_1, Z_2 \ldots Z_N\}$. Line 4 selects the initial LLH from the four available meta-heuristics. The selected hyper-heuristic algorithm (LLH), H_i will then be performed repeatedly until all the interactions in I has been covered and the maximum fitness function limit, F_{max}, has been reached, as shown in lines 5-22. The selected LLH H_i will update the population Z for Θ_{max} iterations, as shown in line 7. In line 8, Z_{best} (the individual with the best quality) is added to the final test suite (TS). To decide whether to select a new LLH or not, the three operators, comprising the improvement, diversification and intensification operator (lines 9-11) will be used. More precisely, the three operators work as follows.

The Improvement Operator

The *improvement operator* compares the current Z_{best} against the previous Z_{best} from the final test suite *TS*. F_1 evaluates to *true only if* $Z_{best} >=$ *previous* Z_{best}.

The Diversification Operator

The diversification operator exploits the hamming distance measure to evaluate the diversification of each Z_{best} solution (i.e. in terms of how far Z_{best} is from the population of candidate solutions). The hamming distance measure between two rows of Z, $d(Z_i,Z_j)$ is defined as the number of values in which they differ. Referring to Figure 6 and assuming $Z_{best} = Z_3$, the hamming distance between Z_1 and Z_{best} is $d(Z_1,Z_{best}) = 4$ (since all the values differ) and the hamming distance between Z_2 and Z_3 is $d(Z_2,Z_{best}) = 3$ (with three values being different). In this case, the diversification value d_v can be defined as the cumulative sum of the hamming distance measure of each individual Z population with Z_{best}. Here, the value of d_v =7 (i.e. as the sum of $d(Z_1,Z_{best})$ and $d(Z_2,Z_{best})$).

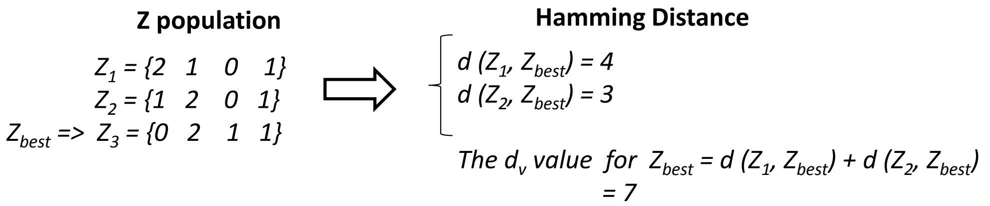


Figure 6. Diversification value and Hamming Distance

As far as the diversification operator is concerned, the current value of d_v will be compared to the previous value of d_v (i.e. from the previous iteration). F_2 evaluates to *true only if the current* d_v *>= the previous* d_v.

The Intensification Operator

Like the diversify operator, the intensification operator also exploits the hamming distance to evaluate the intensification of each Z_{best} solution. Unlike the diversification operator, the intensification operator measures the intensification value, I_v, of Z_{best} against the final test suite *TS* population (i.e. how close is Z_{best} to the final test suite). To be more specific, the intensification value can be defined as the cumulative sum of the hamming distance of each individual *TS* population with Z_{best}. Here, the current value of I_v will be compared to the previous value of I_v (i.e. from the previous iteration). F_3 evaluates to *true only if the current* I_v *<= the previous* I_v.

In line 12 (Figure 5), the meta-heuristic selection and acceptance mechanism, $\psi(H_i,F_1,F_2,F_3)$ evaluates to *true, if and only if,* F_1*=true and* F_2 *= true and* F_3*=true*. If $\psi(H,F_1,F_2,F_3)$ evaluates to *false*, the new H_i will be selected (and the current H_i will be put in the Tabu List). Visually, the internal working of the meta-heuristic selection and acceptance mechanism, ψ, is shown in Figure 7.

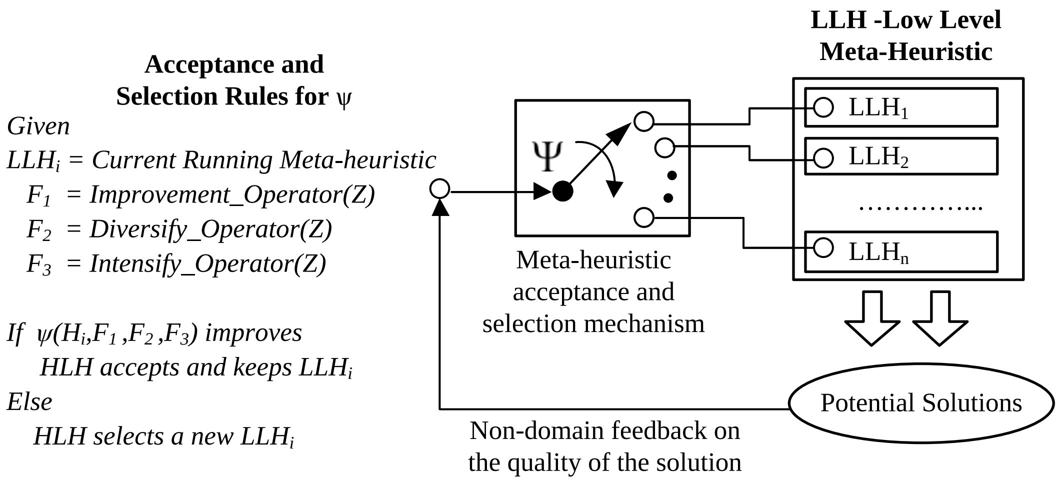


Figure 7. Acceptance and Selection Rules

Referring to lines 17-20, the current H_i is penalized and will miss at least one turn from being selected in the next iteration. Apart from one's own performance in terms of objective value improvement, diversification, and intensification, a particular LLH can be chosen more frequently than others owing to the random selection of meta-heuristics within the Tabu search (line 18).

Finally, it is worth mentioning here that the adopted LLHs are designed for continuous problems. As such, to deal with discrete parameters and values, each individual Z_j needs to capture the parameters as a valid range of integer numbers (i.e. based on the user inputs). When the HHH iterates, each Z_j will be updated accordingly depending on the chosen LLH (i.e. based on the specific LLH transformation equation). Here, the LLH update may result into the need to do rounding off floating point values. Apart from rounding off floating point values, there is also the need to deal with out-of-range values. Within HHH, we establish the boundary condition (i.e. as *clamping rule*) to restrict parameter values to both lower and higher bounds. In this way, when Z_j moves out-of-range, the boundary condition brings it back to the search space. We configure our boundary condition in such a way that when the Z_j value reaches a certain dimensional bound, we reset its position to the other endpoint. For example, if we have a parameter with a range of values from 1 to 3, when the position is greater than 3, the position is reset to 1.

4.2. *Teaching Learning based Optimization LLH*

The Teaching Learning Based Optimization (TLBO) [14] is a population based meta-heuristic that draws on the analogy of the teaching and learning process between teachers and students. In TLBO, teachers attempt to impart knowledge in a way that will enhance the knowledge of their students. With knowledge gained from a particular teacher, the knowledge of the students would be enhanced. As teachers have different competency levels, there could be potential improvements if students learn from other teachers. Students can also learn from other students, yielding similar improvements.

The TLBO algorithm is shown in Figure 8.

Input*: the population* $Z = \{Z_1, Z_2 \ldots Z_S\}$
Output*: the updated population* $Z' = \{Z_1', Z_2' \ldots Z_S'\}$
1: Obtain Z_{mean} *from the population* $Z = \{Z_1, Z_2 \ldots Z_S\}$
2: Choose global $Z_{best\ teacher}$ *from the population* $Z = \{Z_1, Z_2 \ldots Z_S\}$
3: For i=1 to S
4: {
*/******Teacher Phase******/*
5: Obtain teaching factor $T_i^F = round\ (1+ rand(0,1)\{2-1\})$
6: Update the current population according to $Z_i^{(t+1)} = Z_i^{(t)} + rand(0,1)(Z_{best\ teacher} - (T_i^F \cdot Z_{mean})$ *subjected to clamping rule*
7: If $Z_i^{(t+1)}$ *is better than* $Z_i^{(t)}$*, i.e.* $f(Z_i^{(t+1)})>f(Z_i^{(t)})$
8: $Z_i^{(t)} = Z_i^{(t+1)}$
*/******Learner Phase******/*
9: Randomly select Z_j *such that* $j \neq i$
10: If $Z_i^{(t)}$ *is better than* Z_j *, i.e.* $f(Z_i^{(t)})>f(Z_j)$
11: $Z_i^{(t+1)} = Z_i^{(t)} + rand(0,1)(Z_i^{(t)} - Z_j)$
12: Else
13: $Z_i^{(t+1)} = Z_i^{(t)} + rand(0,1)(Z_j^{(t)} - Z_i)$
14: If $Z_i^{(t+1)}$ *is better than* $Z_i^{(t)}$ *, i.e.* $f(Z_i^{(t+1)})>f(Z_i^{(t)})$
15: $Z_i^{(t)} = Z_i^{(t+1)}$
16: }
17: return Z

Figure 8. The Teaching Learning based Optimization LLH

During the teacher phase (lines 7-10), each student learns from the best teacher $Z_{best\ teacher}$, that is, the best individual in the population. The student will move toward the $Z_{best\ teacher}$ teacher by taking into account the current mean value of the learners, Z_{mean}, that represents the qualities of all the students in the population (line 8). The movement of the learner is also affected by the teaching factor, T_i^F (line 7).

For the learner phase (lines 11-18), each student attempts to improve its knowledge through interaction with its peers. Specifically, the student Z_i will select a peer learner Z_j (where i≠j). If Z_i has better fitness than Z_j, the latter is moved toward the former (line 15) and vice versa (line 16).

Concerning its implementation, unlike most meta-heuristic algorithms, TLBO has twice more fitness function evaluations due to the two phases (teacher and learner). For this reason, direct comparative experiments with TLBO and other meta-heuristic algorithms can be misleading. Additionally, despite its parameter free claims, TLBO still requires tuning of its population size and iteration. Recently, the original author for TLBO [14] has been criticized for not reporting the duplicate elimination step in the implementation resulting into inaccurate and unfair comparative results [57]. For our implementation, we consider the TLBO implementation without the duplicate elimination step (similar to that of Yarpiz [58]).

4.3. *Global Neighborhood Algorithm LLH*

Like TLBO, the Global Neighborhood Algorithm (GNA) [15], is a population based meta-heuristic. GNA has only two control parameters; population size and the maximum number of iterations. The GNA algorithm is shown in Figure 9.

```
Input: the population Z = {Z_1, Z_2… Z_S}
Output:  the updated population Z' = {Z_1', Z_2'… Z_S'}
1: For i=1 to S
2:  {
        /******Local Search******/
3:      If i < (N/2)
4:        {
5:           Generate a new solution according to Z_i^(t+1) = pertubate one value from Z_i^(t)
6:           If Z_i^(t+1) is better than Z_i^(t), i.e. f(Z_i^(t+1))>f(Z_i^(t))
7:                Z_i^(t) = Z_i^(t+1)
8:        }
        /******Global Search******/
9:     Else
10:      {
11:         Generate a random solution Z_random
12:         If f(Z_random) is better than Z_i^(t), i.e. f(Z_random)>f(Z_i^(t))
13:               Z_i^(t) = Z_random
14:     }
15:  }
16:  return Z
```

Figure 9. The Global Neighborhood Algorithm LLH

In GNA, the population is divided into two phases. In the first phase, GNA performs a local search through perturbation of Z_i (line 5). If the pertubated Z_i has a better fitness value, then the incumbent is replaced (lines 4-5).

In the second phase, GNA performs a random search. If Z_{random} has better fitness value than the current Z_i, Z_i will be replaced with Z_{random} (lines 10-11).

4.4. *Particle Swarm Optimization LLH*

Particle Swarm Optimization (PSO) [16] is a population based meta-heuristic that simulates the swarm behavior of flocks of birds or schools of fish. PSO comprises a group of particles with negligible mass and volume and which move through hyperspace. Each particle attempts to find a better position (solution) by recording and updating essential information about its movement. This information is related to the i^{th} particle of interest, which includes the current position (Z_i), the velocity (V_i), local best (Z_{lbest}) and global best (Z_{gbest}). The PSO algorithm is shown in Figure 10.

***Input**: the population $Z = \{Z_1, Z_2 \ldots Z_S\}$*
***Output**: the updated population $Z' = \{Z_1', Z_2' \ldots Z_S'\}$*
1: Initialize the inertial weight ω, and the learning factors c_1, c_2
2: Initialize the population velocity $V = \{V_1, V_2 \ldots V_S\}$
3: Choose global Z_{gbest} from the population $Z = \{Z_1, Z_2 \ldots Z_S\}$
4: Set local best $Z_{lbest} = Z_{gbest}$
5: For i=1 to S
6: {
7: Update the current velocity according to $V_i^{(t+1)} = \omega\, V_i^{(t)} + c_1\, rand(0,1)(Z_{lbest}^{(t)} - X_i^{(t)}) + c_2\, rand(0,1)(Z_{gbest} - X_i^{(t)})$
8: Update the current population according to $Z_i^{(t+1)} = Z_i^{(t)} + V_i^{(t+1)}$ subjected to clamping rule
9: If $Z_i^{(t+1)}$ is better than $Z_i^{(t)}$, i.e. $f(Z_i^{(t+1)}) > f(Z_i^{(t)})$
10: $Z_i^{(t)} = Z_i^{(t+1)}$
*/******Update Global Best******/*
11: If $Z_i^{(t)}$ is better than Z_{gbest}, i.e. $f(Z_i^{(t)}) > f(Z_{gbest})$
12: $Z_{gbest} = Z_i^{(t)}$
13: }
14: return Z

Figure 10. The Particle Swarm Optimization LLH

During the search, each Z_{ith} particle of the population stochastically adapts its trajectory through velocity its local best (Z_{lbest}) and global best value ($Z_{global\ best}$) through velocity (V_i) as indicated in lines 7-8. In turn, the velocity (V_i) exploits three coefficients c_1, c_2, and ω respectively. Here, c_1 and c_2 are the acceleration coefficients that control the personal and global best to the updated velocity; and ω is the inertia weight that is used to balance the global/local searches of the particle. In lines 9-10, if the updated Z_i has better fitness than current Z_i, the incumbent is updated. Finally, Z_{best} will be assigned to Z_i if the latter has better fitness (lines 11-12).

4.5. *The Cuckoo Search Algorithm LLH*

The Cuckoo Search algorithm (CS) [17] is a population based meta-heuristic algorithm that is based on the parasitic behavior and aggressive reproduction strategy of Cuckoos. The female Cuckoos lay their eggs (potential solutions in the algorithm) in the nest of other birds and have the ability to imitate the colors and pattern of the host eggs. Cuckoos also have the ability to remove existing (the host birds, or other cuckoos) from the nest. In the algorithm, this is akin to replacing a poorer solution with a better one. The CS algorithm is shown in Figure 11.

.

Input: *the population* $Z = \{Z_1, Z_2 \ldots Z_S\}$
Output: *the updated population* $Z' = \{Z_1', Z_2' \ldots Z_S'\}$
0: *Set* α *and elitism factor* p_a
1: *For i=1 to S*
2: {
3: *Generate new eggs according to* $Z_i^{(t+1)} = Z_i^{(t)} + \alpha \oplus$ *Lévy Flight*
subjected to clamping rule
4: *If* $Z_i^{(t+1)}$ *is better than* $Z_i^{(t)}$, *i.e.* $f(Z_i^{(t+1)}) > f(Z_i^{(t)})$
5: $Z_i^{(t)} = Z_i^{(t+1)}$
6: }
/******Maintain Elitism******/
7: *For j=1 to* $(p_a \cdot S)$
8: {
9: *Generate random* Z_{random}
10: *Get the worst* $Z_{worst}^{(t)}$
11: *If* $f(Z_{random})$ *is better than* $Z_{worst}^{(t)}$, *i.e.* $f(Z_{random}) > f(Z_{worst}^{(t)})$
12: $Z_{worst}^{(t)} = Z_{random}$
13: }
14: *return Z*

Figure 11. The Cuckoo Search LLH

The CS algorithm provides two search capabilities: global search, which allows the algorithm to jump out of local optimum, and local search by intensifying search around the current best solution, via *Lévy Flight* motion. The *Lévy Flight* motion is a random walk that takes a sequence of jumps, which are selected from a heavy tailed probability function. For our *Lévy Flight* implementation, we adopt the well-known Mantegna's algorithm [17]. Within this algorithm, a step length can be defined as:

$$Step = \frac{u}{[v]^{\frac{1}{\beta}}} \qquad (3)$$

where u and v are approximated from the normal Gausian distribution in which:

$$u \approx N(0, {\sigma_u}^2) \cdot \sigma_u \qquad v \approx N(0, {\sigma_v}^2) \cdot \sigma_v \qquad (4)$$

For v value estimation, we use $\sigma_v = 1$. For u value estimation, we evaluate the Gamma function(Γ) with the value of $\beta = 1.5$ [59], and obtain σ_u using:

$$\sigma_u = \left| \frac{\Gamma(1+\beta) \cdot \sin(\frac{\pi\beta}{2})}{\Gamma(\frac{1+\beta}{2}) \cdot \beta \cdot 2^{\frac{(\beta-1)}{2}}} \right|^{\frac{1}{\beta}} \qquad (5)$$

In our case, the Gamma function(Γ) implementation is adopted from William et al [60].

During the search, Z_i will be continuously updated from its previous value with the appropriate step size (α), entry-wise multiplication ($\oplus$) and *Lévy Flight* motion as indicated in line 3. If the updated Z_i has better fitness than the current Z_i, the former is updated (lines 4-5). CS also maintains an elite population of solutions (lines 7-13). CS will replace a fraction $p_a \cdot S$ of the poorly generated Z_{worst} solutions (where p_a is the elitism factor ranging from 0.0<*pa*<1.0) with a random solution Z_{random}, iteratively if the corresponding fitness improves.

It is worth mentioning here that the number of fitness function evaluation for CS cannot be determined statistically owing to the probability p_a. However, a counter can be put during run-time to determine the exact number of fitness evaluation (so as to have the same fitness evaluation as other LLHs).

5. HHH Parameter Calibration

As far as calibration is concerned, there are generally two types of parameters of concern; specific algorithm parameters and common algorithm parameters. The former relates to the algorithm settings of the four LLHs, whilst the latter accounts for the common parameters between LLHs and HLH.

Table 2 summarizes the parameters that are to be calibrated.

Table 2. Specific and Common Algorithm Parameters

	Specific Algorithm Parameters	Common Parameters
Tabu Search HLH	$Tabu_{max} = 4$	Max iteration, $\Theta_{max} = 20$ Population size, $S = 40$ Maximum Fitness Evaluation, $F_{max} = 400{,}000$
Teaching Learning based Optimization LLH	-	
Global Neighborhood Algorithm LLH	-	
Particle Swarm LLH	$c_1 = 1.375$, $c_2 = 1.375$, $\omega = 0.3$	
Cuckoo Search LLH	$p_a = 0.25$	

In the design of HHH, the specific algorithm parameters are the $Tabu_{max}$ of the Tabu lists for TS HLH, c_1,c_2 (i.e. acceleration coefficients) and ω (inertial weight) for PSO LLH as well as p_a (elitism factor) for CS LLH. As for TLBO LLH and GNA LLH, their parameters are Θ_{max} *and* S (which are also shared by PSO LLH, Cuckoo LLH and Tabu HLH as well). As far as the calibration of specific algorithm parameters is concerned, the value of $Tabu_{max}$ = 4 can be easily deduced as the number of adopted meta-heuristics as LLH. Any particular meta-heuristic can either be available for selection or penalized in the Tabu list depending on its prior performance. Meanwhile, the value of c_1= 1.375, c_2= 1.375, ω = 0.3 and p_a = 0.25 are adopted from existing work reported in the context of adopting PSO [7] and CS [11] as *t-way* `test generation strategy respectively.

Unlike the specific parameters, the calibration of common parameters Θ_{max} iteration and population size, S, can be subtle owing to the way each individual LLH operates. As highlighted earlier, TLBO requires twice as much fitness function evaluation as GNA and PSO. Furthermore, Cuckoo also has non-deterministic number of fitness evaluation owing to the elitism probability p_a. This large diversification makes it hard to select the appropriate value for both Θ_{max} and S to ensure fair comparison with other meta-heuristic based strategies particularly when it comes to the fitness function evaluation. For this reason, we decide to establish the limit on the maximum fitness function evaluation, called F_{max}, as seen in Table 2. Specifically, we define F_{max} = 400,000 as it was empirically verified to be large enough to allow HHH's solution to converge even for the largest configurations. Internally, we have implemented a global static counter (as part of fitness function evaluation) that ensures the F_{max} limit is adhered to.

Having established the F_{max} = 400,000 as the stopping criterion, the selection of Θ_{max} and S follow accordingly. In this case, any values of Θ_{max} and S can be selected as long as the value of F_{max} is observed. For example, if the selected values of Θ_{max} =20 and S=40, the minimum fitness evaluation per iteration is at least 20*40=800 and the maximum possible iteration for convergence is 400,000/800 = 500. In similar manner, if the selected values of Θ_{max} =20 and S=100, the minimum fitness evaluation per iteration is at least 20*100=2000 and the maximum possible iteration for convergence is 400,000/2000 = 200. For our case, considering the recommendation from the scientific papers in [61, 62], we opt to adopt the former case with Θ_{max} =20 and S=40.

6. Evaluation

Our evaluation focuses on two related goals. Firstly, we compare the performance of HHH against existing strategies. Then, we verify our findings using statistical analysis. In our evaluation, we note that the comparative performances with the same number of objective function evaluations are not possible for meta-heuristic-based strategies (i.e. most implementations are not publically available; hence, the settings of each of the algorithm parameters are beyond our controls).

6.1. *Benchmarking with Existing Strategies on Test Sizes*

We compare HHH against existing strategies based on the benchmark experiments in terms of test sizes as defined in [9, 11, 56, 63-68]. Considering that the test sizes are absolute and not affected by the computational platform, our comparison spans many *t-way* strategies from meta-heuristic-based to general computation ones. Concerning the comparison between meta-heuristic-based strategies, much criticism can be highlighted in terms of the way the results are presented in the scientific literature. In particular, important information related to individual strategy execution involving the average results as well as the maximum number of fitness evaluations (F_{max}) are often missing, thus, hindering a fair comparison between them.

To put our work into perspectives, we highlight all the algorithm parameters for the meta-heuristic-based strategies of interests obtained from their respective publications (as depicted in Table 3).

Table 3. Algorithm Parameters for Existing Meta-heuristic-based Strategies of Interests

Strategies	Parameters	Values
HSS [8]	*Max Improvisation/Iteration*	*1000*
	Harmony Memory Size	*100*
	Harmony Memory Considering Rate	*0.7*
	Pitch Adjustment Rate	*0.2*
PSTG [6, 7]	*Max Iteration*	*100*
	Population Size	*80*
	Acceleration Coefficients	*1.375*
	Inertia Weight	*0.3*
CS [17]	*Max Iteration*	*100*
	Population Size	*100*
	Probability p_a	*0.25*
SA [51]	*Max Iteration*	*100*
	Starting Temperature	*20*
	Cooling Schedule	*0.9998*
GA [5]	*Max Iteration*	*1000*
	Population Size	*25*
	Best Cloned	*1*
	Tournament Selection	*0.8*
	Random Crossover	*0.75*
	Gene Mutation	*0.03*
	Max Stale Period	*3*
	Escape Mutation	*0.25*
ACO [5]	*Iteration*	*1000*
	Number of Ants	*20*
	Pheromone Control	*1.6*
	Pheromone Persistence	*0.5*
	Pheromone Amount	*0.01*
	Initial Pheromone	*0.4*
	Heuristic Control	*0.2*

	Elite Ants	*2*
	Max Stale Period	*5*

We do not compare the performance in terms of execution times as most implementations are not available to be executed on the computer we used (a desktop PC with Windows 8, 2.5 GHz i5 CPU, 8 GB of RAM). Furthermore, in the case of meta-heuristic-based strategies, the time execution comparison can also be unfair [69-72] as the number of fitness evaluation varies significantly for each strategies.

Owing to its non-deterministic nature, HHH is executed 30 times for each experiment. The parameter settings are based on the values obtained earlier (with $Tabu_{max}$ =4, c_1=1.375, c_2=1.375, ω=0.3, p_a=0.25, S=20, Θ_{max} =40 and F_{max} = 400,000). Both the best and average test sizes for each experiment are reported side-by-side. To facilitate discussion, whenever possible, we have grouped the experiments into meta-heuristic-based and general computation-based strategies respectively. Bold cell entries indicate the best performance size whilst cell entries marked NA (Not Available) indicate that the results are unavailable in the scientific literature. The complete results are summarized in Tables 4 through 11.

Table 4. Benchmarking CA and MCA Configurations

System Configuration	Meta-heuristic-based Strategies								General Computational-based Strategies				
	HHH		HSS	PSTG	CS	SA	GA	ACO	mAETG	AETG	IPOG	Jenny	TVG
	Best	Ave											
CA(N; 2, 3^4)	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	10	11
CA(N; 2, 3^{13})	17	18.55	18	17	20	16	17	17	17	**15**	20	20	19
CA(N; 2, 5^{10})	**42**	43.9	43	45	NA	NA	NA	NA	NA	NA	50	45	51
CA(N; 3, 3^6)	**33**	38.25	39	42	43	**33**	**33**	**33**	38	47	53	51	49
CA(N; 3, 4^6)	**64**	69.35	70	102	105	**64**	**64**	**64**	77	105	64	112	123
CA(N; 3, 6^6)	325	329.35	336	338	350	**300**	331	330	330	343	382	373	407
CA(N; 3, 5^7)	217	220.35	236	229	253	**201**	218	218	218	229	274	236	271
MCA(N; 2, 5^1 3^8 2^2)	20	21.35	21	21	21	**15**	**15**	16	20	19	19	23	22
MCA(N; 2, 7^1 6^1 5^1 4^6 3^8 2^3)	48	51.00	50	48	51	**42**	**42**	**42**	44	45	43	50	51
MCA(N; 2, 6^1 5^1 4^6 3^8 2^3)	36	39.25	38	39	43	**30**	33	32	35	34	35	40	43
MCA(N; 3, 5^2 4^2 3^2)	**100**	108.35	120	125	NA	**100**	108	106	114	NA	111	131	136
MCA(N; 3, 10^1 6^2 4^3 3^1)	382	390.60	378	385	393	**360**	**360**	361	377	NA	383	399	414
MCA(N; 4, $3^4 4^5$)	**427**	434.20	436	447	NA	NA	NA	NA	NA	NA	463	457	487
MCA(N; 4, 5^1 3^8 2^2)	**283**	286.80	286	292	NA	NA	NA	NA	NA	NA	324	303	313
MCA(N; 4, 8^2 7^2 6^2 5^2)	**4305**	4323.05	4395	4506	NA	NA	NA	NA	NA	NA	4776	4580	5124
MCA(N; 4, 6^5 5^4 3^2)	**2436**	2446.50	2520	3154	NA	NA	NA	NA	NA	NA	3273	3033	2881
MCA(N; 4, 10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1)	**5873**	5912.75	5915	5906	NA	NA	NA	NA	NA	NA	5492	6138	6698

Table 5. CA (N; t, 2^{10}) with t varied from 2 to 6

t	Meta-Heuristic-based Strategies					General Computational-based Strategies				
	HHH		HSS	PSTG	CS	IPOG	ITCH	Jenny	TConfig	TVG
	Best	Ave								
2	8	8.25	7	8	8	10	**6**	10	9	10
3	**16**	16.2	**16**	17	**16**	19	18	18	20	17
4	**36**	39.45	37	37	**36**	49	58	39	45	41
5	**79**	80.6	81	82	**79**	128	NA	87	95	84
6	**153**	156.15	158	158	157	352	NA	169	183	168

Table 6. CA (N; t, 5^{10}) with t varied from 2 to 6

t	Meta-Heuristic-based Strategies					General Computational-based Strategies								
	HHH		HSS	PSTG	CS	IPOG	ITCH	Jenny	PICT	TConfig	TVG	GTWay	MIPOG	CTE-XL
	Best	Ave												
2	**43**	44.2	**43**	45	45	50	45	45	47	48	50	46	45	50
3	280	282.95	276	287	297	313	**225**	290	310	312	342	293	281	347
4	**1638**	1642.21	1624	1716	1731	1965	1750	1719	1812	1878	1971	1714	1643	NA
5	8704	8704.01	8866	9425	9616	11009	NA	9437	9706	NA	NA	9487	**8169**	NA
6	47800	48300.32	47550	50350	50489	57290	NA	NA	47978	NA	NA	**44884**	45168	NA

Table 7. CA (N; 4, 5^k) with k varied from 5 to 12

k	Meta-Heuristic-based Strategies					General Computational-based Strategies							
	HHH		HSS	PSTG	CS	IPOG	ITCH	Jenny	PICT	TConfig	TVG	GTWay	MIPOG
	Best	Ave											
5	746	754.45	751	779	776	908	**625**	837	810	773	849	731	**625**
6	967	976.30	990	1001	991	1239	**625**	1074	1072	1092	1128	1027	**625**
7	1151	1159.25	1186	1209	1200	1349	1750	1248	1279	1320	1384	1216	**1125**
8	**1320**	1327.00	1358	1417	1415	1792	1750	1424	1468	1532	1595	1443	1384
9	**1483**	1488.85	1530	1570	1562	1793	1750	1578	1643	1724	1795	1579	1543
10	1635	1642.05	**1624**	1716	1731	1965	1750	1719	1812	1878	1917	1714	1643
11	1784	1786.15	1860	1902	2062	2091	1750	1839	1957	2038	2122	1852	**1722**
12	1915	1925.1	2022	2015	2223	2285	**1750**	1964	2103	NA	2268	2022	1837

Table 8. CA (N; 4, v^{10}) with v varied from 2 to 7

v	Meta-Heuristic-based Strategies					General Computational-based Strategies							
	HHH		HSS	PSTG	CS	IPOG	ITCH	Jenny	PICT	TConfig	TVG	GTWay	MIPOG
	Best	Ave											
2	36	39.45	37	34	**28**	49	58	39	43	45	40	46	43
3	**207**	209.51	211	213	211	241	336	221	231	235	228	224	217
4	668	670.52	691	685	698	707	704	703	742	718	782	**621**	637
5	1635	1642.05	**1624**	1716	1731	1965	1750	1719	1812	1878	1917	1714	1643
6	**3405**	3410.42	3475	3880	3894	3935	NA	3519	3735	NA	4159	3514	3657
7	6412	6505.34	6398	NA	NA	7061	NA	6462	NA	NA	7854	6459	**5927**

Table 9. CA (N; t, v^7) with variable values $2 \leq v \leq 5$, with t varied up to 6

t	*V*	Meta-Heuristic-based Strategies					General Computational-based Strategies						
		HHH		HSS	PSTG	CS	Jenny	TConfig	ITCH	PICT	TVG	CTE-XL	IPOG
		Best	Ave										
2	2	7	7.00	7	**6**	**6**	8	7	**6**	7	7	8	8
	3	**14**	15.20	**14**	15	15	16	15	15	16	15	16	17
	4	**23**	24.9	25	26	25	28	28	28	27	27	30	28
	5	**35**	36.35	**35**	37	37	37	40	45	40	42	42	42
3	2	15	15.0	**12**	13	**12**	14	16	13	15	15	15	19
	3	49	50.3	50	50	49	51	55	**45**	51	55	54	57
	4	**112**	115.4	121	116	117	124	**112**	**112**	124	134	135	208
	5	**216**	219.7	223	225	223	236	239	225	241	260	265	275
4	2	31	32.35	29	29	**27**	31	36	40	32	31	NA	48
	3	**148**	153.25	155	155	155	169	166	216	168	167		185
	4	**482**	484.85	500	487	487	517	568	704	529	559		509
	5	**1153**	1160.40	1174	1176	1171	1248	1320	1750	1279	1385		1349
5	2	58	58.40	**53**	**53**	**53**	57	56	NA	57	59	NA	128
	3	**435**	439.35	437	441	439	458	477		452	464		608
	4	1805	1815.40	1831	1826	1845	1938	**1792**		1933	2010		2560
	5	**5413**	5431.25	5468	5474	5479	5895	NA		5814	6257		8091
6	2	**64**	64.0	**64**	**64**	66	87	**64**	NA	72	78	NA	**64**
	3	**853**	922.55	916	977	973	1087	921		1015	1016		1281
	4	5478	5497.15	**4096**	5599	5610	6127	NA		5847	5978		**4096**
	5	**21107**	21159.95	21748	21595	21597	23492	NA		22502	23218		28513

Table 10. CA (N; t, 3^k) with variable number of parameters $3\leq k \leq 12$, with t varied up to 6

t	*K*	Meta-Heuristic-based Strategies					General Computational-based Strategies						
		HHH		HSS	PSTG	CS	Jenny	TConfig	ITCH	PICT	TVG	CTE-XL	IPOG
		Best	Ave										
	3	**9**	9.80	**9**	**9**	**9**	**9**	10	**9**	10	10	10	11
	4	**9**	9.00	**9**	**9**	**9**	13	10	**9**	13	12	14	12
	5	**11**	11.35	12	12	**11**	14	14	15	13	13	14	14
	6	**13**	14.20	**13**	**13**	**13**	15	15	15	14	15	14	15
2	7	**14**	15.00	15	15	**14**	16	15	15	16	15	16	17
	8	**15**	15.60	**15**	**15**	**15**	17	17	**15**	16	**15**	17	17
	9	**15**	16.30	17	17	16	18	17	**15**	17	**15**	18	17
	10	16	16.90	17	17	17	19	17	**15**	18	16	18	20
	11	17	17.75	17	17	18	17	20	**15**	18	16	20	20
	12	16	17.95	18	18	18	19	20	**15**	19	16	20	20
	4	**27**	29.45	30	30	28	34	32	**27**	34	34	34	39
	5	39	41.25	39	39	**38**	40	40	45	43	41	43	43
	6	**33**	39.00	45	45	43	51	48	45	48	49	52	53
	7	49	50.80	50	50	48	51	55	**45**	51	55	54	57
3	8	52	53.65	54	54	53	58	58	**45**	59	60	63	63
	9	**57**	57.85	59	58	58	62	64	75	63	64	66	65
	10	**60**	61.25	62	62	62	65	68	75	65	68	71	68
	11	**63**	64.45	66	64	66	65	72	75	70	69	76	76
	12	**66**	67.45	67	67	70	68	77	75	72	70	79	76
	5	**81**	86.5	94	96	94	109	97	153	100	105		115
	6	**131**	133.5	132	133	132	140	141	153	142	139		181
	7	**150**	153.3	154	155	154	169	166	216	168	172		185
4	8	**173**	175.15	174	175	**173**	187	190	216	189	192	NA	203
	9	**167**	188.65	195	195	195	206	213	306	211	215		238
	10	**207**	209.45	212	210	211	221	235	336	231	233		241
	11	**222**	225.05	223	**222**	229	236	258	348	249	250		272
	12	**238**	240.35	244	244	253	252	272	372	269	268		275
	6	**267**	287.55	310	312	304	348	305		310	321		393
	7	**432**	437.25	436	441	434	458	477		452	462		608
	8	**514**	518.05	515	515	515	548	583		555	562		634
5	9	**585**	590.75	597	598	590	633	684	NA	637	660	NA	771
	10	**656**	663.31	670	667	682	714	773		735	750		784
	11	**728**	733.55	753	747	778	791	858		822	833		980
	12	**798**	802.01	809	809	880	850	938		900	824		980
	7	**900**	927.65	977	977	963	1087	921		1015	1024		1281
	8	**1392**	1399.43	1402	1402	1401	1466	1515		1455	1484		2098
	9	**1679**	1688.35	1684	1684	1689	1840	1931	NA	1818	1849	NA	2160
6	10	**1960**	1967.75	1991	1980	2027	2160			2165	2192		2726
	11	**2230**	2240.05	2255	2255	2298	2459	NA		2496	2533		2739
	12	**2503**	2503.02	2528	2528	2638	2757			2815	2597		3649

Table 11. Four CA derived from Real Software System Configurations, *t* is varied up to 6

System Configuration	Meta-Heuristic-based Strategies					General Computational-based Strategies							
	HHH		HSS	PSTG	CS	Jenny	TConfig	ITCH	PICT	TVG	CTE-XL	IPOG-D	IPOG
	Best	Ave											
BBS													
CA(2,3^4)	**9**	9.00	**9**	**9**	**9**	13	10	**9**	13	12	10	15	12
CA(3,3^4)	**27**	29.70	**27**	**27**	**27**	34	32	**27**	34	32	37	**27**	39
TCAS													
CA(2,$2^7\,3^2\,4^1\,10^2$)	**100**	110.45	106	**100**	**100**	106	109	120	**100**	**100**	**100**	130	**100**
CA(3,$2^7\,3^2\,4^1\,10^2$)	**400**	411.05	412	**400**	**400**	413	472	2388	400	434	426	480	**400**
CA(4,$2^7\,3^2\,4^1\,10^2$)	1509	1524.75	1531	1520	1537	1536	1548	1484	1369	1599	NA	NA	**1377**
CA(5,$2^7\,3^2\,4^1\,10^2$)	4492	4492.01	4569	4566	4566	4621	NA	NA	**4250**	4773	NA	13458	4283
CA(6,$2^7\,3^2\,4^1\,10^2$)	11735	11742.80	11740	11743	11431	11625	NA	NA	**11342**	NA	NA	41280	11939
Mobile Phone													
CA(2,$2^2\,3^3$)	**9**	11.02	**9**	**9**	**9**	12	12	15	10	10	**9**	15	11
CA(3,$2^2\,3^3$)	**27**	33.25	**27**	**27**	**27**	29	30	45	29	30	32	34	**27**
CA(4,$2^2\,3^3$)	**54**	54.15	**54**	**54**	**54**	59	56	138	59	55	NA	NA	**54**
Spin Simulator													
CA(2,$2^{13}\,4^5$)	24	26.11	25	24	NA	26	29	28	23	27	26	28	**20**
CA(3,$2^{13}\,4^5$)	102	104.81	106	101	NA	111	113	196	96	111	113	112	**78**
CA(4,$2^{13}\,4^5$)	387	397.05	390	380	NA	412	427	1296	353	**288**	NA	NA	341
CA(5,$2^{13}\,4^5$)	1260	1270.43	1265	1270	NA	1304	NA	NA	1185	**842**	NA	5054	1243
CA(6,$2^{13}\,4^5$)	3636	3645.41	3641	3648	NA	3538	NA	NA	**3420**	NA	NA	30214	3516

6.2. *Statistical Analysis*

We conduct our statistical analysis for all the obtained results in Tables 4 through 11 based on multiple pairwise comparisons with 95% confidence level (i.e. α=0.05). To be specific, we adopt the Friedman tests subjecting HHH as the control strategy (i.e. 1xN pair comparison). In all the cases, the table entries with NA are ignored as their contributions are incomplete (i.e. Friedman tests must be based on complete samples). Under the null hypothesis, Friedman test states that all the strategies are the equivalent, so a rejection of this hypothesis implies the existence of differences among the performance of all the strategies studied. It must be stressed here that the differences of performances may not be necessarily involved the pairing of HHH alone (i.e. the Friedman test analysis can also signify the differences in performances for other pairing). The null hypothesis (H_0) is only rejected if the Friedman statistic (χ^2) is greater than the critical value. After this, a *post-hoc* test based on the Wilcoxon Rank-Sum could be used to find whether the control strategy presents statistical difference with regards to the remaining strategies in the comparison. The null hypothesis (H_0) is that there is no significant difference as far as the test size is concerned for HHH and each individual strategy (i.e. the two populations have the same medians). Our alternative hypothesis (H_1) is that test size for HHH is less than that of each individual strategy (i.e. HHH has a lower population median). To control the Type I - family wise error rate (FWER) due to multiple comparisons, we have adopted the Bonferroni-Holm correction for α values. In this case, the p-values are first sorted in ascending order such that $p_1 < p_2 < p_3 ... < p_i ... < p_k$. Then, α is adjusted based on:

$$\alpha_{Holm} = \frac{\alpha}{k - i + 1} \qquad (6)$$

If $p_1 < \alpha_{Holm}$, the corresponding hypothesis is rejected and we are allowed to make similar comparison for p_2. If the second hypothesis is rejected, the test proceeds with the third and so on. As soon as a certain null hypothesis cannot be rejected, all the remaining hypotheses are retained as well. The complete statistical analyses are shown in Tables 12 through 27.

Table 12. Friedman Test for Table 4

Friedman Test	Conclusion
Degree of freedom = 5, α = 0.05, Critical value = 11.0705 Friedman statistic (χ^2) = 49.099	49.099 > critical value, reject H_o and proceed to *post-hoc* test

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of CS, SA, GA, ACO, mAETG and AETG are ignored

Table 13. *Post-hoc* Wilcoxon Rank-Sum Tests for Table 4

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm}	Conclusion
HHH vs Jenny	0.0001451	0.01	p-value < α_{Holm}, Reject H_o
HHH vs TVG	0.0001454	0.0125	p-value < α_{Holm}, Reject H_o
HHH vs PSTG	0.0007959	0.0166	p-value < α_{Holm}, Reject H_o
HHH vs HSS	0.0007959	0.0166	p-value < α_{Holm}, Reject H_o
HHH vs IPOG	0.01537	0.05	p-value < α_{Holm}, Reject H_o

Table 14. Friedman Test for Table 5

Friedman Test	Conclusion
Degree of freedom = 7, α = 0.05, Critical value = 14.0671 Friedman statistic (χ^2) = 31.2134	31.2134 > critical value, reject H_o and proceed to *post-hoc* test

*Owing to incomplete sample (i.e. with one or more NA entries), the contribution of ITCH is ignored

Table 15. *Post-hoc* Wilcoxon Rank-Sum Tests for Table 5

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm}	Conclusion
HHH vs Jenny	0.02108	0.007142	p-value > α_{Holm}, Cannot reject H_o
HHH vs TVG	0.02108		Cannot reject H_o
HHH vs TConfig	0.02156		Cannot reject H_o
HHH vs IPOG	0.02156		Cannot reject H_o
HHH vs PSTG	0.0328		Cannot reject H_o
HHH vs HSS	0.09873		Cannot reject H_o
HHH vs CS	0.1587		Cannot reject H_o

Table 16. Friedman Test for Table 6

Friedman Test	Conclusion
Degree of freedom = 7, α = 0.05, Critical value = 14.0671 Friedman statistic (χ^2) = 28.388	28.388 > critical value, reject H_o and proceed to *post-hoc* test

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of ITCH, Jenny, TConfig, TVG and CTE-XL are ignored

Table 17. *Post-hoc* Wilcoxon Rank-Sum Tests for Table 6

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm}	Conclusion
HHH vs HSS	0.02156	0.007142	p-value > α_{Holm}, Cannot reject H_o
HHH vs PSTG	0.02156		Cannot reject H_o
HHH vs CS	0.02156		Cannot reject H_o
HHH vs IPOG	0.02156		Cannot reject H_o
HSS vs PICT	0.02156		Cannot reject H_o
HSS vs GTWay	0.2501		Cannot reject H_o
HSS vs MIPOG	0.6571		Cannot reject H_o

Table 18. Friedman Test for Table 7

Friedman Test	Conclusion
Degree of freedom = 10, α = 0.05, Critical value = 18.3070 Friedman statistic (χ^2) = 56.022	56.022 > critical value, reject H_o and proceed to *post-hoc* test

*Owing to incomplete sample (i.e. with one or more NA entries), the contribution of TConfig is ignored

Table 19. *Post-hoc* Wilcoxon Rank-Sum Tests for Table 7

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm}	Conclusion
HHH vs HSS	0.005859	0.005	p-value (3 d.p) ≈ α_{Holm}, Reject H_o if confident level is set at 90%
HHH vs PSTG	0.005859	0.005	p-value (3 d.p) ≈ α_{Holm}, Reject H_o if confident level is set at 90%
HHH vs CS	0.005859	0.005	p-value (3 d.p) ≈ α_{Holm}, Reject H_o if confident level is set at 90%
HHH vs IPOG	0.005859	0.005	p-value (3 d.p) ≈ α_{Holm}, Reject H_o if confident level is set at 90%
HSS vs Jenny	0.005859	0.005	p-value (3 d.p) ≈ α_{Holm}, Reject H_o if confident level is set at 90%
HSS vs PICT	0.005859	0.005	p-value (3 d.p) ≈ α_{Holm}, Reject H_o if confident level is set at 90%
HSS vs TVG	0.008645	0.0125	p-value < α_{Holm}, Reject H_o if confident level is set at 90%
HHH vs HSS	0.01253	0.0166	p-value < α_{Holm}, Reject H_o if confident level is set at 90%
HHH vs ITCH	0.2877	0.025	p-value > α_{Holm}, Cannot reject H_o
HHH vs MIPOG	0.8962		Cannot reject H_o

Table 20. Friedman Test for Table 8

Friedman Test	Conclusion
Degree of freedom = 6, α = 0.05, Critical value = 12.5916 Friedman statistic (χ^2) = 25.500	25.500 > critical value, reject H_o and proceed to *post-hoc* test

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of PSTG, CS, ITCH, PICT, and TConfig are ignored

Table 21. *Post-hoc* Wilcoxon Rank-Sum Tests for Table 8

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm}	Conclusion
HHH vs IPOG	0.01385	0.008333	p-value > α_{Holm}, Cannot reject H_o
HHH vs Jenny	0.01385		Cannot reject H_o
HHH vs TVG	0.01385		Cannot reject H_o
HHH vs GTWay	0.07056		Cannot reject H_o
HHH vs HSS	0.2315		Cannot reject H_o
HHH vs MIPOG	0.4583		Cannot reject H_o

Table 22. Friedman Test for Table 9

Friedman Test	Conclusion
Degree of freedom = 7, α = 0.05, Critical value = 14.0671 Friedman statistic (χ^2) = 96.883	96.883 > critical value, reject H_o and proceed to *post-hoc* test

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of TConfig, ITCH, CTE-XL are ignored

Table 23. *Post-hoc* Wilcoxon Rank-Sum Tests for Table 9

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm}	Conclusion
HHH vs Jenny	0.0001235	0.007142	p-value < α_{Holm}, Reject H_o
HHH vs PICT	0.0001266	0.008333	p-value < α_{Holm}, Reject H_o
HHH vs TVG	0.0001462	0.01	p-value < α_{Holm}, Reject H_o
HHH vs IPOG	0.0008479	0.0125	p-value < α_{Holm}, Reject H_o
HHH vs PSTG	0.001536	0.01666	p-value < α_{Holm}, Reject H_o
HHH vs HSS	0.0024	0.025	p-value < α_{Holm}, Reject H_o
HHH vs CS	0.02462	0.05	p-value < α_{Holm}, Reject H_o

\

Table 24. Friedman Test for Table 10

Friedman Test	Conclusion
Degree of freedom = 7, α = 0.05, Critical value = 14.0671 Friedman statistic ($\chi 2$) = 216.963	216.963 > critical value, reject H_o and proceed to *post-hoc* test

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of TConfig, ITCH, and CTE-XL are ignored

Table 25. *Post-hoc* Wilcoxon Rank-Sum Tests for Table 10

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm}	Conclusion
HHH vs PICT	1.762e-08	0.007142	p-value < α_{Holm}, Reject H_o
HHH vs IPOG	1.766e-08	0.008333	p-value < α_{Holm}, Reject H_o
HHH vs Jenny	3.786e-08	0.01	p-value < α_{Holm}, Reject H_o
HHH vs TVG	9.877e-08	0.0125	p-value < α_{Holm}, Reject H_o
HHH vs HSS	1.698e-07	0.01666	p-value < α_{Holm}, Reject H_o
HHH vs PSTG	2.527e-07	0.025	p-value < α_{Holm}, Reject H_o
HHH vs CS	6.752e-07	0.05	p-value < α_{Holm}, Reject H_o

Table 26. Friedman Test for Table 11

Friedman Test	Conclusion
Degree of freedom = 5, α = 0.05, Critical value = 11.0705 Friedman statistic (χ^2) = 23.779	23.779 > critical value, reject H_o and proceed to *post-hoc* test

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of CS, TConfig, ITCH, TVG, CTE-XL and IPOG-D are ignored

Table 27. *Post-hoc* Wilcoxon Rank-Sum Tests for Table 11

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm}	Conclusion
HHH vs HSS	0.002474	0.01	p-value > α_{Holm}, Cannot reject H_o
HHH vs Jenny	0.03042		Cannot reject H_o
HHH vs PSTG	0.03149		Cannot reject H_o
HHH vs PICT	0.090750		Cannot reject H_o
HHH vs IPOG	0.9226		Cannot reject H_o

7. Discussion

A number of observations can be summarized based on the obtained results from Table 4 through 11 as well as their corresponding statistical analyses from Tables 12 through 27.

Referring to Table 4, SA appears to outperform most other strategies for low interaction (i.e. t<3). Specifically, SA gives the best overall results for CA(N; 3, 6^6), CA(N; 3, 5^7), and MCA(N; 2, 6^1 5^1 4^6 3^8 2^3). However, no results for SA are available for t>3. HHH generates the best overall results for MCA (N; 4, $3^4 4^5$), MCA (N; 4, 5^1 3^8 2^2), MCA (N; 4, 8^2 7^2 6^2 5^2), MCA (N; 4, 6^5 5^4 3^2) and MCA (N; 4, 10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1). Putting SA and HHH aside, GA and ACO obtain similar performance. PSTG appears to perform the poorest as far as the meta-heuristic strategies are concerned. Overall, the meta-heuristic based strategies outperform the general computational based counterparts. Concerning the statistical analysis of Table 4 (as highlighted by the Friedman test analysis in Tables 12 and the post-hoc test in 13), we observe that HHH performance is statistically superior to all other strategies (ignoring the results of CS, SA, GA, ACO, mAETG and AETG owing to incomplete samples). It must be stressed that SA could have been the best strategy if the samples are complete.

In Table 5, HHH outperforms all other strategies in terms of test size with CS being the runner up. HHH obtains the best overall results for CA (N; 6, 2^{10}). ITCH gives the best overall result for CA (N; 2, 2^{10}). Despite the ITCH result, meta-heuristic strategies outperform the general computational counterparts (similar to the previous results). The Friedman test analysis of Table 5 (as summarized in Table 14) indicates that there are significant differences among the pairwise performance of all the strategies. However, *post-hoc* analysis of Table 5 (as seen in Table 15) demonstrates that the performance of HHH is not significantly better as compared to the given strategies (ignoring the contribution of ITCH).

Concerning Table 6, there are little differences as far as comparative performances between strategies are concerned. HHH generates only one best overall result for CA (N; *4*, 5^{10}). In similar manner, ITCH, MIPOG and GTWay obtain the best overall result for CA(N; 3, 5^{10}), CA(N; 5, 5^{10}), and CA(N; 6, 5^{10}) respectively. As expected, although the Friedman test analysis indicates that there are differences among the pairwise performance of all the strategies (as depicted in Table 16), the *post-hoc* analysis of Table 6 (as seen in Table 17) demonstrates that there is no significant difference in the performance of HHH with other strategies (ignoring the contributions of ITCH, Jenny, TConfig, TVG and CTE-XL).

As for Table 7, MIPOG dominates the overall best results. In fact, MIPOG establishes the best overall results for CA(N; 4, 5^7) and CA(N; 4, 5^{11}). ITCH is the runner up giving the best overall result for CA(N; 4, 5^{12}). HHH generates the best overall results for two configurations involving CA (N; 4, 5^8) and CA (N; 4, 5^9). HSS manages to generate the best overall result for CA(N; 4, 5^{10}). The null hypothesis for the Friedman test analysis for Table 7 (as seen in Table 18) is rejected indicating that there are significant differences in term of the comparative performances of between each individual strategy. The *post-hoc* analysis, however, is not in favor of the alternate hypothesis with 95% confidence level. Nevertheless, if the confidence level is reduced to 90%, the null hypothesis can now be revisited (refer to Table 19). Here, the HHH performance is statistically superior as compared to all other strategies with the exception of MIPOG and ITCH (ignoring the contribution of TConfig).

As far as Table 8 is concerned, HHH gives the best overall performance with two best overall results for configurations involving CA (N; 4, 3^{10}), and CA (N; 4, 6^{10}) respectively. Unlike HHH, most other strategies manage to obtain the best result only for one configuration. Specifically, HSS offers the best overall results for CA(N; 4, 5^{10}) with MIPOG for CA(N; 4, 7^{10}). GTWay gives the best overall result for CA (N; 4, 4^{10}). Statistical analysis (involving the Friedman test analysis in Table 20) and *post-hoc* analysis in Table 21) confirm our observation. HHH performance is statistically superior to all other strategies (not considering the contribution of PSTG, CS, ITCH, PICT, and TConfig).

Concerning Table 9, HHH outperforms most other given strategies. HHH offers the best overall results for CA (N; 2, 4^7), CA (N; 3, 5^7), CA (N; 4, 3^7), CA (N; 4, 4^7), CA (N; 4, 5^7), CA (N; 5, 3^7), CA (N; 5, 5^7), CA (N; 6, 3^7), and CA (N; 6, 5^7). Not considering HHH, the best overall results for other configurations are partially shared by many strategies. CS offers the best overall result for CA(N; 4, 2^7). TConfig gives the best overall result for CA(N; 5, 4^7). ITCH offers the best overall results for CA(N; 3, 3^7). Based on our statistical analysis (involving the Friedman test analysis in Table 22) and *post-hoc* analysis (in Table 23), HHH performance is statistically superior as compared to other strategies (not considering the contributions of TConfig, ITCH, CTE-XL).

Referring to Table 10, HHH obtains the best overall results for almost half of the table including CA (N; 3, 3^6), CA (N; 3, 3^9), CA (N; 3, 3^{10}), CA (N; 3, 3^{11}), CA (N; 3, 3^{12}), CA (N; 4, 3^5), CA (N; 4, 3^6), CA (N; 4, 3^7), CA (N; 4, 3^8), CA (N; 4, 3^9), CA (N; 4, 3^{10}), CA (N; 4, 3^{11}), CA (N; 4, 3^{12}), CA (N; 5, 3^6), CA (N; 5, 3^7), CA (N; 5, 3^8), CA (N; 5, 3^9), CA (N; 5, 3^{10}), CA (N; 5, 3^{11}), CA (N; 5, 3^{12}), CA (N; 6, 3^7), CA (N; 6, 3^8), CA (N; 6, 3^9), CA (N; 6, 3^{10}), CA (N; 6, 3^{11}), and CA (N; 6, 3^{12}) respectively. ITCH is the runner up offering the best overall results in five configurations involving CA(N; 2, 3^{10}), CA(N; 2, 3^{11}), CA(N; 2, 3^{12}), CA(N; 3, 3^7), and CA(N; 3, 3^8). Meanwhile, CS offers the best overall results for CA(N; 3, 3^6). The Friedman test analysis of Table 10 (in Table 24) is in favor of the alternate hypothesis. The *post-hoc* analysis (in Table 25) is also in favor of the alternate hypothesis. We conclude that HHH performance is statistically superior as compared all other strategies (not considering the contributions of TConfig, ITCH, CTE-XL).

Finally, the best overall results for Table 11 are evenly distributed across many strategies. IPOG gives the best performance with three new best overall results for CA(4,2^7 3^2 4^1 10^2), CA(2,2^{13} 4^5), CA(3,2^{13} 4^5). PICT establishes the best overall results for CA(5,2^7 3^2 4^1 10^2), CA(6,2^7 3^2 4^1 10^2), and CA(6,2^{13} 4^5). TVG offers the best overall results for CA(4,2^{13} 4^5) and CA(5,2^{13} 4^5). Although does not generate any new best overall results, HHH performance is sufficiently competitive. HHH manages to match seven existing best results involving CA(2,3^4), CA(3,3^4), CA(2,2^7 3^2 4^1 10^2), CA(3,2^7 3^2 4^1 10^2), CA(2,2^2 3^3), CA(3,2^2 3^3) and CA(4,2^2 3^3). Based on the statistical analysis (involving the Friedman test analysis in Table 26) and *post-hoc* analysis (in Table 27), we conclude that there is no significant difference as far as the HHH test size is concerned with each individual strategy (ignoring the contribution of CS, TConfig, ITCH, TVG, CTE-XL and IPOG-D).

8. Threats to Validity

Empirical and experimental studies often encounter many threats to validity. Substantial efforts have been undertaken to minimize such threats. In the context of our study, several threats could be leveled. Firstly, the fairness of our benchmark experiments involving meta-heuristic-based strategies can be an issue. Owing to unavailability of source codes, much comparison with the related work is solely based on the published results. Revisiting Table 3, given the population size and iteration value, the most minimum number of fitness function evaluation per iteration is the only value that can be ascertained (population size · max iteration). As the *t-way* test suite generation is not a single solution problem, there is a need for the outer loop to ensure convergence (i.e. producing a population of solution as the complete test suite covering all the required interactions). For this reason, the maximum number of fitness function evaluation (F_{max} = number of iteration for convergence · population size · max iteration) cannot be exactly determined as it depends on each algorithm's convergence process (i.e. until all interactions are covered). As such, there is no guarantee that the maximum number of fitness function evaluation (F_{max}) is the same for all strategies (even with the same number of fitness function evaluation per iteration). Only when F_{max} is empirically fixed (as the stopping criterion large enough to allow convergence for the largest configurations) for all strategies can the fair comparison be made.

Secondly, as far as the calibration of the specific parameters for each LLH is concerned, we relied on the reported best tuned values for our LLH (i.e. for PSO and Cuckoo) from the similar problems in the scientific literature. As meta-heuristics algorithm such as PSO is highly sensitive to parameter changes, the adopted best tuned values may not be applicable to our case. Thus, in order to ensure optimum performance, re-tuning of the related HHH parameters may be beneficial.

Thirdly, our statistical analysis has been based on the best reported values and not the mean values for all the configurations (as the mean values are not highlighted in most published results). The main issue here is that some of the best results may be obtained by chance (especially in the case of meta-heuristic-based strategies), hence, affecting our conclusion.

Finally, our last threat to validity also relates to statistical analysis. As highlighted in earlier sections, our statistical analysis (based on the Friedman test and *post-hoc* Wilcoxon-Rank Sum analysis) requires the complete samples for all the strategies. In the case of strategies with missing values (i.e. cells with NA entries), their contributions are completely ignored. For this reason, the complete statistical analysis involving all the strategies cannot be feasibly performed.

9. Concluding Remark

In this paper, we have described a novel approach of applying hyper-heuristic, called HHH, as a strategy for *t-way* test generation. Comparatively, the performances of HHH with other strategies have been promising. In the case of meta-heuristic-based strategies, we are unable to ensure the same number of objective function evaluations for all the experiments. To be specific, the common F_{max} cannot be introduced as most implementation source codes are not publically available for modification.

To the best of our knowledge, HHH is the first hyper-heuristic heuristic based strategy that addresses the problem of *t-way* test suite generation. The main feature of HHH is that it enhances the diversification and intensification of the searching process by adaptively selecting the LLHs based on their previous performances. Each LLH works in synergy with the HLH whereby highly performing LLH has more chance of being selected during the search process.

Within HHH, the three defined operators (i.e. improvement, intensification and diversification operators) serve as the virtual "switch" for HHH. Achieving balance diversification and intensification through diversity learning [73] within the Reinforced Learning Framework [18], the three HHH operators can be comparable to Choice Function (CF) accept all moves operator [74]. Similar to HHH, CF accepts the current move but the current LLH can only continue to the next iteration if the quality of the current solution improves. Specifically, CF maintains the performance score of the LLHs based on three criteria: previous performance, pair-independence between LLHs, and time performance. As the name suggests, the previous performance consideration is the same for both CF and HHH but not the pair-independence and time performance. Although pair-independence and individual time performance criteria are useful for adaptive meta-heuristic switching, their contribution within the context of (domain specific) *t-way* test generation might not be significant as compared to intensification and diversification measure provided by the HHH operators. Recall that the HHH diversification operator measures how diverse the current and the previously generated test case are, whilst the intensification HHH operator evaluates how close the current and the previously generated test case are against the final test suite list.

As our findings have been encouraging, we are planning to improve our work further. Comparative analysis can also be made between HHH operators with the potentially different heuristic selection and acceptance operators (e.g. the Exponential Monte Carlo with counter [75]) using the same LLHs as reference within the context of *t*-way test suite generation. In this manner, the performance of different selection and acceptance operators can be objectively evaluated. Perhaps, the scope of the test generation can also be broadened to address variable strength interaction as well as constraints support to cater for the testing of software product lines.

Additionally, instead of relying on a full-blown algorithm as the LLH candidate, HHH can also support singling out the operator from the meta-heuristic of interest. For instance, the Teacher Phase search in TLBO can independently serve as LLH whilst the Student Phase search can be used as another LLH. As a result, tuning of each individual (full-blown) LLH would not be issues that have to be addressed.

To recap, as demonstrated by the implementation of HHH, a hyper-heuristic is able to generate *good enough, soon enough and cheap enough* solutions [76] so as to improve generality, raise domain independence, as well as enhance the level of flexibility [18].

Acknowledgement

The work reported in this paper is partially funded by the Science Fund Grant from the Ministry of Science, Technology, and Innovation (MOSTI), Malaysia and the the Long-Term National Plan for Science, Technology and Innovation (LT-NPSTI) Grant, the King Abdul-Aziz City for Science and Technology (KACST), Kingdom of Saudi Arabia.

References

[1] Y. Lei and K. C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing," in *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium (HASE '98)*, Washington, DC, USA, 1998, pp. 254–261.

[2] M. Harman, Y. Jia, J. Krinke, B. Langdon, J. Petke, and Y. Zhang, "Search based Software Engineering for Software Product Line Engineering: A Survey and Directions for Future Work," in *Proceeding of the 18th International Software Product Line Conference* 2014, pp. 5-18.

[3] M. Harman, A. Mansouri, and Y. Zhang, "Search Based Software Engineering: Trends, Techniques and Applications " *ACM Computing Surveys.* , vol. 45, p. 11, 2012.

[4] M. Harman and B. F. Jones, "Search-Based Software Engineering," *Information and Software Technology,* vol. 43, pp. 833-839, 2001.

[5] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," in *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, Hong Kong, 2004, pp. 72-77.

[6] B. S. Ahmed and K. Z. Zamli, "A Variable-Strength Interaction Test Suites Generation Strategy Using Particle Swarm Optimization," *Journal of Systems and Software,* vol. 84, pp. 2171-2185, 2011.

[7] B.S. Ahmed, K.Z. Zamli, and C. P. Lim, "Application of Particle Swarm Optimization to Uniform and Variable Strength Covering Array Construction," *Applied Soft Computing,* vol. 12, pp. 1330-1347, 2012.

[8] A. R. A. Alsewari and K. Z. Zamli, "Design and Implementation of a Harmony-Search-based Variable-Strength t-way Testing Strategy with Constraints Support," *Information and Software Technology,* vol. 54, pp. 553-568, 2012.

[9] J. Stardom, "Metheuristic and the Search for Covering and Packing Array," MSc, Department of Mathematics, Simon Fraser University, 2001.

[10] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, "Constructing Strength Three Covering Arrays with Augmented Annealing," *Discrete Mathematics,* vol. 308, pp. 2709-2722, 2008.

[11] 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, pp. 13-29, 2015.

[12] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation,* vol. 1, pp. 67-82, 1997.

[13] E.-G. Talbi, "A Unified Taxonomy of Hybrid Metaheuristics with Mathematical Programming, Constraint Programming and Machine Learning," in *Studies in Computational Intelligence*. vol. 434, E.-G. Talbi, Ed., ed: Springer, 2013, pp. 3-76.

[14] R. V. Rao, V. J. Savsani, and D. P. Vakharia, "Teaching-Learning-based Optimization: A Novel Method for Constrained Mechanical Design Optimization Problems," *Computer Aided Design,* vol. 43, pp. 303-313, 2011.

[15] A. Alazzam and H. W. Lewis III, "A New Optimization Algorithm For Combinatorial Problems," *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, pp. 63-68, 2013.

[16] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of the IEEE International Conference Neural Networks*, 1995, pp. 1942-1948.

[17] X.-S. Yang and S. Deb, "Cuckoo Search via Levy Flight," in *Proceedings of World Congress on Nature and Biologically Inspired Computing*, 2009, pp. 210-214.

[18] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, *et al.*, "Hyper Heuristics: A Survey of the State of the Art," *Journal of the Operational Research Society*, vol. 64, pp. 1695–1724 2013.

[19] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. W. . "A Classification of Hyper-heuristic Approaches," in *In Handbook of Meta-Heuristics*, M. Gendreau and J.-Y. Potvin, Eds., ed: Kluwer, 2010, pp. 449-468.

[20] E. K. Burke, G. Kendall, and E. Soubeiga, "A Tabu-Search Hyperheuristic for Timetabling and Rostering," *Journal of Heuristics,* vol. 9, pp. 451-470, 2003.

[21] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A Dynamic Multiarmed Bandit-Gene Expression Programming Hyper-Heuristic for Combinatorial Optimization Problems," *IEEE Transactions on Cybernetics,* vol. 45, pp. 217-228, 2015.

[22] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "The Automatic Design of Hyper-Heuristic Framework with Gene Expression Programming for Combinatorial Optimization Problems," *IEEE Transactions on Evolutionary Computation,* vol. 309-325, p. 3, 2015.

[23] B. Stevens and E. Mendelsohn, "Efficient Software Testing Protocols," in *Proceedings of the 8th IBM Centre for Advanced Studies Conference (CASCON '98),* Toronto, Ontario, Canada, 1998, pp. 279-293.

[24] C. Yilmaz, M. B. Cohen, and A. Porter, "Covering arrays for efficient fault characterization in complex configuration spaces," *ACM SIGSOFT Software Engineering Notes,* vol. 29, pp. 45-54, 2004.

[25] C. J. Colbourn, "Strength two covering arrays: existence tables and projection," *Discrete Mathematics,* vol. 308, pp. 772-786, 2008.

[26] M. Chateauneuf and D. L. Kreher, "On the state of strength-three covering arrays," *Journal of Combinatorial Designs,* vol. 10, pp. 217-238, 2002.

[27] M. B. Cohen, "Designing Test Suites for Software Interaction Testing," Doctor of Philosophy PhD Thesis, Department of Computer Science, University of Auckland, New Zealand, 2004.

[28] M. B. Cohen, C. J. Colbourn, P. B. Gibbons, and W. B. Mugridge, "Constructing Test Suites for Interaction Testing," in *Proceedings of the 25th IEEE International Conference on Software Engineering*, Portland, Oregon, 2003, pp. 38-48.

[29] A. Hartman and L. Raskin, "Problems and Algorithms for Covering Arrays," *Discrete Mathematics,* vol. 284, pp. 149-156, 2004.

[30] K. A. Bush, "Orthogonal Arrays of Index Unity," *Annals of Mathematical Statistics,* vol. 23, pp. 426-434, 1952.

[31] R. Mandl, "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing," *Communications of the ACM,* vol. 28, pp. 1054-1058, 1985.

[32] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software Fault Interactions and Implications for Software Testing," *IEEE Transactions on Software Engineering,* vol. 30, pp. 418-421, 2004.

[33] 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, pp. 437-444, July 1997.

[34] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction Testing of Highly-Configurable Systems in the Presence of Constraints," in *Proceeding of International Symposium on Software Testing and Analysis*, London, UK, 2007, pp. 129 -139

[35] M. Klaib, K. Zamli, N. Isa, M. Younis, and R. Abdullah, "G2Way A Backtracking Strategy for Pairwise Test Data Generation," in *Proceeding of the 15th Asia-Pacific Software Engineering Conference APSEC '08'* 2008, pp. 463-470.

[36] K. Z. Zamli, M. F. J. Klaib, M. I. Younis, N. A. M. Isa, and R. Abdullah, "Design and Implementation of A T-Way Test Data Generation Strategy with Automated Execution Tool Support," *Information Sciences,* vol. 181, pp. 1741-1758, 2011.

[37] J. Arshem. (2009, 20 July). *TVG*. Available: http://sourceforge.net/projects/tvg

[38] B. Jenkins. *Jenny Test Tool, available from http://www.burtleburtle.net./bob/math/jenny.html, last accessed on April, 2010.*

[39] A. Hartman, *IBM Intelligent Test Case Handler*: http://www.alphaworks.ibm.com/tech/whitch, 2010.

[40] Keith and H. Doug. (2006, 16 June). *PICT.* Available: http://testmuse.wordpress.com/2006/04/05/pict-tool-available/2006

[41] J. Czerwonka, "Pairwise Testing in the Real World: Practical Extensions to Test-Case Scenarios," in *Proceedings of the 24th Pacific Northwest Software Quality Conference*, 2006, pp. 419-430.

[42] E. Lehmann and J. Wegener, "Test Case Design by Means of the CTE XL," in *Proceeding of the 8th European International Conference on Software Testing, Analysis & Review*, 2000, pp. 1-10.

[43] Y. T. Yu, S. P. Ng, and E. Y. K. Chan, "Generating, Selecting and Prioritizing Test Cases From Specifications with Tool Support," in *Proceedings of the 3rd International Conference on Quality Software*, 2003, pp. 83-90.

[44] Y. Lei and K. C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing," in *Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering*, Washington, DC, USA, 1998, pp. 254 - 261.

[45] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A General Strategy for T-Way Software Testing," in *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Tucson, AZ U.S.A, 2007, pp. 549-556.

[46] 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 Verification and Reliability,* vol. 18, pp. 125-148, 2008.

[47] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the In-Parameter-Order Strategy for Constructing Covering Arrays," *Journal of Research of the National Institute of Standards and Technology,* vol. 113, pp. 287-297., October 2008 2008.

[48] M. I. Younis and K. Z. Zamli, "MC-MIPOG: A Parallel T-Way Test Generation Strategy for Multicore Systems," *ETRI Journal,* vol. 32, pp. 73-83, 2010.

[49] M. I. Younis and K. Z. Zamli, "MIPOG-An Efficient T-Way Minimization Strategy for Combinatorial Testing," *International Journal of Computer Theory and Engineering,* vol. 3, pp. 388-397, 2011.

[50] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "MIPOG-Modification of the IPOG Strategy for T-Way Software Testing," in *Proceeding of the Distributed Frameworks and Applications*, 2008, pp. 1-6.

[51] M. B. Cohen, C. J. Colbourn, and A.C.H.Ling, "Augmenting Simulated Annealing to build Interaction Test Suite," in *Proceedings of the14th International Symposium on Software Reliability Engineering*, 2003, pp. 394-405.

[52] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "Evaluating Improvements to a Meta-Heuristic Search for Constrained Interaction Testing," *Empirical Software Engineering,* pp. 61-102, 2011.

[53] B. S. Ahmed and K. Z. Zamli, "PSTG: A T-Way Strategy Adopting Particle Swarm Optimization," in *Proceedings of the 4th Asia International Conference on Mathematical /Analytical Modelling and Computer Simulation*, 2010, pp. 1-5.

[54] B. S. Ahmed and K. Z. Zamli, "T-Way Test Data Generation Strategy Based on Particle Swarm Optimization," in *Proceedings of the 2nd International Conference on Computer Research and Development*, 2010, pp. 93-97.

[55] B. S. Ahmed and K. Z. Zamli, "The Development of a Particle Swarm Based Optimization Strategy for Pairwise Testing," *Journal of Artificial Intelligence,* vol. 4, pp. 156-165, 2011.

[56] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Constructing a T-Way Interaction Test Suite Using the Particle Swarm Optimization Approach," *International Journal of Innovative Computing, Information and Control,* vol. 8, pp. 431-452, 2012.

[57] M. Črepinšek, S.-H. Liu, and L. Mernik, "A Note on Teaching-Learning-Based Optimization Algorithm," *Information Sciences,* vol. 212, pp. 79-93, 2012.

[58] *TLBO Implementation -Yarpiz.com, available from http://yarpiz.com/83/ypea111-teaching-learning-based-optimization ,last accessed on January, 2016.*

[59] X.-S. Yang, *Nature-Inspired Metaheursitic Algorithm*: Luniver Press, 2010.

[60] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C- The Art of Scientific Computing*: Cambridge University Press, 1992.

[61] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithm*: Luniver Press, 2010.

[62] B. Xing and W.-J. Gao, *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms*: Springer, 2014.

[63] M. B. Cohen, "Designing Test Suites for Software Interaction Testing," PhD Thesis, Department of Computer Science, University of Auckland, 2005.

[64] B. Garvin, M. Coehn, and M. Dwyer, "Evaluating Improvements to a Meta-Heuristic Search for Constrained Interaction Testing," *Empirical Software Engineering,* vol. 16, pp. 61-102, 2011.

[65] C. B. Renee and C. J. Colbourn, "One-test-at-a-time Heuristic Search for Interaction Test Suites," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, ACM*, 2007, pp. 1082-1089.

[66] A. R. A. Alsewari, "Design and Implementation of a Harmony Search based t-way Testing Strategy with Constraints Support," PhD Thesis, School of Electrical and Electronics Engineering, Universiti Sains Malaysia, 2012.

[67] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Application of Particle Swarm Optimization to Uniform and Variable Strength Covering Array Construction," *Applied Soft Computing,* vol. 12, pp. pp. 1330-1347, 2012.

[68] B. S. Ahmed, "Adopting a Particle Swarm based Test Generator Strategy for Variable Strength and t-way Testing," PhD Thesis, School of Electrical and Electronics Engineering, Universiti Sains Malaysia, 2012.

[69] M. Črepinšek, S.-H. Liu, L. Mernik, and M. Mernik, "Is a Comparison of Results Meaningful from the Inexact Replications of Computational Experiments?," *Soft Computing,* vol. 20, pp. 223-235, 2015.

[70] M. Črepinšek, S.-H. Liu, and M. Mernik, "Replication and Comparison of Computational Experiments in Applied Evolutionary Computing: Common Pitfals and Guidelines to avoid them," *Applied Soft Computing,* vol. 19, pp. 161-170, 2014.

[71] M. Mernik, S.-H. Liu, D. Karaboga, and M. Črepinšek, "On Clarifying Misconceptions when Comparing Variants of the Artificial Bee Colony Algorithm by Offering a New Implementation," *Information Sciences,* vol. 291, pp. 115-127, 2015.

[72] A. Draa, "On the Performances of the Flower Pollination Algorithm- Qualitative and Quantitative Analyses," *Applied Soft Computing,* vol. 34, pp. 349-371, 2015.

[73] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and Exploitation in Evolutionary Algorithms: A Survey," *ACM Computing Surveys,* vol. 45, 2013.

[74] P. Cowling and G. Kendall, "A Hyper Heuristic Approach to Scheduling a Sales Summit," in *Proceedings of 3rd International Conference on Practice and Theory of Automated Time Tabling*, 2001, pp. 176-190.

[75] M. Ayob and G. Kendall, " A Monte Carlo Hyper-Heuristic to Optimise Component Placement Sequencing For Multi Head Placement Machine," in *Proceedings of the International Conference on Intelligent Technologies*, 2003, pp. 132-141.

[76] E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-Heuristics: An Emerging Direction in Modern Search Technology," in *Handbook of Meta-Heuristics -International Series in Operations Research and Management Science*. vol. 57, F. Glover and G. Kochenberger, Eds., ed: Kluwer, 2003, pp. 457-474.