An Experimental Study of Hyper-Heuristic Selection and Acceptance Mechanism for Combinatorial *t*-way Test Suite Generation

KAMAL Z. ZAMLI and FAKHRUD DIN
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

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

BESTOUN S. AHMED
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University Karlovo n'am 13, 121 35 Praha 2, Czech Republic
Email: bestoon82@gmail.com

Abstract

Recently, many meta-heuristic algorithms have been proposed to serve as the basis of a *t*-way test generation strategy (where *t* indicates the interaction strength) including Genetic Algorithms (GA), Ant Colony Optimization (ACO), Simulated Annealing (SA), Cuckoo Search (CS), Particle Swarm Optimization (PSO), and Harmony Search (HS). Although useful, meta-heuristic algorithms that make up these strategies often require specific domain knowledge in order to allow effective tuning before good quality solutions can be obtained. Hyper-heuristics provide an alternative methodology to meta-heuristics which permit adaptive selection and/or generation of meta-heuristics automatically during the search process. This paper describes our experience with four hyper-heuristic selection and acceptance mechanisms namely Exponential Monte Carlo with counter (EMCQ), Choice Function (CF), Improvement Selection Rules (ISR), and newly developed Fuzzy Inference Selection (FIS), using the *t*-way test generation problem as a case study. Based on the experimental results, we offer insights on why each strategy differs in terms of its performance.

***Keywords*:** Software Testing; *t*-way Testing; Hyper-Heuristics; Meta-Heuristics; Fuzzy Inference Selection;

1. Introduction

Testing is an important process in software development to help identify areas where the software is not performing as expected. This process is often expensive owing to the time taken to execute the set of test cases. It has been a research focus to find suitable sampling strategies to generate a small yet efficient set of test cases for testing a software system.

Over the years, a plethora of sampling strategies has been proposed in the literature (including that of boundary value analysis, equivalence partitioning, and decision tables; to name just a few). Although useful for some classes of software testing problems, these sampling strategies have not been designed to effectively deal with faults due to interaction. For this reason, many (sampling) *t*-way strategies (where *t* indicates the interaction strength) have been proposed in the scientific literature. Some early algebraic based *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. The emergence of computational based *t*-way strategies ease these restrictions allowing for the support for arbitrary configuration at the expense of producing high quality solutions rather than guaranteed optimal solutions.

Tackling this issue, and formulating interaction testing as an optimization problem, recent efforts have focused on the adoption of meta-heuristic algorithms as the basis for *t*-way testing strategy. Search Based Software Engineering (SBSE) [24], has developed many meta-heuristic based *t*-way strategies (e.g. based on Genetic Algorithms (GA) [14, 33], Particle Swarm Optimization (PSO) [4, 32, 45], Harmony Search (HS) [6], Ant Colony Optimization Algorithm (ACO) [14, 41], Simulated Annealing [17] and Cuckoo Search (CS) [2]), which have been reported in the scientific literature.

Meta-heuristic based strategies are known to produce a good quality *t*-way test suite. However, as suggested by the *No Free Lunch theorem* [44], the search for a single meta-heuristic that can outperform others in all optimization problem instances is fruitless. Hybridization of more than one meta-heuristic can be useful in enhancing the performance of *t-way* strategies, as hybridization can capitalize on the strengths and compensate the deficiencies of each individual algorithm.

Hybridization could be in the form of the integration of two or more search operators from different meta-heuristics, partly or in full, creating a new algorithm. Hybridization could also be an ensemble of two or more heuristics and running them sequentially or in parallel. Hyper-heuristics can also be viewed as a form of hybridization. Unlike the hybridization (including ensembles) of meta-heuristics, hyper-heuristics permit the integration of two or more meta-heuristic search operators from different meta-heuristics through one defined parent heuristic via non-domain feedback (i.e. *(meta)-heuristic to choose (meta)-heuristics* [10]). With a hyper-heuristic, the selection of a particular search operator to be used at any particular instance can be adaptively (and dynamically) decided based on the feedback from its previous performance.

In this paper, we explore the hybridization of meta-heuristics utilizing a hyper-heuristic approach. We present a new *t-way* testing strategy. In the context of our study, this paper focuses on an experimental study of hyper-heuristic selection and acceptance mechanism for adaptively selecting low-level search operators. Although there has been existing work (e.g. timetabling problems), this methodology has not been considered for *t*-way test generation as a case study. This paper describes our comparative studies with four hyper-heuristic selection and acceptance mechanisms namely Exponential Monte Carlo with counter (EMCQ) [9], Choice Function (CF) [20, 28], Improvement Selection Rules (ISR) [49], and the newly developed Fuzzy Inference Selection (FIS). These mechanisms utilize four common search operators comprising a Genetic Algorithm (GA) crossover search operator [25], Teaching Learning based Optimization (TLBO) algorithm's peer learning search operator [39], Flower Algorithm's global Pollination (FPA) search operator [48] and Jaya algorithm's search operator [38].

The contributions of this paper can be summarized as follows:

- A new experimental study of existing hyper-heuristic selection and acceptance mechanisms, using *t*-way test generation as a case study. The study also benchmarks the results against existing meta-heuristic based strategies. Based on the results, we provide guidelines for choosing the appropriate mechanism and some insights on why each strategy differs in terms of performance.
- A new hyper-heuristic selection and acceptance mechanism based on Fuzzy Inference Selection (FIS).

The paper is organized as follows. Section 2 presents the theoretical framework covering the *t*-way test generation problem, its mathematical notation, related work as well as the main components of the hyper-heuristic. Section 3 describes the hyper-heuristic selection and acceptance mechanisms along with a description of each search operator. Section 4 presents our benchmarking experiments. Section 5 discusses our experimental observations. Finally, section 6 gives our concluding remarks along with the scope for future work.

2. Theoretical Framework

2.1. The *t*-way Test Generation Problem

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

$$f(Z) = |\{I \ in \ VIL\colon Z \ covers \ I\}| \qquad (1)$$
$$Subject \ to \ Z = Z_1, \ Z_2, \dots , \ Z_i \ in \ P_1, P_2, \dots \dots P_i; \ i = 1, 2, \dots , \ N$$

where, *f*(*Z*) is an objective function (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.

A simple configurable software system is used as a model to illustrate the *t*-way test generation problem. Figure 1 represents the topology of a modern e-commerce software system based on the Internet [3]. The system may use different components or parameters. In this example, the system comprises five parameters. The client side has two parameters or two types of clients: those who use smart phones and those who use normal computers. There are different configurations in both cases. On the other side are different servers and databases.

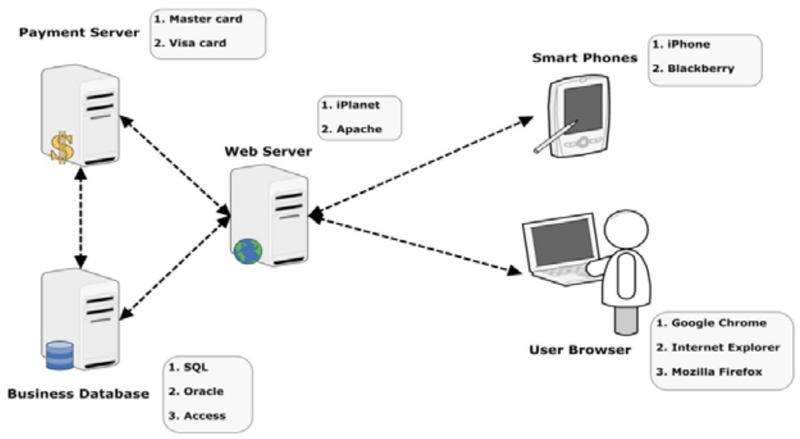


Figure 1. An E-commerce Software System [3]

The term "value" (i.e. *v*) is used to describe the configuration of each component. Thus, the system in Figure 1 can be summarized as a five-parameter system with a combination of three parameters with two values, and two parameters with three values, as in Table 1.

Table 1. An E- Commerce System Components and Configurations

	Components or Parameters				
	Payment Server	Smart Phone	Web Server	User Browser	Business Database
Configurations or Values	Master Card	iPhone	iPlanet	Chrome	SQL
	Visa Card	Blackberry	Apache	Explorer	Oracle
				Firefox	Access

To reduce the risk and ensure the quality of such software, manufacturers may need to test all combinations of interactions (i.e. exhaustive testing), which requires 72 test cases (i.e. 2×2×2×3×3). However, testing of all combinations is practically impossible given large configurations or large components. Considering the pairwise (*2-way*) test generation for the E-commerce yields only 9 test cases (see Table 2). It should be noted that all the *2-way* interaction tuples between parameters are covered at-least once.

Table 2. Pairwise Test Suite for E-Commerce System

Test No.	Payment Server	Smart Phone	Web Server	User Browser	Business Database
1	Master Card	Blackberry	iPlanet	Firefox	Oracle
2	Visa Card	iPhone	Apache	Firefox	SQL
3	Master Card	iPhone	iPlanet	Explorer	Access
4	Visa Card	Blackberry	Apache	Chrome	Access
5	Visa Card	iPhone	iPlanet	Chrome	Oracle
6	Master Card	Blackberry	Apache	Explorer	SQL
7	Master Card	iPhone	iPlanet	Chrome	SQL
8	Visa Card	iPhone	Apache	Explorer	Oracle
9	Visa Card	iPhone	iPlanet	Firefox	Access

2.2. 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 [42]. 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, 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.

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

To improve readability, it is customary to represent the covering array as CA ($N;t,k,v$) or simply CA($N;t,v^k$). Considering CA (9; 2, 3^4) as an example, the covering array represents the strength of 2 with 4 parameters and 3 values each. In the case when the number of component values varies, this can be handled by Mixed Covering Array (MCA), MCA($N;t,k,(v_1,v_2,\ldots v_k)$) [16]. Similar to the covering array, the notation can be represented by MCA ($N;t,k,v^k$). Using our earlier example of the E-commerce system in Figure 1, the test suite can be represented as MCA (9; 2, $2^3 3^2$).

2.3. Meta-Heuristic based *t*-way Strategies

The *t*-way test suite generation is an NP-hard problem [31] and significant research efforts have been carried out to investigate the problem. Computationally, the current approach can be categorized into one-parameter-at-a-time (OPAT) and one-test-at-a-time (OTAT) methods [35].

Derived from the in-parameter-order (IPO) strategy [31], the OPAT method begins with an initial array comprising of several selected parameters. The array is then horizontally extended until reaching all the selected parameters based on the required interaction coverage. This is followed by vertical extension, if necessary, to cover the remaining uncovered interactions. The iteration continues until all the interactions are covered.

Credited to the work of AETG [15], the OTAT method normally iterates over all the combinatorial interaction elements and generates a complete test case per iteration. While iterating, the strategy greedily checks whether the generated solution is the best fit value (i.e. covering the most uncovered interactions) from a list of potential solutions.

Adopting either the OPAT or the OTAT method, much effort has recently been focused on the use of meta-heuristic algorithms as part of the computational approach for *t*-way test suite generation. Termed Search based Software Engineering (SBSE), the adoption of meta-heuristic based strategies often produces more optimal test suite sizes although there may be tradeoffs in terms of computational costs.

Meta-heuristic based strategies can start with a population of random solutions. Then, one or more search operators are iteratively applied to the population in an effort to improve the overall fitness (i.e. in terms of greedily covering the interaction combinations). While there are many variations, the main difference between meta-heuristic strategies are the search operators. As far as the *t*-way test suite construction, meta-heuristics such as Genetic Algorithms (GA) [41], Ant Colony Optimization (ACO) [14], Simulated Annealing (SA) [18, 21], Particle Swarm Optimization (PSTG [4], DPSO [45], APSO [32]), Cuckoo Search (CS) [2] and Harmony Search Strategy (e.g. HSS) [6] have been reported in the scientific literature.

Each meta-heuristic algorithm has its own advantages and disadvantages. With hybridization, each algorithm can exploit the strengths and cover the weaknesses of the collaborating algorithms. Many recent results from the scientific literature (e.g. [22, 40]) seem to indicate that hybridization improves the performance of meta-heuristic algorithms.

We propose an elegant form of hybridization based on the use of hyper-heuristics. To be specific, our work investigates the use of common heuristic selection and acceptance mechanisms, based on Exponential Monte Carlo with counter (EMCQ), Choice Function (CF), Improvement Selection Rules (ISR) and the newly developed Fuzzy Inference Selection (FIS), for selection based hyper-heuristics as a strategy for *t*-way test suite constructions. Additionally, we also evaluate the effectiveness of our newly developed fuzzy inference based heuristic selection and acceptance mechanism.

2.4. Hyper-Heuristics and Related Work

Hyper-heuristics are alternative to meta-heuristics. Hyper-heuristics can be viewed as a high-level methodology which performs a search over the space formed by a set of low level heuristics which operate on the problem space. Unlike typical meta-heuristics, there is a logical separation between the problem domain and the high level hyper-heuristic. Apart from increasing the level of generality, hyper-heuristics can also be competitive with bespoke meta-heuristics.

Generally, hyper-heuristics can be classified as generative or selective [10]. Generative hyper-heuristics combine low-level heuristics to generate new higher level heuristics. Selective hyper-heuristics select from a set of low-level heuristics. Our work is based on selective hyper-heuristics. Selective hyper-heuristics can be online or offline. The former is unsupervised and learning happens dynamically during the search process, whilst the latter requires an additional training step prior to addressing the problem. For our work, we deal with online selective hyper-heuristics.

Owing to how the search process is undertaken, the aforementioned hyper-heuristic classification (i.e. selective or generative) can further be extended to either perturbative or constructive [10]. Perturbative heuristics (also known as improvement heuristics) manipulate complete candidate solutions by iteratively changing their component(s). In the case of selection methodologies, perturbative hyper-heuristics provide a combination of low-level meta-heuristic operators and/or simple heuristic searches with the aim of selecting and applying them for the improvement of the current solution. Some problems addressed with such hyper-heuristics are vehicle routing [40], project scheduling [8], timetabling [11], GA parameter tuning [23], and CAs construction [27, 49].

Constructive hyper-heuristics process partial candidate solutions by iteratively extending missing element(s) to build complete solutions. As a selection methodology, this approach combines several pre-existing low-level constructive meta-heuristic operators, selecting and using the (perceived) best heuristic for the current problem state. Combinatorial optimization problems such as production scheduling [13], cutting and packing [43], and timetabling [7] have been successfully addressed with this approach.

In the context of the current study, some previous hyper-heuristics research is particularly relevant. Choice Function (CF) and Exponential Monte Carlo with Counter (EMCQ) [9] are among the earliest hyper-heuristics reported in the scientific literature. CF exploits the reinforcement learning framework to penalize and reward (meta)-heuristics through a set of choice functions (f_1, f_2, f_3). The first parameter f_1 relates to the effectiveness of the currently employed heuristic. The second parameter f_2 evaluates the effectiveness of two heuristics when used consecutively. The third parameter f_3 increases the probability of a heuristic being selected, over time, to encourage exploration. EMCQ adopts a simulated annealing like probability density function that is a function of the number of iterations. A worsening fitness causes EMCQ to decrease its acceptance probability. Both CF and EMCQ are further discussed in the next section.

With regard to the use of a fuzzy inference system, as part of a hyper-heuristic, Asmuni et al. [7] developed a constructive hyper-heuristic for addressing the timetabling problem. In their work, the Mamdani type fuzzy system is responsible for scheduling courses based on the perceived difficulty. Different orderings are considered, for example, the event with the highest crisp value (most difficult) is scheduled first. Recently, Gudino-Penaloza et al. [23] developed a new hyper-heuristic using a Takagi-Sugeno based fuzzy inference system to adaptively adjust the control parameters of a GA. Although using fuzzy inference system, the works of both Asmuni et al. and Gudino-Penaloza et al. have a slightly different focus. Specifically, our work deals with heuristic selection and not event ordering or adaptive meta-heuristic parameter control adjustment.

As far as the *t*-way test suite generation problem is concerned, the work of Jia et al. [27] can be considered the pioneering effort to investigate the usefulness of hyper-heuristics for *t*-way test generation. Similar to EMCQ, the work adopts a simulated annealing based hyper-heuristic, called HHSA, to select from variants of six operators (i.e. single/multiple/smart mutation, simple/smart add and delete row). HHSA demonstrates good performance in terms of test suite size as well as displaying elements of learning in the selection of the search operators.

Complementing Jia et al., Zamli et al. [49] implemented improvement selection rules (ISR) utilizing a selection hyper-heuristic based on tabu search and three measures (quality, diversify and intensify) to assist the heuristic selection process. Although showing promising results, the ISR selection rules are too strict, supporting only Boolean outcomes. Furthermore, the original ISR also implemented full meta-heuristic algorithms (i.e. comprising of Teaching Learning based Optimization (TLBO) [39], Global Neighborhood Algorithm (GNA) [5], Particle Swarm Optimization (PSO)[29], and Cuckoo Search Algorithm (CS) [47]) as its search operators. As such, the original ISR implementation is computationally heavy. Addressing the limitation of ISR, the proposed FIS adopts fuzzy rules that are able to accommodate partial truth allowing smoother transition between the search operators. Additionally, it also incorporates lightweight search operators to minimize computational resources.

3. The Hyper-Heuristic Selection and Acceptance Mechanism

The selection and acceptance mechanism for selection based hyper-heuristics is shown in Figure 2. The hyper-heuristic selection and acceptance mechanism is represented by the dashed rectangle.

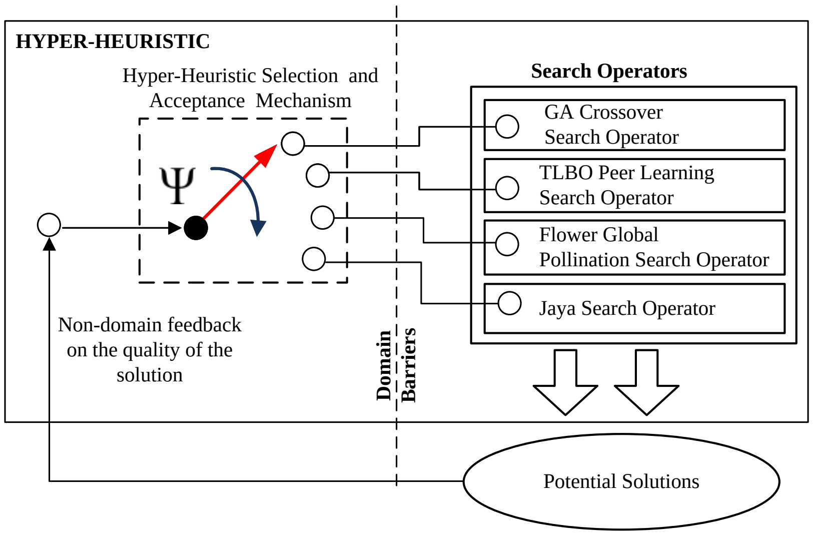

HYPER-HEURISTIC
Hyper-Heuristic Selection and Acceptance Mechanism
Ψ
Search Operators
GA Crossover Search Operator
TLBO Peer Learning Search Operator
Flower Global Pollination Search Operator
Jaya Search Operator
Non-domain feedback on the quality of the solution
Domain Barriers
Potential Solutions

Figure 2. The Hyper-Heuristic Selection and Acceptance Mechanism

We compare the performance of Exponential Monte Carlo with counter (EMCQ), Choice Function (CF), Improvement Selection Rules (ISR) and the newly developed Fuzzy Inference Selection (FIS) as the selection and acceptance mechanism. We use four common search operators comprising of a GA crossover operator, a TLBO peer learning search operator, an FPA global pollination search operator and Jaya algorithm's search operator.

The selection of the search operators needs to take into account the balance between diversification and intensification. As such, any arbitrary (but balanced) selection of the search operators is also possible. In our case, the FPA global pollination and the Jaya algorithm serve as the global search operators. The GA crossover and the TLBO peer learning serve as the local search operators.

3.1 Description of the Selection and Acceptance Mechanism

The next subsections detail the selection and acceptance mechanisms.

3.1.1 The Exponential Monte Carlo with Counter

The Exponential Monte Carlo with Counter (EMCQ) is a parameter free hyper-heuristic developed by Ayob and Kendall [9]. EMCQ probabilistically accepts lesser quality solutions (similar to simulated annealing [30]) in order to escape from local optima. In EMCQ, the probability density is defined in Equation 3 as:

$$\Psi = e^{-\delta * T/q} \qquad (3)$$

where δ is the difference in fitness value between the current solution (S_i) and the previous solution (S_0) (i.e. $\delta = f(S_i) - f(S_0)$), t is the iteration counter, and q is a control parameter for consecutive non-improving iterations.

Like simulated annealing, the probability density, Ψ, decreases towards zero as T increases. However, unlike simulated annealing, EMCQ does not use any specific cooling schedule, hence, it has no specific parameters that require tuning. Another feature is that EMCQ allows dynamic manipulation of the q parameter to increase or decrease the probability of accepting lesser quality moves. To be specific, q is always incremented upon a poor move, and reset to 1 upon a good move in order to enhance the diversification of the solution.

Referring to the pseudo code of EMCQ in Figure 3, line 1 initializes the populations of the required *t-way* interactions, $I = \{I_1, I_2 \ldots I_M\}$. The value of M depends on the given inputs interaction strength (t), parameter (k) and its corresponding value (v). Specifically, M captures the number of required interactions that needs to be captured in the constructed covering array. Mathematically, M can be obtained as the sum of products of each individual's *t-wise* 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, S. Line 3 randomly initializes the initial population of solutions $Z = \{Z_1, Z_2 \ldots Z_N\}$. Line 4 selects the random initial search operator, H_0. Line 5 applies H_0 to generate initial solution, S_0. Line 6 sets $S_{best} = S_0$ as an initial value and $H_i = H_0$ as the initial search operator. The main loop starts in line 7 and will iterate until the coverage of all interaction tuples (I). Line 8 assigns 1 to variable T which acts as a loop counter. The inner while loop starts in line 9 with Θ_{max} as the maximum number of iterations. Line 10 applies the current H_i to produce best S_i to be added in the final test suite, *Fs*. Line 11 computes the fitness difference, $\delta = f(S_i) - f(S_{best})$. In lines 12-15, if the fitness improves (i.e. $\delta > 0$), H_i is kept for the next iteration. Here, q is reset to 1 in line 14 (i.e. because the fitness improves). Line 17 computes the probability density, Ψ. In line 18, upon a poor move, the solution might be accepted based on the probability density Ψ. If accepted, H_i is kept and q is reset to 1 (as in lines 19-21), otherwise, H_i is changed and q is incremented by 1 (see lines 23-25). Lines 27-28 update the values of S_{best} and T for the next iteration. If there are uncovered *t*-wise interaction, the mentioned procedure is repeated again until termination.

```
Algorithm 1: Pseudo Code for EMCQ
   Input: Interaction strength ($t$), parameter ($k$) and its corresponding value ($v$)
   Output: Final Test Suite $F_s$
 1 Initialize the population of the required t-wise interactions, $I = \{I_0, I_1 \cdots I_M\}$ based on $k$ and $v$ values
 2 Initialize $\Theta_{max}$ iteration and population size $S$
 3 Initialize the random population of solutions $Z = \{Z_0, Z_1, \cdots, Z_{S-1}\}$
 4 Select random $H_0$ operator from the pool of defined heuristic operators $H = \{H_0, H_1, \cdots, H_N\}$
 5 Apply $H_0$ to produce $S_0$
 6 Set $S_{best} = S_0$ and $H_i = H_0$
 7 while (all interaction tuples (I) are not covered) do
 8     $T = 1$
 9     while ($T < \Theta_{max}$) do
10         Apply $H_i$ to produce the best $S_i$ from the population $Z$ and add to $F_s$
11         Compute $\delta = f(S_i) - f(S_{best})$
12         if ($\delta > 0$) then
               // improving fitness
13             Keep $H_i$
14             $q = 1$
15         end
16         else
17             Compute probability density $\Psi = e^{-BT/q}$
18             if ($random(0,1) < \Psi$) then
19                 Keep $H_i$
20                 $q = 1$
21             end
22             else
23                 Randomly select a new search operator $H_{i^*}$ where $i^* \neq i$
24                 $q++$
25             end
26         end
27         $S_{best} = S_i$
28         $T++$
29     end
30 end
```

Figure 3. Pseudo Code for EMCQ

3.1.2 The Choice Function

The Choice Function (CF), termed *Choice Function Accept All Moves*, was first proposed by Kendall et al.[28]. Based on the reward and punish approach, CF utilizes the choice function (*F*) to select from a set of low level heuristics. The corresponding values of *F* are calculated and updated for each individual low level search operator during execution. In our implementation, we adopt the variant of the choice function implementation by Drake et al. [20]; the Modified Choice Function.

Similar to the original choice function implementation, the calculation of *F* depends on three parameters f_1, f_2, and f_3. Parameter f_1 measures the effectiveness of the currently employed search operator h_i. The value of f_1 for a particular search operator is evaluated using Equation 4:

$$f_1(H_i) = I((S_i(H_i))/T(H_i) \; + \phi \, f_1(H_i) \tag{4}$$

where $I(S_i(H_i)$ is the change in solution fitness produced by h_i, $T(H_i)$ is the time taken by the search operator h_i, and ϕ is a parameter from the interval (0,1) which gives greater importance to the heuristic's recent performance.

Parameter f_2 (H_i,H_j) measures the effectiveness of the current search operator h_i when employed immediately following h_j. The value of f_2 is computed using Equation 5.

$$f_2(H_i, H_j) = I((S_i(H_i), (S_j(H_j))/T\big(H_i, H_j\big) \; + \phi f_2(H_i, H_j) \tag{5}$$

where $I((S_i(H_i), (S_j(H_j))$ is the change in fitness of h_i and h_j, $T\ (H_i,\ H_j)$ is the time taken by both the heuristics and ϕ is same as in f_1.

Parameter f_3 captures the time elapsed since the search operator h_k had been called. The parameter f_3 is computed using Equation 6:

$$f_3(H_k) = \tau(H_k) \; where \; k = 0 \; to \; N - 1 \tag{6}$$

where *N* is equal to the total number of available operators.

Using the calculated f_1, f_2, and f_3 values, the *Modified Choice Function F* gives a score to each search operator in order to select the best one based on Equation 7.

$$F_t(H_i) = \phi f_1(H_i) + \; \phi f_2(H_i, H_j) + \; \delta f_3(H_i) \tag{7}$$

where *t* represents the current invocation.

Following the recommendation by Drake et al. [20], the values of ϕ and δ are initially set at 0.5. If the solution fitness improves in any iteration, ϕ is given the highest value of the interval (0, 1) whereas δ is given the lowest value. In case of a low-quality solution, the value of ϕ is decreased by 0.01 and the value of δ is automatically increased (see Equation 9). This leads to the diversification of the heuristic search process. The settings make the intensification factor prominent in the evaluation of *F*. For each iteration, the values of ϕ_t and δ_t in the *Modified Choice Function* are calculated as shown in Equations 8 and 9:

$$\phi_t = \begin{cases} 0.99, & if \; quality \; improves \\ max\{\phi_{t-1} - 0.01, 0.01\}, & if \; quality \; deteriorates \end{cases} \tag{8}$$

$$\delta_t = 1 - \phi_t \tag{9}$$

For each heuristic, the value 0.01 always ensures some non-negative influence of the ϕ on the value of *F*. The complete pseudo code for the Modified Choice Function is shown in Figure 4.

Lines 1-3 perform the necessary initialization related to the *t-way* problem (similar to the case of EMCQ). Line 4 initializes the value of ϕ and δ. Line 5 randomly selects any meta-heuristic H_i to produce an initial solution S_i. The initial values for the three measures f_1, f_2, and f_3 are computed (6-8). Line 9 sets the current heuristic H_i to last heuristic H_j. The main loop starts in line 10 and will iterate until the coverage of all interaction tuples (I). Line 11 assigns 1 to variable T which acts as a loop counter. The inner while loop starts in line 12 with Θ_{max} as the maximum number of iterations. The marking of the heuristics for selection begins in line 13 with the computation of the *Modified Choice Function F*. Line 14 applies the search operator which maximizes *F*. The best S_i is added to the final test suite, *Fs*. The computation of the three measures is performed in lines 15-17. If the solution fitness improves (i.e. $I((S_i(H_i) \geq 0))$ the values of ϕ and δ are set to 0.99 and 0.01 (in lines 19-20) respectively. In line 21, the solution fitness of the last heuristic, H_j is also computed. In the case of a poor fitness, ϕ is decreased linearly (lines 24-25) and the new value for δ *as* $\delta = 1 - \phi$ is computed in line 27. The solution fitness of the current heuristic is set to 0.00 (line 28) as it is poor. Lines 30-31 update H_j and T for the next iteration.

```
Algorithm 2: Modified Choice Function
   Input: Interaction strength ($t$), parameter ($k$) and its corresponding value ($v$)
   Output: Final Test Suite $F_s$
1  Initialize the population of the required t-wise interactions, $I = \{I_0, I_1, \cdots, I_M\}$ based on $k$ and $v$ values
2  Initialize $\Theta_{max}$ iteration and population size $S$
3  Initialize the random population of solutions $Z = \{Z_0, Z_1, \cdots, Z_{S-1}\}$
4  Initialize initial $\phi = 0.5$ and $\delta = 0.5$
5  Randomly select any search operator $H_i$ to produce $S_i$
6  Compute $f_1(H_i) = I((S_i(H_i))/T(H_i)$
7  Set $f_2(H_i, H_j) = 0.00$
8  Set $f_3(H_i) = 0.00$
9  $H_j = H_i$
10 while (all interaction tuples (I) are not covered) do
11     $T = 1$
12     while ($T < \Theta_{max}$) do
13         Compute $F_t(H_i) = \phi f_1(H_i) + \phi f_2(H_i, H_j) + \delta f_3(H_i)$for every search operator in $H$
14         Apply $H_i$ to produce the best $S_i$ from the population $Z$ and add to $F_s$
15         Compute $f_1(H_i) = I((S_i(H_i))/T(H_i) + \phi f_1(H_i)$
16         Compute$f_2(H_i, H_j) = I((S_i(H_i), (S_j(H_j))/T(H_i, H_j) + \phi f_2(H_i, H_j)$
17         Compute $f_3(H_k) += \tau(H_k)$ for every heuristic except $H_i$ and set $f_3(H_i) = 0.00$
18         if ($I((S_i(H_i)) \geqslant 0$) then
               // solution improves
19             Set $\phi = 0.99$
20             Set $\delta = 0.01$
21             $I((S_i(H_j)) = I((S_i(H_i))/T(H_i)$
22         end
23         else
               // solution not improves
24             if $\phi > 0.01$ then
25                 $\phi = \phi - 0.01$
26             end
27             $\delta = 1 - \phi$
28             $I((S_i(H_j)) = 0.00$
29         end
30         $H_j = H_i$
31         $T{+}{+}$
32     end
33 end
```

Figure 4: Modified Choice Function

3.1.3 Improvement Selection Rules

The improvement selection rules (ISR) is proposed by Zamli et al. [49]. The main feature of ISR is that it exploits three rules via its improvement, diversification and intensification operators. The improvement operator checks for improvements in the objective function. The diversification operator measures how diverse the current and the previously generated solutions 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. Apart from its three operators, ISR also exploits a Tabu List to penalize its poorly performing heuristics. Figure 5 summarizes the pseudo code for ISR.

```
Algorithm 3: Improvement Selection Rules
Input: Interaction strength ($t$), parameter ($k$) and its corresponding value ($v$)
Output: Final Test Suite $F_s$
1  Initialize the population of the required t-way interaction tuples, $I = \{I_0, I_1 \cdots I_M\}$
2  Initialize $Tabu_{max}, \Theta_{max}$ iteration and population size $S$
3  Initialize the random population of solutions $Z = \{Z_0, Z_1, \cdots, Z_S\}$
4  Select random $H_0$ operator from the pool of defined heuristic operators $H = \{H_0, H_1, \cdots, H_N\}$
5  Apply $H_0$ to produce $S_0$
6  while (all interaction tuples (I) are not covered) do
7      $T = 1$
8      while ($T < \Theta_{max}$) do
9          Apply $H_i$ to produce the best $S_i$ from the population $Z$ and add to $F_s$
10         $F_1 = Improvement\_Operator(S_i)$
11         $F_2 = Diversify\_Operator(S_i)$
12         $F_3 = Intensify\_Operator(S_i)$
13         if ( the meta-heuristic selection and acceptance mechanism evaluation $\Psi(H_i, F_1, F_2, F_3)$
            improves) then
14             Keep $H_i$
15         end
16         else
17             Add $H_i$ to Tabu List
18             if (Tabu List is Full) then
                   // $Tabu_{max}$
19                 Randomly select a new search operator $H_{i^*}$ where $i^* \neq i$ from Tabu List
20             end
21             else
22                 Select a new search operator $H_i$ from the pool of search operators
23             end
24         end
25         $T++$
26     end
27 end
```

Figure 5. Improvement Selection Rules

Lines 1-3 perform the *t*-way problem initialization (similar to the EMCQ and the choice function described earlier). Lines 4-5 select H_0 randomly to produce S_0 from the four available meta-heuristics. The main loop starts in line 6 and will iterate until the coverage of all interaction tuples (*I*). Line 7 assigns 1 to variable T which acts as a loop counter. The inner while loop starts in line 8 with Θ_{max} as the maximum number of iterations. In line 9, *Hi* is summoned to produce the best S_i to be added to the final test suite, F_s. To decide whether to select a new LLH or not, the three operators, comprising the improvement, diversification and intensification (lines 10-12) will be used. The *improvement operator* compares the current S_i against the previous S_{i-1} from the final test suite F_s. F_1 evaluates to *true only if* $S_i \geq$ *previous* S_{i-1}. The diversification operator exploits the hamming distance measure to evaluate the diversification of each S_i solution (i.e. in terms of how far S_i is from the population of candidate solutions). Like the diversification operator, the intensification operator also exploits the hamming distance to evaluate the intensification of each previous *S* solution. Unlike the diversification operator, the intensification operator measures the intensification value, I_v, of S_i against the final test suite F_s population (i.e. how close is S_{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 F_s population with S_i. 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* $I_v \leq$ *the previous* I_v.

In line 13, the 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 $\Psi (H_i, F_1, F_2, F_3)$ evaluates to *false*, the new H_i will be selected (and the current H_i will be put in the Tabu List).

Referring to lines 18-22, 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 search operator can be chosen more frequently than others owing to the random selection of search operators within the Tabu List (line 19).

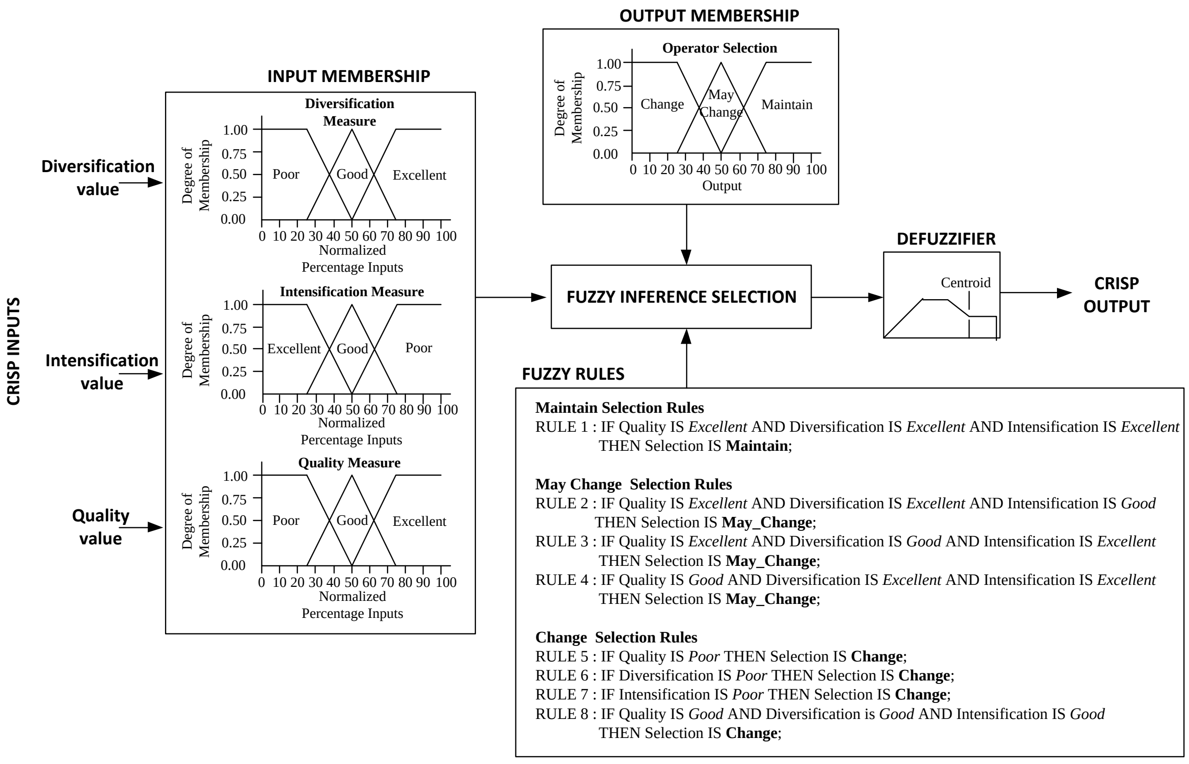
3.1.4 Fuzzy Inference Selection

Finding the right fuzzy membership estimation is actually a very challenging process (as the only restriction that a membership function has to satisfy is its value be in [0,1] range). Design choices are often problem dependent, hence, cannot be easily generalized. Literature [36] suggests at least three approaches for membership function estimation (i.e. *expert-driven approaches* via knowledge acquisition from experts, *data-driven approaches* via structuralisation of data, and *principle of justifiable granularity* via information granularity in terms of sufficient experimental evidence and high specificity). In our current work, we have adopted the variant of *expert-driven* (as we have exploited existing knowledge on the fuzzy inference as well as on our problem domain).

A number of design choices are relevant in the implementation of the proposed Fuzzy Inference Selection (FIS) as follows:

- Mamdani with triangular/trapezoidal membership – As fuzzy rules can be expressed as linguistic constraints that are easy to understand and maintain, Mamdani inference is preferred over Sugeno. Furthermore, previous studies which combine fuzzy and meta-heuristics often favor Mamdani inference. In fact, the majority of these studies used Mamdani inference with centroid defuzzification and implemented either triangular/trapezoidal or Gaussian membership function. Empirical analysis using both types of membership functions showed that triangular/trapezoidal membership functions gave better performance over Gaussian ones [12, 19]. Therefore, in this study, the fuzzy inference system that uses Mamdani type inference with triangular/trapezoidal membership function and centroid defuzzification has been chosen for our implementation.
- Membership cardinality, fuzzy rules and normalization – The proposed FIS as the search operator selection and acceptance mechanism is derived from our earlier work on ISR described in [49]. Like ISR, FIS adopts three operators (i.e. improvement, diversification intensification) based on a Hamming distance measure. Recall that the improvement operator checks for improvements in the quality of the objective function. The diversification operator measures how diverse the current and the previously generated solutions are against the population of potential candidate solutions. Finally, the intensification operator evaluates how close the current and the previously generated solutions are against the population of solutions. Based on the three defined operators, we propose three membership functions representing input for each operator. Owing to its origin, the FIS fuzzy rules have been designed based on the ISR Boolean logic. However, unlike ISR which uses strict Boolean logic, the proposed FIS also accepts partial truth (i.e. based on some degree of membership) allowing more objective control to maintain or potentially change any particular search operator during runtime. In this case, the operator selection is set as the output variable. Concerning normalization of input and output values, we exploit our knowledge on the maximum possible hamming distance range based on the specified input parameters and its values.
- Linguistic terms and their overlapping functions – We have chosen three overlapping (and equal-width) linguistic terms for all membership functions between the multiple interval ranges of 0, 25, 50, 75 and 100. The choice for the number of linguistic terms can be seen as two sides of the same coin. Too many linguistic terms invite more rules, hence, potentially introduce more elaborate computations (and it also affects the widths and the interval ranges). Too little linguistic terms hinder good decision making. As our application involves non-intricate fuzzy decision making, we foresee three linguistic terms for inputs and two linguistic terms for output are sufficiently adequate. Concerning overlapping, we have adopted the work of Mizumoto [34] which suggests that overlapping linguistic terms must start at their center points, where the performance of the fuzzy system is at best (i.e. considering completely non-overlapping of linguistic terms may not fire any rules given out-of-range input values).

Given the aforementioned design choices, we have elaborately experimented with a number of triangular/trapezoidal membership function estimations (with 3 membership functions, 3 input linguistic terms and 2 output linguistic terms) and evaluated our results (i.e. guided by optimal mean results) against the well-known covering arrays as published in [45]. As suggested by our findings, the current membership function estimation (as shown in Figure 6) gives the best overall performances.



Maintain Selection Rules
RULE 1 : IF Quality IS *Excellent* AND Diversification IS *Excellent* AND Intensification IS *Excellent* THEN Selection IS **Maintain**;

May Change Selection Rules
RULE 2 : IF Quality IS *Excellent* AND Diversification IS *Excellent* AND Intensification IS *Good* THEN Selection IS **May_Change**;
RULE 3 : IF Quality IS *Excellent* AND Diversification IS *Good* AND Intensification IS *Excellent* THEN Selection IS **May_Change**;
RULE 4 : IF Quality IS *Good* AND Diversification IS *Excellent* AND Intensification IS *Excellent* THEN Selection IS **May_Change**;

Change Selection Rules
RULE 5 : IF Quality IS *Poor* THEN Selection IS **Change**;
RULE 6 : IF Diversification IS *Poor* THEN Selection IS **Change**;
RULE 7 : IF Intensification IS *Poor* THEN Selection IS **Change**;
RULE 8 : IF Quality IS *Good* AND Diversification is *Good* AND Intensification IS *Good* THEN Selection IS **Change**;

Figure 6. Fuzzy Inference Selection

The block, labeled INPUT MEMBERSHIP takes the crisp values of the three operators and fuzzifies them. The fuzzification process is based on three defined triangular/trapezoidal membership functions with linguistic terms namely *Poor*, *Good* and *Excellent*. It is worth noting that the triangular/trapezoidal membership functions for the diversification operator and improvement operator are identical. The values in the range of 0-50 are considered *Poor*. The values in the range of 25-75 are considered *Good* and the values in the range of 50-100 are considered *Excellent*. In the case of the intensification operator, the *Excellent* range and the *Poor* range are swapped (i.e. *Excellent* range is defined from 0-50 whilst the *Poor* range is defined from 50-100). There is no change as far as the *Good* range is concerned.

Given the defined membership functions and based on the parameter inputs (i.e. *interaction strength (t)*, *parameter (k)* and its corresponding *value (v)*), each of the crisp input from each operator need to undergo normalized scaling to fit in the defined percentage range. In general, the normalized values are computed as follows (based on Equation 10):

$$F_{scaled} = (F_{actual} * 100)/(F_{max}) \qquad (10)$$

The F_{max} value depends on the operator. Concerning the diversification operator, the F_{max} corresponds to the maximum diversity possible (i.e. all the values within inputs are completely changed). For this reason, F_{max} is always equal to the input parameter *(k)*. As for the intensification operator, F_{max} corresponds to the maximum intensification possible (i.e. again with all the values within inputs are completely changed). As such, F_{max} for the intensification operator is also always equal to the input *parameter (k)*. Contrary to this, F_{max} calculation is different for the improvement operator. Here, F_{max} corresponds to the maximum possible interaction coverage given as input *parameter (k)* and *interaction strength (t)*. Specifically, F_{max} for the improvement operator can be mathematically defined by Equation 11.

$$F_{max}\ for\ Improvement\ Operator = {}^{k}C_t = \frac{k!}{t!\,(k-t)!} \qquad (11)$$

For the OUTPUT MEMBERSHIP block, a single output operator called Selection is defined. The Selection operator has three linguistic terms called *Change, May Change,* and *Maintain* represented by the triangular/ trapezoidal membership function (similar to intensification and diversification operators) taking the ranges of 0-50, 25-75 and 50-100 respectively.

The FUZZY RULES block lists the linguistic rules of FIS. The total number of rules *r* for a fuzzy system is determined by Equation 12.

$$r = \prod_i^N f_i \qquad (12)$$

where N is the total number of crisp inputs and f_i is the number of terms for each input variable.

In our case, there are potentially 3^3 or 27 rules for the FIS (as each operator takes three linguistic terms). Based on our observation, the rules can be reduced to 8 rules as shown in Figure 6. Specifically, the selection of the search operator will not be changed (i.e. Maintain) if all three operators' values are evaluated as *Excellent*. The search operator may be changed (i.e. May Change) if any of the two operators' values are *Excellent* and the third value is *Good*. FIS changes (i.e. Change) the search operator for the next iteration if any one of the operators is *Poor* or all operators are *Good.*

Finally, the Fuzzy Inference Selection aggregates the reasoning and takes fuzzy actions in light of input/output memberships, linguistic variables and fuzzy rules. The fuzzy results are then forwarded to the DEFUZZIFIER block. This block translates the fuzzy results into crisp output using the Center of Gravity, based on the defined DEFUZZIFIER block in order to produce crisp values for the control variables (see Equation 13).

$$U = \frac{\int_{Min}^{Max} U \, \mu\,(U)\, dU}{\int_{Min}^{Max} \mu\,(U)\, dU} \qquad (13)$$

Summing up, Figure 7 summarizes the complete FIS pseudo code.

```
Algorithm 4: Fuzzy Inference Selection
   Input: Interaction strength ($t$), parameter ($k$) and its corresponding value ($v$)
   Output: Final Test Suite $F_s$
1  Initialize the population of the required t-way interaction tuples, $I = \{I_0, I_1 \cdots I_M\}$
2  Initialize $\Theta_{max}$ iteration and population size $S$
3  Initialize the random population of solutions $Z = \{Z_0, Z_1, \cdots, Z_S\}$
4  Compute $F_{max}$ for all operators
5  Define Fuzzy Rules
6  while (all interaction tuples (I) are not covered) do
7      $T = 1$
8      while ($T < \Theta_{max}$) do
9          Apply $H_i$ to produce the best $S_i$ from the population $Z$ and add to $F_s$
10         $F_1 = Improvement\_Operator(S_i)$
11         $F_2 = Diversify\_Operator(S_i)$
12         $F_3 = Intensify\_Operator(S_i)$
13         Compute $F_{scaled} = (F_{actual} \times 100)/F_{max}$ for all operators
14         Transform into linguistic terms based on the defined membership functions
15         Apply FIS
16         Defuzzify to produce crisp output
17         Set Selection = crisp output
18         if ($Selection \geq 0.0$ and $\leq 40.0$) then
               // Selection = Change
19             Randomly select a new search operator $H_{i^*}$ where $i^* \neq i$
20         end
21         else if ($Selection > 40.0$ and $\leq 60.0$) then
               // Selection = May Change
22             Randomly select a new search operator $H_i$
23         else
               // Selection =Maintain
24             Keep $H_i$
25         end
26         $T++$
27     end
28 end
```

Figure 7. Fuzzy Inference Selection

Lines 1-3 perform t-way problem initialization similar to the earlier described hyper-heuristics. The maximum operator value, *F_Max*, is computed in line 4. Line 5 defines the fuzzy rules used by the FIS. The main loop starts in line 6 which repeats the necessary steps until the coverage of all interaction tuples *(I)*. Line 7 sets variable *T* to value 1. The inner loop starts in line 8 that runs for Θ_{max}. Line 9 selects H_i in order to produce a solution S_i and add it to *Fs*. Lines 10-12 compute the values of the three operators (i.e. improvement F_1, diversification F_2, as well as intensification F_3). These values will be utilized for the fuzzy based selection of H_i. Line 13 computes the s*caled* value for each operator F_i as F_{scaled} as $F_{scaled} = (F_{actual}\times 100)/F_{max}$. Lines 14-17, encompass the FIS logic. In line 14, the scaled value of each operator and the Selection output variable are translated into linguistic terms with trapezoidal membership functions (as depicted in the INPUT/OUTPUT MEMBERSHIP blocks in Figure 6). FIS, in line 15, combines all the fuzzy information for the DEFUZZIFIER block. The defuzzifier produces the crisp output in Line 16. Line 17 assigns the crisp output to the Selection variable. In lines 18-25, the fuzzy inference selection will decide, based on the Selection value, whether to *maintain*, *change* or *may change* the heuristic H_i for the next iteration. In case, the operator is found *Poor* (lines 18-19), the current H_i will be replaced in the next iteration. When the Selection variable is in range (Selection > 40.0 and ≤ 60.0), the current H_i may or may not be changed for the next iteration (lines 21-22). FIS keeps the current H_i (line 24) for the next iteration if the Selection variable is greater than 60. Line 26 updates *T* for the next iteration.

3.2 Description of the Search Operators

The next subsections provide the description of the adopted search operators.

3.2.1 The Genetic Algorithm Crossover Search Operator

The crossover search operator is derived from the Genetic Algorithm [25]. The complete algorithm is defined in Figure 8. Initially, S_{best} is set to Z_0 in line 1. The loop starts in line 2. The crossover operation occurs between a randomly selected Z_i against the existing Z_i in the population over the randomized length (α) (in lines 3-5). If the newly updated Z_i has a better fitness value, the value of Z_i is updated accordingly (in lines 6-7). The fitness of the current Z_i is checked against S_{best}. S_{best} will be updated if it has better fitness than Z_i (in lines 9-10).

```
Algorithm 5: GA Crossover Search Operator
   Input: The population $Z = \{Z_1, Z_2, \cdots, Z_S\}$
   Output: $S_{best}$ and the updated population $Z\prime = \{Z'_0, Z'_2, \cdots, Z'_{S-1}\}$
1  $S_{best} = Z_0$
2  for $i = 0$ to population size $S - 1$ do
       /* ****** Crossover ******                                    */
3      Randomly select $Z_j$ such that $j \neq i$
4      Set crossover length $\alpha$ = random(0,length of $Z_j$))
5      $Z_i^{(t+1)}$ = exchange from position 0 till $\alpha$ from $Z_j$ to $Z_i^{(t)}$
6      if $Z_i^{(t+1)}$ is better than $Z_i^{(t)}$, i.e. $f(Z_i^{(t+1)}) > f(Z_i^{(t)})$ then
7          $Z_i^{(t)} = Z_i^{(t+1)}$
8      end
9      if $f(Z_i^{(t)}) > f(S_{best})$ then
10         $S_{best} = Z_i^{(t)}$
11     end
12 end
13 Return $S_{best}$
```

Figure 8. GA Crossover Search Operator

3.2.2 Teaching Learning based Optimization Peer Learning Search Operator

As the name suggests, the TLBO peer learning search operator is derived from the learning phase of the Teaching Learning based Optimization Algorithm [39]. The algorithm was originally proposed as a local search operator. Figure 9 presents the complete algorithm.

Initially, S_{best} is set to Z_0 in line 1. The loop starts from line 2. The learning happens within the loop (lines 3-10). The idea is that each student attempts to improve his knowledge through interaction with his peers. To be specific, the student Z_i will select a random peer learner Z_j (where $Z_i \neq Z_j$) (line 3). The scaling factor is set randomly chosen from (0,1) in line 4. If Z_i has better fitness than Z_j, the latter is moved toward the former (line 6) and vice versa (line 9). If the newly updated Z_i has a poorer fitness value, no update is made to Z_i (in lines 11-12). The fitness of the current Z_i is checked against S_{best} and will be updated if it has better fitness than Z_i (in lines 14-15).

```
Algorithm 6: TLBO Algorithm's Peer Learning Search Operator
   Input: The population $Z = \{Z_1, Z_2, \cdots, Z_S\}$
   Output: $S_{best}$ and the updated population $Z\prime = \{Z'_0, Z'_2, \cdots, Z'_{S-1}\}$
1  $S_{best} = Z_0$
2  for $i = 0$ to population size $S - 1$ do
       /* ****** Peer Learning ******                                    */
3      Randomly select $Z_j$ such that $j \neq i$
4      Set scaling factor $\rho = random(0, 1)$
5      if $Z_i^{(t)}$ is better than $Z_j$ , i.e. $f(Z_i^{(t)}) > f(Z_j)$ then
6          $Z_i^{(t+1)} = Z_i^{(t)} + \rho \times (Z_i^{(t)} - Z_j)$
7      end
8      else
9          $Z_i^{(t+1)} = Z_i^{(t)} + \rho \times (Z_j^{(t)} - Z_i)$
10     end
11     if $Z_i^{(t+1)}$ is better than $Z_i^{(t)}$ i.e. $f(Z_i^{(t+1)}) > f(Z_i^{(t)})$ then
12         $Z_i^{(t)} = Z_i^{(t+1)}$
13     end
14     if $f(Z_i^{(t)}) > f(S_{best})$ then
15         $S_{best} = Z_i^{(t)}$
16     end
17 end
18 Return $S_{best}$
```

Figure 9. TLBO Algorithm's Peer Learning Search Operator

3.2.3 Flower Pollination Algorithm Global Pollination Operator

The FPA global pollination search operator is derived from the Flower Pollination Algorithm [48]. The global pollination operator exploits *Lévy Flight* motion to update all the (column-wise) values for Z_i of interest instead of only perturbing one value, thus, making it a global search operator. The complete algorithm is summarized in Figure 10.

Considering the flow of the global pollination operator, S_{best} is initially set to Z_0 in line 1. The loop starts in line 2. The value of Z_i will be iteratively updated using the transformation equation exploiting the *Lévy Flight* motion (in lines 4-6). 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 [47]. Within this algorithm, a step length can be defined as (See Equation 14):

$$Step = \frac{u}{[v]^{\frac{1}{\beta}}} \qquad (14)$$

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

$$u \approx N(0, {\sigma_u}^2) \cdot \sigma_u \qquad v \approx N(0, {\sigma_v}^2) \cdot \sigma_v \qquad (15)$$

For v value estimation, we use $\sigma_v = 1$. For u value estimation, we evaluate the Gamma function(Γ) with the value of $\beta = 1.5$ [46], and obtain σ_u using Equation 16:

$$\sigma_u = \left| \frac{\Gamma(1+\beta) \times sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{(1+\beta)}{2}\right) \times \beta \times 2\left(\frac{\beta-1}{2}\right)} \right|^{\frac{1}{\beta}} \qquad (16)$$

In our case, the Gamma function(г) implementation is adopted from William et al. [37].

If the newly updated Z_i has a better fitness value, then the current Z_i is replaced (in lines 6-7). The value of S_{best} is also updated if it has a better fitness value than that of Z_i (in lines 8-9). If the newly updated Z_i has a poorer fitness value, no update is made to Z_i (in lines 12-16). The fitness of the current Z_i is checked against S_{best}. S_{best} will be updated if it has better fitness than Z_i (in lines 13-14).

```
Algorithm 7: Flower Algorithm's Global Pollination Search Operator
  Input: The population $Z = \{Z_1, Z_2, \cdots, Z_S\}$
  Output: $S_{best}$ and the updated population $Z\prime = \{Z'_0, Z'_2, \cdots, Z'_{S-1}\}$
1 $S_{best} = Z_0$
2 for $i = 0$ to population size $S - 1$ do
      /* ****** Global Pollination ****** */
3     Set scaling factor $\rho = random(0, 1)$
4     Generate a step vector Ŀ which obeys Lévy Flight distribution
5     Update the current population $Z_i^{(t+1)} = Z_i^{(t)} + \rho \times Ŀ \times (S_{best} - Z_i^{(t)})$
6     if $(f(Z_i^{(t+1)}) > f(Z_i^{(t)})$ then
7         $Z_i^{(t)} = Z_i^{(t+1)}$
8         if $(f(Z_i^{(t+1)}) > f(S_{best}))$ then
9             $S_{best} = Z_i^{(t+1)}$
10        end
11    end
12    else
13        if $(f(Z_i^{(t)}) > f(S_{best}))$ then
14            $S_{best} = Z_i^{(t)}$
15        end
16    end
17 end
18 Return $S_{best}$
```

Figure 10. Flower Algorithm's Global Pollination Operator

3.2.4 Jaya Algorithm's Search Operator

The Jaya search operator is derived from the Jaya algorithm [38]. The complete description of the Jaya operator is summarized in Figure 11.

Unlike the search operators described earlier (i.e. keeping track of only S_{best}), the Jaya search operator keeps track of both S_{best} and S_{poor}. As seen in line 6, the Jaya search operator exploits both S_{best} and S_{poor} as part of its transformation equation. Although biased towards global search, the transformation equation can also address local search. In the case when $\Delta S = S_{best} - S_{poor}$ is sufficiently small, the transformation equation offset (in line with the term $\upsilon \cdot (S_{best} - Z_i) - \zeta \cdot (S_{poor} - Z_i))$ will be insignificant relative to the current location of Z_i allowing steady intensification process.

As far as the flow of the Jaya operator is concerned, lines 1-2 sets up the initial values for $S_{best} = Z_0$ and $S_{poor} = S_{best}$. The loop starts from line 3. Two random values υ and ζ are generated to compensate and scale down the delta differences between Z_i with S_{best} and S_{poor} in the transformation equation (in lines 4-6). If the newly updated Z_i has a better fitness value, then the current Z_i is replaced accordingly (in lines 7-8). In a similar manner, the value of S_{best} is also updated if it has a better fitness value than that of Z_i (in lines 9-10). In the case when the newly updated Z_i has a poorer fitness value, no update is made to Z_i (in lines 13-20). In such a case, the fitness of the current Z_i is checked against both the fitness of S_{best} and S_{poor}. If the fitness of the current Z_i is better than that of S_{best}, Z_i is assigned to S_{best} (in lines 14-15). Similarly, if the fitness of the current Z_i is poorer than that of S_{poor}, Z_i is assigned to S_{poor} (in lines 17-18).

```
Algorithm 8: Jaya Search Operator
   Input: The population $Z = \{Z_1, Z_2, \cdots, Z_S\}$
   Output: $S_{best}$ and the updated population $Z\prime = \{Z_0', Z_2', \cdots, Z_{S-1}'\}$
 1 $S_{best} = Z_0$
 2 $S_{poor} = S_{best}$
 3 for $i = 0$ to population size $S - 1$ do
       /* ****** Jaya ******                                              */
 4     Set scaling factor $\mho = random(0, 1)$
 5     Set scaling factor $\zeta = random(0, 1)$
 6     Update the current population $Z_i^{(t+1)} = Z_i^{(t)} + \mho \times (S_{best} - Z_i^{(t)}) - \zeta \times (S_{poor} - Z_i^{(t)})$
 7     if $(f(Z_i^{(t+1)}) > f(Z_i^{(t)})$ then
 8         $Z_i^{(t)} = Z_i^{(t+1)}$
 9         if $(f(Z_i^{(t+1)}) > f(S_{best}))$ then
10             $S_{best} = Z_i^{(t+1)}$
11         end
12     end
13     else
14         if $(f(Z_i^{(t)}) > f(S_{best}))$ then
15             $S_{best} = Z_i^{(t)}$
16         end
17         if $(f(Z_i^{(t)}) < f(S_{poor}))$ then
18             $S_{poor} = Z_i^{(t)}$
19         end
20     end
21 end
22 Return $S_{best}$
```

Figure 11. Jaya Search Operator

4. The Experiments

Our experiments focus on three related goals: (1) to characterize the performance of the implemented hyper-heuristic with each other; (2) to gauge the distribution pattern of the selected search operators by each hyper-heuristic selection and acceptance mechanism; and (3) to benchmark the implemented hyper-heuristics against other meta-heuristic approaches.

We have divided our experiments into three parts. In the first part, we highlight the average time and size performance of the implemented hyper-heuristics. In the second part, we benchmark the size performance of our hyper-heuristic implementation against themselves as well as against existing meta-heuristic based strategies.

As highlighted in an earlier section (see Figure 2), although all the hyper-heuristics are adopting different selection and acceptance mechanisms, they employ the same low level search operators (i.e. based on GA crossover search operator, TLBO peer learning search operator, FPA global pollination search operator, and Jaya search operator). In our experiments, all the hyper-heuristics have the same population size (Θ_{max} = 20) and the same maximum number of iterations (*S=100*). All the hyper-heuristics use the same data structure and are implemented using the Java programming language. For these reasons, comparative experiments amongst the various hyper-heuristics, we believe, are fair.

The same observation cannot be generalized in the case of meta-heuristics based strategies. Each meta-heuristic requires the specific parameter settings (e.g. PSO relies on population size, inertia weight, social and cognitive parameters, while Cuckoo Search relies on elitism probability, iteration and population). As the meta-heuristic based strategy implementations are not available to us, we cannot modify the algorithm internal settings and fairly run our own experiments. For this reason, we opt only to compare test size performance and its average.

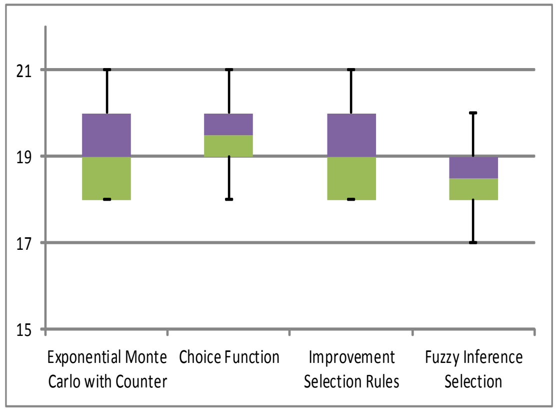
Our experimental platform comprises of a PC running Windows 10, CPU 2.9 GHz Intel Core i5, 16 GB 1867 MHz DDR3 RAM and a 512 MB of flash HDD. We represent all our experimental results in the tables for all the corresponding strategies. With the exception of the hyper-heuristic results, all other results are based on each strategy's respective publication. Cells marked "NA" (not available) indicate that the results were not available for those specific configurations of the strategy. In line with the published results and the evidence in the literature, we have run each experiment 30 times and report the best and the average size (as shaded cells) as well as the best average time (as bold cells in Table 3) whenever possible for these runs to give a better indication for the performance of the strategies of interest. Additionally, we also record the normalized percentage distribution of low level search operators by each hyper-heuristic selection and acceptance mechanism for each benchmark experiment undertaken.

4.1. *Characterizing the Implemented Hyper-Heuristic Selection and Acceptance Mechanisms*

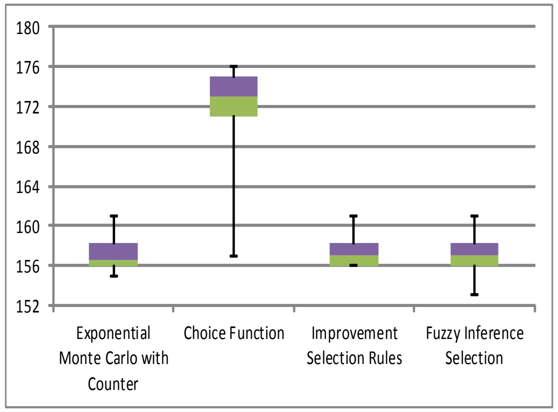
To characterize the size and average time performances of the implemented hyper-heuristic selection and acceptance mechanism, we have adopted an experiment from [45]. Table 3 highlights our results. In order to depict the variation of the obtained results (i.e. patterns) from each implemented hyper-heuristic selection and acceptance mechanism, we construct the box plots (see Figure 12) based on the results in Table 3. Meanwhile, Figure 13 summarizes the percentage distribution of low level search operators by each hyper-heuristic strategy.

Table 3. Size and Average Time Performances for the Implemented Selection and Acceptance Mechanisms

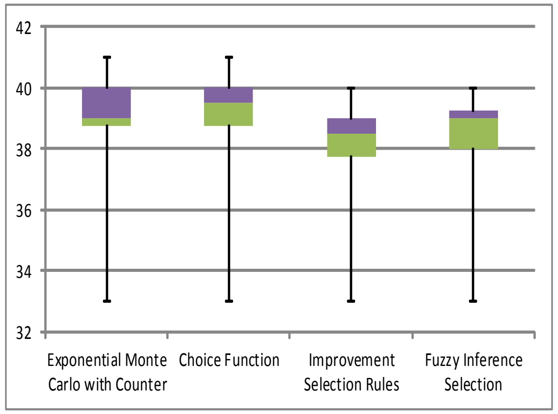
CA	Exponential Monte Carlo with Counter			Choice Function			Improvement Selection Rules			Fuzzy Selection		
	Size		Ave Time (sec)	Size		Ave Time (sec)	Size		Ave Time (sec)	Size		Ave Time (sec)
	Best	Ave		Best	Ave		Best	Ave		Best	Ave	
CA$_1$ (N; 2, 3^{13})	18	19.05	29.71	18	19.45	**20.37**	18	18.90	30.12	17	18.65	30.28
CA$_2$ (N; 2, 10^{10})	155	157.20	**116.49**	157	172.05	116.71	156	157.35	127.18	153	157.10	131.21
CA$_3$ (N; 3, 3^6)	33	38.85	13.32	33	38.90	**12.17**	33	37.75	13.71	33	38.20	13.64
CA$_4$ (N; 3, 6^6)	323	326.70	**165.28**	323	327.40	165.70	322	326.20	168.22	323	326.15	170.56
CA$_5$ (N; 3, 10^6)	1485	1496.50	**999.65**	1483	1499.25	1000.10	1482	1486.80	1003.99	1481	1486.20	1005.23
CA$_6$ (N; 3, $5^2 4^2 3^2$)	100	107.35	42.13	100	113.20	**36.76**	100	105.55	48.22	100	105.95	43.35



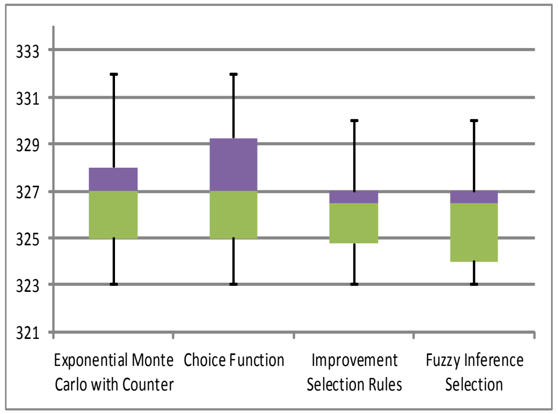
a) Box Plot for CA_1 $(N; 2, 3^{13})$



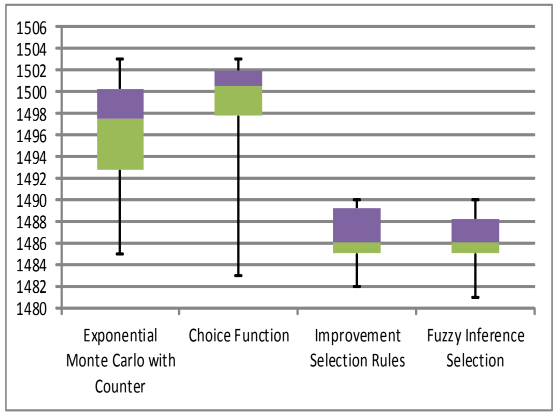
b) Box Plot for CA_2 $(N; 2, 10^{10})$



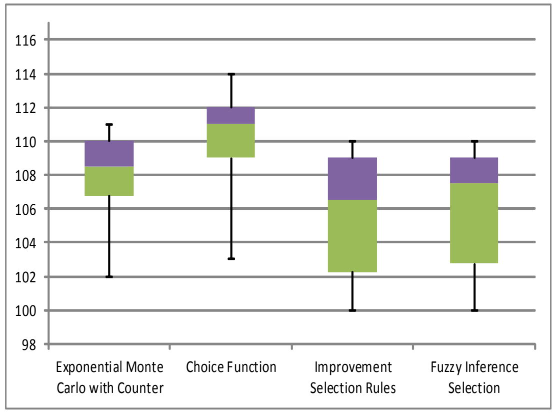
c) Box Plot for CA_3 $(N; 3, 3^{6})$



d) Box Plot for CA_4 $(N; 3, 6^{6})$



e) Box Plot for CA_5 $(N; 3, 10^{6})$



f) Box Plot for CA_6 $(N; 3, 5^{2}4^{2}3^{2})$

Figure 12. Box Plots for Table 3

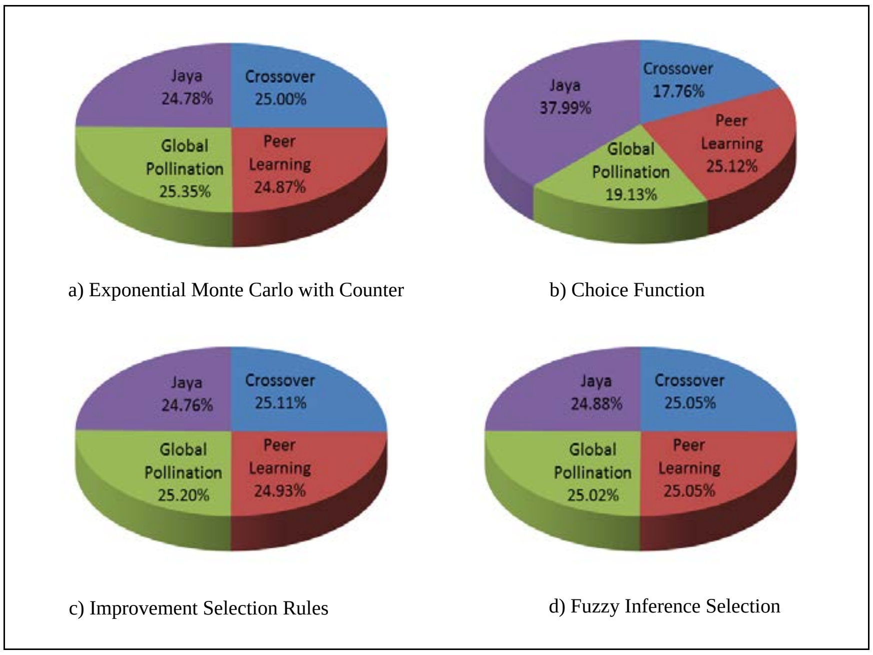


Figure 13. Search Operator Normalized Percentage Distribution for all CA_1-CA_6 in Table 3

4.2. ***Benchmarking against existing Meta-Heuristic based Strategies***

To put our work into perspective, we also benchmark our work against existing meta-heuristic based strategies as published in [4, 32, 45]. Tables 4–9 depict the results obtained for the comparative experiments. Figures 14–19 summarize the percentage distribution of low level search operators by each hyper-heuristic strategy of interest.

Table 4. Size Performance for CA (N; 2, 3^k)

K	Meta-Heuristic based Strategies								Hyper-Heuristic based Strategies							
	PSTG [4]		DPSO [45]		APSO [32]		CS [2]		Exponential Monte Carlo with Counter		Choice Function		Improvement Selection Rules		Fuzzy Selection	
	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave
3	9	9.55	NA	NA	9	9.21	9	9.60	9	9.83	9	9.7	9	9.90	9	9.67
4	9	10.15	9	9.00	9	9.95	9	10.00	9	9.00	9	9.0	9	9.00	9	9.00
5	12	13.81	11	11.53	11	12.23	11	11.80	11	11.24	11	11.3	11	11.30	11	11.23
6	13	15.11	14	14.50	12	13.78	13	14.20	14	14.27	13	14.36	13	14.46	13	14.03
7	15	16.94	15	15.17	15	16.62	14	15.60	15	15.07	15	15.23	15	15.10	14	15.07
8	15	17.57	15	16.00	15	16.92	15	15.80	15	15.77	15	16.16	15	15.90	15	15.79
9	17	19.38	15	16.43	16	18.31	16	17.20	15	16.23	15	16.43	15	16.10	15	15.97
10	17	19.78	16	17.30	17	18.12	17	17.80	16	17.10	16	17.2	16	17.50	16	17.03
11	17	20.16	17	17.70	NA	NA	18	18.60	17	18.90	18	18.50	17	18.30	16	17.45
12	18	21.34	16	17.93	NA	NA	18	18.8	16	17.96	17	18.29	17	18.40	16	17.80

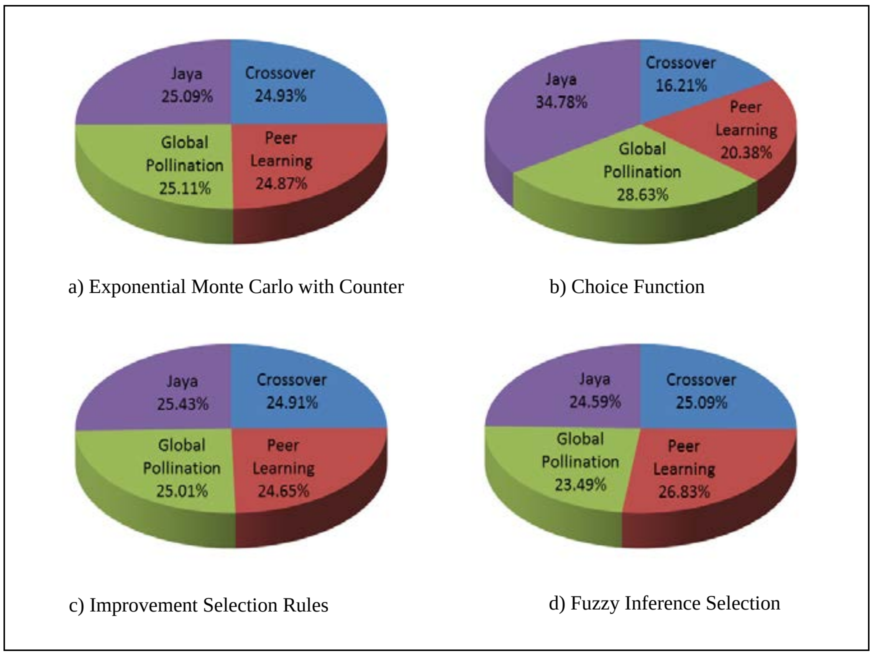


Figure 14. Search Operator Normalized Percentage Distribution for CA (N; 2, 3^k) in Table 4

Table 5. Size Performance for CA (N; 3, 3^k)

K	Meta-Heuristic based Strategies								Hyper-Heuristic based Strategies							
	PSTG [4]		DPSO [45]		APSO [32]		CS [2]		Exponential Monte Carlo with Counter		Choice Function		Improvement Selection Rules		Fuzzy Inference Selection	
	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave
4	27	29.30	NA	NA	27	28.90	28	29.00	27	28.83	27	29.20	27	30.06	27	27.23
5	39	41.37	41	43.17	41	42.20	38	39.20	39	41.47	38	41.40	39	41.60	37	41.30
6	45	46.76	33	38.30	45	46.51	43	44.20	33	38.63	33	38.37	33	38.47	33	36.77
7	50	52.20	48	50.43	48	51.12	48	50.40	49	50.46	49	50.50	49	50.47	48	50.40
8	54	56.76	52	53.83	50	54.86	53	54.80	52	53.27	52	53.93	52	53.27	53	53.40
9	58	60.30	56	57.77	59	60.21	58	59.80	56	57.79	57	58.07	56	57.87	56	57.77
10	62	63.95	59	60.87	63	64.33	62	63.60	59	61.17	60	60.77	60	60.10	59	61.03
11	64	65.68	63	63.97	NA	NA	66	68.20	63	63.87	64	65.27	63	63.67	63	63.53
12	67	68.23	65	66.83	NA	NA	70	71.80	65	67.61	66	68.13	65	66.93	65	66.13

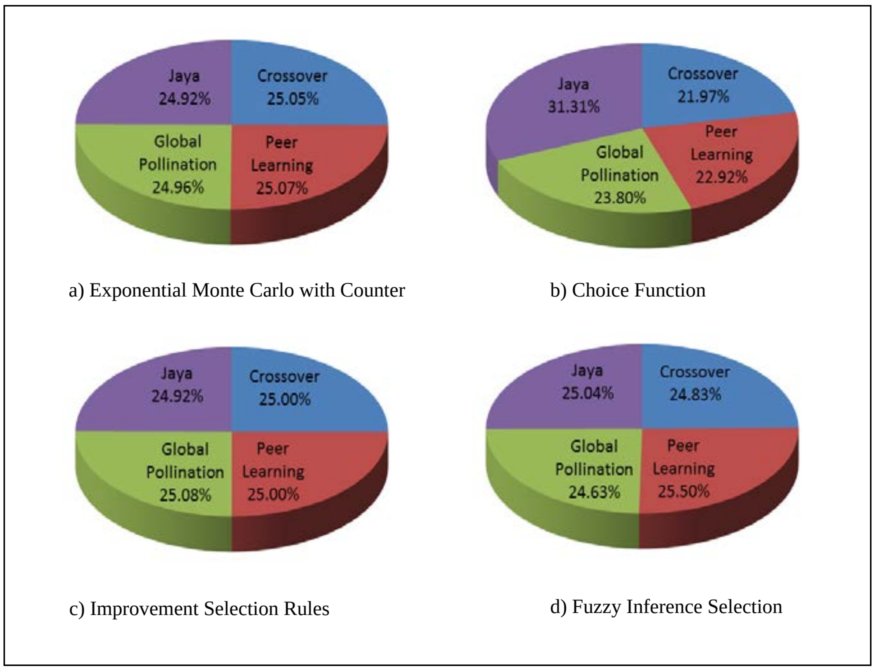
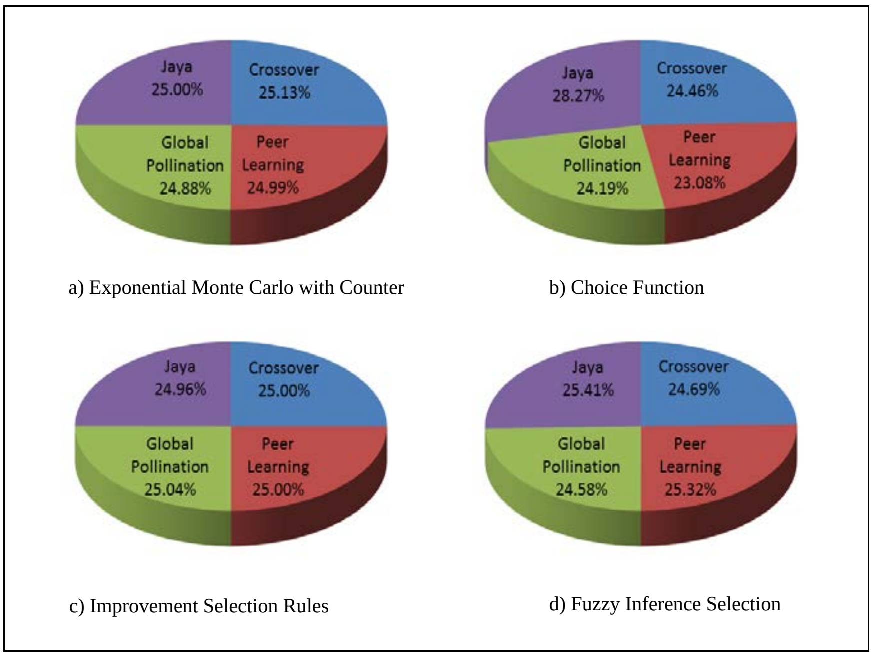


Figure 15. Search Operator Normalized Percentage Distribution for *CA* (N; 3, 3^k) in Table 5

Table 6. Size Performance for *CA* $(N; 4, 3^k)$

k	Meta-Heuristic based Strategies								Hyper-Heuristic based Strategies							
	PSTG [4]		DPSO [45]		APSO [32]		CS [2]		Exponential Monte Carlo with Counter		Choice Function		Improvement Selection Rules		Fuzzy Inference Selection	
	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave
5	96	97.83	NA	NA	94	96.33	94	95.80	81	84.23	81	89.07	81	88.27	81	87.27
6	133	135.31	131	134.37	129	133.98	132	134.20	130	133.33	129	133.83	129	134.17	129	134.10
7	155	158.12	150	155.23	154	157.42	154	156.80	149	154.27	151	155.17	147	153.53	147	153.90
8	175	176.94	171	175.60	178	179.70	173	174.80	172	174.96	173	175.47	171	174.83	171	174.47
9	195	198.72	187	192.27	190	194.13	195	197.80	160	187.87	142	190.53	171	190.33	159	189.47
10	210	212.71	206	219.07	214	212.21	211	212.20	206	209.00	205	208.83	206	208.77	206	208.67
11	222	226.59	221	224.27	NA	NA	229	231.00	221	224.67	222	226.13	221	224.33	221	223.13
12	244	248.97	237	239.85	NA	NA	253	255.80	237	238.51	237	239.21	236	238.11	235	237.43



a) Exponential Monte Carlo with Counter

b) Choice Function

c) Improvement Selection Rules

d) Fuzzy Inference Selection

Figure 16. Search Operator Normalized Percentage Distribution for *CA* $(N; 4, 3^k)$ in Table 6

Table 7.Size Performance for *CA* $(N; 2, v^7)$

V	Meta-Heuristic based Strategies								Hyper-Heuristic based Strategies							
	PSTG [4]		DPSO [45]		APSO [32]		CS [2]		Exponential Monte Carlo with Counter		Choice Function		Improvement Selection Rules		Fuzzy Inference Selection	
	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave
2	6	6.82	7	7.00	6	6.73	6	6.80	7	7.00	7	7.00	7	7.00	7	7.00
3	15	15.23	14	15.00	15	15.56	15	16.20	15	15.13	15	15.13	15	15.17	14	15.00
4	26	27.22	24	25.33	25	26.36	25	26.40	24	25.07	24	25.47	23	25.00	24	24.87
5	37	38.14	34	35.47	35	37.92	37	38.60	34	35.83	34	36.63	34	35.90	34	35.70
6	NA	NA	47	49.23	NA	NA	NA	NA	48	49.00	48	49.67	47	49.51	47	48.75
7	NA	NA	64	66.37	NA	NA	NA	NA	64	65.93	64	66.85	64	66.25	64	65.65

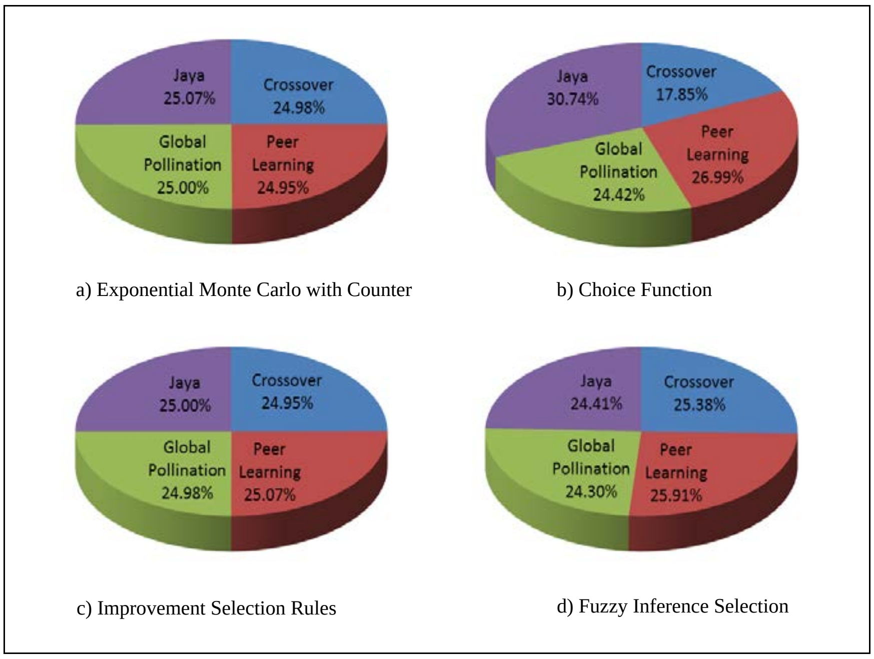


Figure 17. Search Operator Normalized Distribution for *CA (N; 2, v^7)* in Table 7

Table 8. Size Performance for CA *(N; 3, v^7)*

v	Meta-Heuristic based Strategies								Hyper-Heuristic based Strategies							
	PSTG [4]		DPSO [45]		APSO [32]		CS [2]		Exponential Monte Carlo with Counter		Choice Function		Improvement Selection Rules		Fuzzy Inference Selection	
	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave
2	13	13.61	15	15.06	15	15.80	12	13.80	14	15.17	15	15.17	15	15.20	12	15.00
3	50	51.75	49	50.60	48	51.12	49	51.60	49	50.6	48	50.53	48	50.57	48	50.47
4	116	118.13	112	115.27	118	120.41	117	118.40	113	115.7	114	115.07	113	115.37	112	114.90
5	225	227.21	216	219.20	239	243.29	223	225.40	217	220.37	215	219.00	216	218.65	216	218.60
6	NA	NA	365	370.57	NA	NA	NA	NA	365	373.91	369	374.43	365	373.51	366	370.20
7	NA	NA	574	577.67	NA	NA	NA	NA	575	579.00	575	580.91	575	579.75	575	577.80

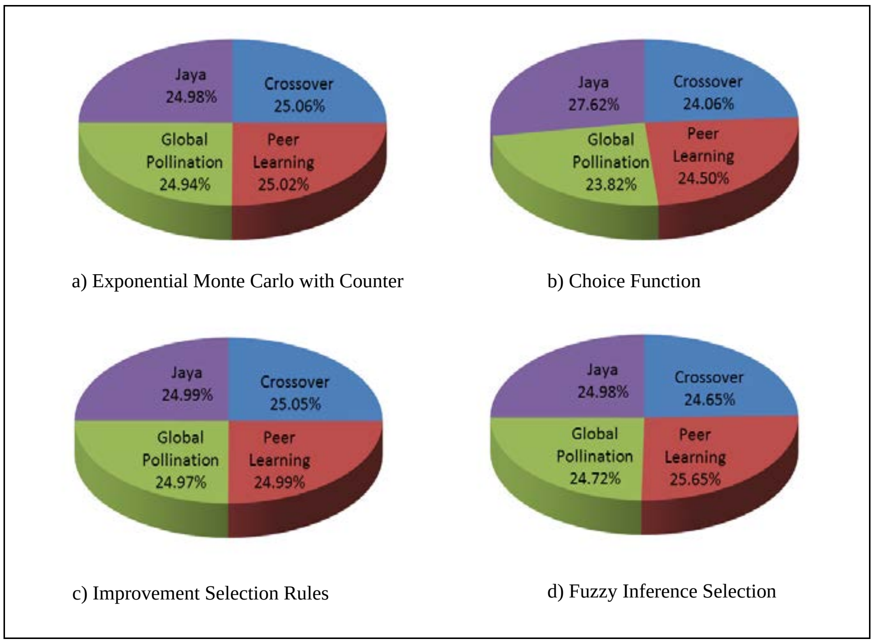


Figure 18. Search Operator Normalized Percentage Distribution for *CA (N; 3, v^7)* in Table 8

Table 9. Size Performance for *CA (N; 4, v^7)*

V	Meta-Heuristic based Strategies								Hyper-Heuristic based Strategies							
	PSTG [4]		DPSO [45]		APSO [32]		CS [2]		Exponential Monte Carlo with Counter		Choice Function		Improvement Selection Rules		Fuzzy Inference Selection	
	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave
2	29	31.49	34	34.00	30	31.34	27	29.60	31	32.23	31	31.80	31	32.27	26	31.67
3	155	157.77	150	154.73	153	155.2	155	156.80	151	155.30	151	155.27	151	154.53	150	154.40
4	487	489.91	472	481.53	472	478.9	487	490.20	479	484.83	479	485.17	479	484.00	480	484.00
5	1176	1180.63	1148	1155.63	1162	1169.94	1171	1175.20	1156	1161.47	1151	1160.03	1154	1162.43	1154	1161.03
6	NA	NA	2341	2357.73	NA	NA	NA	NA	2348	2364.23	2353	2369.91	2352	2367.11	2349	2363.11
7	NA	NA	4290	4309.60	NA	NA	NA	NA	4294	4311.10	4295	4312.70	4295	4311.90	4293	4310.54

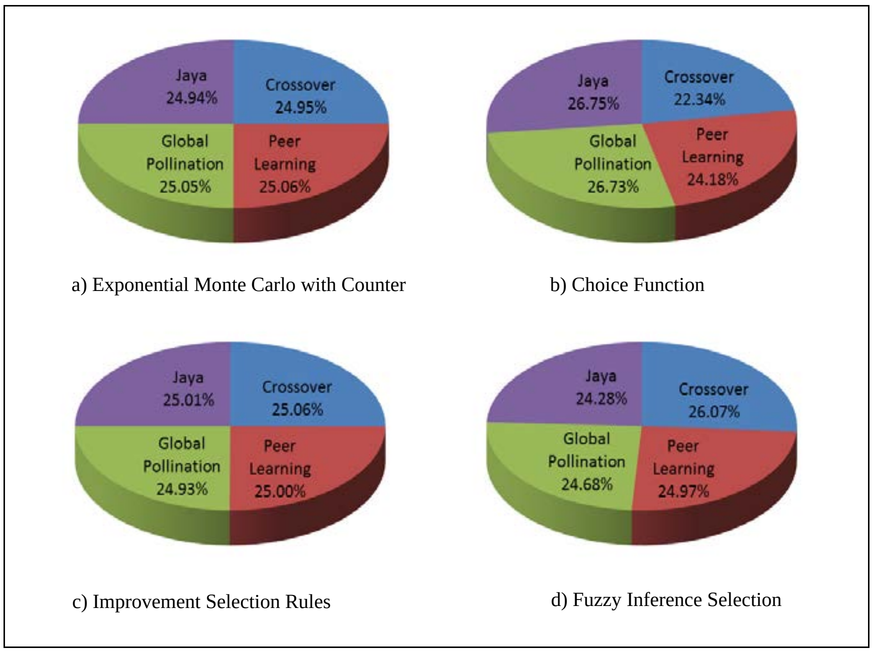


Figure 19. Search Operator Normalized Percentage Distribution for *CA (N; 4, v^7)* in Table 9

4.3. *Statistical Analysis*

We conduct our statistical analysis for all the obtained results (from Table 3 and Tables 4–9) based on the 1xN pair comparisons with 95% confidence level (i.e. α=0.05) and 90% confidence level (i.e. α=0.1). The Wilcoxon Rank-Sum is used to find whether the control strategy presents statistical difference with regards to the remaining strategies in the comparison. The rationale for adopting the Wilcoxon Rank-Sum stemmed from the fact that the obtained results are not normally distributed, thus, rendering the need for a non-parametric test.

The null hypothesis (H_0) is that there is no significant difference as far as the test size is concerned for FIS and each individual strategy (i.e. the two populations have the same medians). Our alternative hypothesis (H_1) is that test size for FIS is less than that of each individual strategy (i.e. FIS has a lower population median).

To control the Type I - family wise error rate (FWER) owing to multiple comparisons, we have adopted the Bonferroni-Holm correction for adjusting α value (i.e. based on Holm's sequentially rejective step down procedure [26]). To be specific, 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:

$$\alpha_{Holm} = \frac{\alpha}{k - i + 1} \qquad (17)$$

where *k* is the total number of paired samples and *i* signifies the test number.

If $p_1 < \alpha_{Holm}$, the corresponding hypothesis is rejected and we are allowed to make a 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 10–16.

Table 10. Wilcoxon Rank-Sum Tests for Table 3

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction α_{Holm} with 95% confidence level	Bonferroni-Holm Correction α_{Holm} with 90% confidence level
FIS vs EMCQ	0.014	α_{Holm} =0.016667, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.033333, p-value < α_{Holm}, Reject H_o
FIS vs CF	0.014	α_{Holm} =0.025, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.05, p-value < α_{Holm}, Reject H_o
FIS vs ISR	0.376	α_{Holm} =0.05, p-value > α_{Holm}, Cannot reject H_o	α_{Holm} =0.10, p-value > α_{Holm}, Cannot reject H_o

Table 11. Wilcoxon Rank-Sum Tests for Table 4

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction α_{Holm} with 95% confidence level	Bonferroni-Holm Correction α_{Holm} with 90% confidence level
FIS vs PSTG	0.0035	α_{Holm} =0.01, p-value < α_{Holm} , Reject H_o	α_{Holm} =0.02, p-value < α_{Holm} , Reject H_o
FIS vs CF	0.004	α_{Holm} =0.0125, p-value < α_{Holm} , Reject H_o	α_{Holm} =0.04, p-value < α_{Holm} , Reject H_o
FIS vs ISR	0.004	α_{Holm} =0.016667, p-value < α_{Holm} , Reject H_o	α_{Holm} =0.06, p-value < α_{Holm} , Reject H_o
FIS vs CS	0.0045	α_{Holm} =0.025, p-value < α_{Holm} , Reject H_o	α_{Holm} =0.08, p-value < α_{Holm} , Reject H_o
FIS vs EMCQ	0.0125	α_{Holm} =0.05, p-value < α_{Holm} , Reject H_o	α_{Holm} =0.10, p-value < α_{Holm} , Reject H_o

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of DPSO and APSO are ignored

Table 12. Wilcoxon Rank-Sum Tests for Table 5

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction α_{Holm} with 95% confidence level	Bonferroni-Holm Correction α_{Holm} with 90% confidence level
FIS vs PSTG	0.004	α_{Holm} =0.01, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.02, p-value < α_{Holm}, Reject H_o
FIS vs CF	0.006	α_{Holm} =0.0125, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.04, p-value < α_{Holm}, Reject H_o
FIS vs EMCQ	0.0105	α_{Holm} =0.016667, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.06, p-value < α_{Holm}, Reject H_o
FIS vs ISR	0.0105	α_{Holm} =0.025, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.08, p-value < α_{Holm}, Reject H_o
FIS vs CS	0.025	α_{Holm} =0.05, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.1, p-value < α_{Holm}, Reject H_o

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of DPSO and APSO are ignored

Table 13. Wilcoxon Rank-Sum Tests for Table 6

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction α_{Holm} with 95% confidence level	Bonferroni-Holm Correction α_{Holm} with 90% confidence level
FIS vs PSTG	0.006	α_{Holm} =0.01, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.02, p-value < α_{Holm}, Reject H_o
FIS vs CS	0.006	α_{Holm} =0.0125, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.04, p-value < α_{Holm}, Reject H_o
FIS vs CF	0.0125	α_{Holm} =0.016667, p-value < α_{Holm}, Reject H_o	α_{Holm} =0.06, p-value < α_{Holm}, Reject H_o
FIS vs ISR	0.025	α_{Holm} =0.0125, p-value > α_{Holm}, Cannot Reject H_o	α_{Holm} =0.08, p-value < α_{Holm}, Reject H_o
FIS vs EMCQ	0.242	Cannot Reject H_o	α_{Holm} =0.1, p-value > α_{Holm}, Cannot Reject H_o

*Owing to incomplete sample (i.e. with one or more NA entries), the contributions of DPSO and APSO are ignored

Table 14. Wilcoxon Rank-Sum Tests for Table 7

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction α_{Holm} with 95% confidence level	Bonferroni-Holm Correction α_{Holm} with 90% confidence level
FIS vs EMCQ	0.021	α_{Holm} =0.0125, p-value > α_{Holm}, Cannot Reject H_o	α_{Holm} =0.025, p-value < α_{Holm}, Reject H_o
FIS vs CF	0.0215	Cannot Reject H_o	α_{Holm} =0.033333, p-value < α_{Holm}, Reject H_o
FIS vs ISR	0.0215	Cannot Reject H_o	α_{Holm} =0.05, p-value < α_{Holm}, Reject H_o
FIS vs DPSO	0.072	Cannot Reject H_o	α_{Holm} =0.1, p-value < α_{Holm}, Reject H_o

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

Table 15. Wilcoxon Rank-Sum Tests for Table 8

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction α_{Holm} with 95% confidence level	Bonferroni-Holm Correction α_{Holm} with 90% confidence level
FIS vs CF	0.0135	α_{Holm} =0.0125, p-value > α_{Holm}, Cannot Reject H_o	α_{Holm} =0.025, p-value < α_{Holm}, Reject H_o
FIS vs EMCQ	0.014	Cannot Reject H_o	α_{Holm} =0.033333, p-value < α_{Holm}, Reject H_o
FIS vs ISR	0.014	Cannot Reject H_o	α_{Holm} =0.05, p-value < α_{Holm}, Reject H_o
FIS vs DPSO	0.046	Cannot Reject H_o	α_{Holm} =0.1, p-value < α_{Holm}, Reject H_o

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

Table 16. Wilcoxon Rank-Sum Tests for Table 9

Pair Comparison	p-value in ascending order	Bonferroni-Holm Correction: α_{Holm} with 95% confidence level	Bonferroni-Holm Correction: α_{Holm} with 90% confidence level
FIS vs EMCQ	0.0135	α_{Holm} =0.0125, p-value > α_{Holm}, Cannot reject H_o	α_{Holm} =0.025, p-value < α_{Holm}, Reject H_o
FIS vs ISR	0.0215	Cannot reject H_o	α_{Holm} =0.033333, p-value < α_{Holm}, Reject H_o
FIS vs DPSO	0.058	Cannot reject H_o	α_{Holm} =0.05, p-value < α_{Holm}, Reject H_o
FIS vs CF	0.0865	Cannot reject H_o	α_{Holm} =0.1, p-value < α_{Holm}, Reject H_o

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

5. Experimental Observation

Reflecting on the work undertaken, a number of observations can be elaborated based on the results obtained from each experiment as well as the corresponding statistical analysis.

Concerning the first part of the experiments elaborated in Section 4.1, Table 3 depicts the comparative performances of hyper-heuristics (and their selection and acceptance mechanism) amongst themselves. Here, we can observe that FIS dominates as far as getting the best average test sizes with 60.67% (i.e. 4 out of 6 entries) as compared to its other hyper-heuristics counterparts. ISR comes as the runner up with 33.33% (i.e. outperforming FIS in 2 entries). As for the best test size, FIS also manages to get best results with a percentage of 83.33% (i.e. 5 out of 6 entries). In the case of CA_3*(N; 3, 3^6)* and CA_6*(N; 3, $5^2 4^2 3^2$)*, it is interesting to note that although all hyper-heuristics share the best test size, ISR gives the best overall average. Conversely, in the case of involving CA_4 *(N; 3, 6^6)*, we can note that the strategy (i.e. ISR) that obtain the best test size will not necessarily produce the best test size averages. Concerning the average execution time, we observe that EMCQ and CF outperform other hyper-heuristic strategies with 50.00% (i.e. 3 out of 6 entries). ISR and FIS perform the poorest in all cases observed.

The boxplot analysis of Table 3 in Figure 12 (a) to (f) reveals a number of salient characteristic and patterns of EMCQ, CF, ISR and FIS searching process. Considering CA_1 (*N*; *2*, 3^{13}), the distribution of box plot results is symmetric and the range of results is similar for all hyper-heuristics (i.e. similar bottom and top whiskers). CF and FIS have smaller interquartile range as compared to EMCQ and ISR. In this case, FIS has lowest median. As far as CA_2 (*N*; *2*, 10^{10}) is concerned, the distribution of box plot results is symmetric for CF, ISR and FIS but not for EMCQ. CF has large range of results (i.e. large top and bottom whiskers) and its median is far off the other hyper-heuristics medians (i.e. with EMCQ having the lowest median). In CA_3 *(N; 3, 3^6)*, the distribution of box plot results is asymmetric for all hyper-heuristics. Furthermore, all hyper-heuristics appear to have a large range of results with EMCQ and CF reaching the extreme top. The interquartile range is similar for all hyper-heuristics with ISR having the lowest median. Concerning CA_4 *(N; 3, 6^6)*, the distribution of box plot results is asymmetric for all hyper-heuristics. CF and EMCQ have the largest range of results as compared to other hyper-heuristics. CF has the highest interquartile range. Here, FIS and ISR share the same lowest median. Similar to CA_4 *(N; 3, 6^6)*, the distribution of results is also asymmetric for CA_5 *(N; 3, 10^6)*. CF has the largest range of results with EMCQ comes in as the runner up. The lowest median is shared between ISR and FIS. In CA_6 *(N; 3, $5^2 4^2 3^2$)*, the distribution of box plot results is again asymmetric for all hyper-heuristics. CF has the largest range of results with EMCQ the runner up. ISR has the largest interquartile range yet with the smallest median.

Statistical analysis of Table 3 (given in Table 10) indicates that two null hypothesis are rejected for both 95% and 90% confidence levels. FIS is statistically better than MC and CF. However, there is no difference as far as the performance of FIS with ISR is concerned.

Referring to pie chart representations in Figure 13(a) to (d) for CA_1–CA_6 in Table 3, we note that CF has more preference towards the Jaya search operators (with 37.99%). EMCQ and ISR appear to favor global pollination search operator (with 25.35% and 25.2% respectively) whilst FIS favors both crossover and peer learning search operators (with 25.05%).

In the final part of the experiments highlighted in Section 4.3, benchmarking results highlight the overall comparative performances of our implemented hyper-heuristics. Unlike Table 3, the scope of comparison for the rest of the Tables (i.e. Tables 4–9) is extended to include comparison with both meta-heuristic and hyper-heuristic based strategies (although without average time comparison). Table 4 demonstrates that FIS outperforms all other strategies as far as the best average test size with 90% (i.e. 9 out of 10 entries). The other 10% (1 out of 10) of the best average test size has been produced by APSO. Concerning the best test size, hyper-heuristic strategies generally outperform the meta-heuristic counterparts. Each hyper-heuristic has a nearly fair share of the best test size entries (i.e. FIS with 90%, EMCQ with 70%, ISR with 60%, and CF with 50%). The closest rival from the meta-heuristic strategy comes from DPSO (with 60%), APSO (with 50%) and CS (with 50%). PSTG gives the poorest performance with 30%.

Statistical analysis of Table 4 (given in Table 11) favors the alternate hypothesis for both 95% and 90% confidence levels indicating that FIS has better performance than PSTG, CF, ISR, CS and EMCQ (whilst ignoring the contributions of DPSO and APSO).

Concerning pie chart representations in Figure 14(a) to (d) CA (N; 2, 3^k) in Table 4, we again see that CF has more preference towards the Jaya search operators (with 34.78%). As for other hyper-heuristics, EMCQ favors the global pollination search operator (with 25.11%) whilst ISR favors the Jaya search operator (with 25.43%) and FIS favors the peer learning search operator (with 26.83%).

In Table 5, we observe FIS gives the best average test size with 88.88% (i.e. 8 out of 9 entries). The other best average test size entries are shared amongst CS with 11.11% (i.e. 1 out of 9 entries), EMCQ with 11.11% (i.e. 1 out of 9 entries) and DPSO with 11.11% (i.e. 1 out of 9 entries) respectively. As far as the best test size is concerned, hyper-heuristics are outperforming the meta-heuristic ones. DPSO and APSO are two of the meta-heuristic strategies that produce commendable results with the hyper-heuristics counterparts. It is interesting to note that while APSO gives the best test size for CA *(N; 3, 3^8)*, it is EMCQ and ISR that give the best average test size.

Statistical analysis of Table 5 (given in Table 12) favors the alternate hypothesis for both 95% and 90% confidence levels. FIS has statistically better performance than PSTG, CF, EMCQ, ISR and CS (i.e. ignoring the contributions of DPSO and APSO).

Pie chart representations in Figure 15(a) to (d) for CA (N; 3, 3^k) in Table 5, we observe similar findings for CF as in the two cases earlier (i.e. refer to Figure 13 and 14). We note that CF has more preference towards the Jaya search operators (with 31.31%). EMCQ favors the peer learning search operator (with 25.07%) as do FIS (with 25.50%). ISR appears to favor the global pollination search operator (with 25.08%).

Concerning Table 6, FIS offers the best average test size with 50% (i.e. 4 out of 8 entries) followed by EMCQ with 25% (i.e. 2 out of 8), CF with 12.50% (i.e. 1 out of 8 entries), and ISR with 12.50% (i.e. 1 out of 8 entries). As for the best test size, FIS obtains 75% (i.e. 6 out of 8 entries). ISR comes in with 62.50% (i.e. 5 out of 8 entries), CF with 50% (i.e. 4 out of 8 entries), EMCQ and DPSO with 25% (i.e. 2 out of 8 entries), and APSO with 12.50% (i.e. 1 out of 8 entries). In this case, PSO and CS perform the poorest for both average and best test size. It is interesting to note that although not getting the best average (i.e. best average is obtained by FIS), CF has produced the best test size for *CA (N; 4, 3^{10}).*

Statistical analysis of Table 6 (given in Table 13) gives mixed results at 95% and 90% confidence levels. At 95% confidence level, FIS has statistically better performance as compared to PSTG, CS, and CF (i.e. ignoring the contributions of DPSO and APSO). However, the performance of FIS is not statistically better than ISR and MC. At 90% confidence level, the statistical analyses are all in favor of the alternate hypothesis with the exception of MC. Hence, with 90% confidence level, FIS is better than other strategies with the exception of MC (i.e. ignoring the contributions of DPSO and APSO).

Referring to pie chart representations in Figure 16(a) to (d) for *CA (N; 4, 3^k)* in Table 6 , we again observe similar findings as in all earlier cases for CF. We also note that the preference towards the Jaya search operator is 28.27% (i.e. slightly less than 30%). In the three earlier cases, the percentages are all above 30% (with 37.99%, 37.78%, and 31.31% respectively). EMCQ favors the crossover search operator (with 25.13%) whilst ISR favors the global pollination search operator (with 25.04%) and FIS favors the Jaya search operator (with 25.41%).

In Table 7, the overall performance for average and best test size is scattered amongst meta-heuristic and hyper-heuristic based strategies. FIS has the best performance as far as best average test size with 66.67% (i.e. 4 out of 6 entries). DPSO is the runner up with 33.33% (i.e. 2 out of 6 entries). APSO comes third with 16.67% (i.e. 1 out of 6 entries). CS, PSTG, EMCQ, CF and ISR perform the poorest. Concerning the best test size, DPSO, ISR and FIS share the percentage of 66.67% (i.e. 4 out of 6 entries). EMCQ and CF also share the percentage of 33.33% (i.e. with 2 out of 6 entries). The rest of the strategies (PSTG, APSO, and CS) are at 16.67% (i.e. 1 out of 6 entries).

Statistical analysis of Table 7 (given in Table 14) gives two different indications. At 95% confidence level, the statistic is in favor of the null hypothesis indicating that FIS has similar performance with all other strategies (i.e. ignoring the contribution of PSTG, APSO and CS). Nonetheless, at 90% confidence level, FIS has statistically significant performance as compared to EMCQ, CF, ISR and DPSO (i.e. ignoring the contribution of PSTG, APSO and CS).

Pie chart representations in Figure 17(a) to (d) for CA(*N; 2, v^7)*, CF again prefers Jaya search operators with 30.74%. EMCQ also favors the Jaya search operator (with 25.07%) whilst ISR and FIS favor the peer learning search operator (with 25.07% and 25.91% respectively).

In Table 8, FIS outperforms all other strategies as far as the average test size is concerned with 66.67% (i.e. 4 out of 6 entries). The other best average test size entry is shared by CS and DPSO with 16.67% (i.e. 1 out of 6 entries). Concerning the best test size, FIS also outperforms all other strategies with 66.67% (i.e. 3 out of 6 entries). DPSO comes in as the runner up with 50% (i.e. 3 out of 6 entries) followed by CF and EMCQ with 33.33% (i.e. 2 out of 6 entries). Meanwhile, APSO, CS, and EMCQ have the same percentage of 16.67% (i.e. 1 out of 6 entries).

Statistical analysis of Table 8 (given in Table 15) favors the null hypothesis at 95% confidence level but not at 90% confidence level. Thus, the performance of FIS is only statistically better than CF, EMCQ, ISR, and DPSO at 90% confidence level (i.e. ignoring the contribution of PSTG, APSO and CS).

Concerning pie chart representations in Figure 18(a) to (d) for CA (*N; 3, v^7)* in Table 8, we also confirm that Jaya is more preferred as far as CF is concerned with 27.62%. EMCQ favors crossover search operator (with 25.06%). ISR appears to favor both the Jaya and the peer learning search operators (with 24.99%). FIS favors the peer learning search operator (with 25.65%).

In Table 9, DPSO dominates as far as the best average test size is concerned with 50% (i.e. 3 out of 6 entries). FIS comes in as the runner up with 33.33% (i.e. 2 out of 6) followed by APSO comes with 16.67% (i.e. 1 out of 6 entries). Similar pattern can also be seen in the case of best test size. DPSO dominates with 75% (i.e. 5 out of 6 entries). FIS comes in as the runner up with 33.33% (i.e. 2 out of 6) whilst APSO comes third with 16.67% (i.e. 1 out of 6 entries). Putting DPSO, FIS and APSO aside, no other strategies are able to register the best average test size as well as the best test size.

Statistical analysis of Table 9 (given in Table 16) gives an indication that the null hypothesis can only be rejected at 90% confidence level but not at 95% confidence level. Hence, the performance of FIS is statistically better than EMCQ, ISR, DPSO and CF at 90% confidence level.

Referring to pie chart representations in Figure 19(a) to (d), our observations are still in support of our earlier findings as far as CF is concerned. Jaya is again more preferred in the case of CF with 26.75%. EMCQ favors the peer learning search operator (with 25.06%) whilst ISR favors the crossover search operator (with 25.06%)). Finally, FIS favors the crossover search operator (with 26.07%).

6. Concluding Remark

In this paper, we have described an experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial *t-way* test suite generation. Additionally, we have proposed a new hyper-heuristic selection acceptance mechanism, called FIS, based on the fuzzy inference system and compared with other hyper-heuristic and meta-heuristic approaches.

In terms of overall performance, hyper-heuristic based strategies appear to be more superior as compared to the meta-heuristic counterparts (as evidenced by the given results in Section 5). In fact, in terms of best test suite size, hyper-heuristic based strategies often produce good results for most cases. As highlighted in the early sections of this paper, a hyper-heuristic can be viewed as a form of hybridization in the sense that more than one search operator can work together in synergy. Hyper-heuristics, when properly designed, can potentially outperform meta-heuristics owing to three reasons. Firstly, unlike most meta-heuristics, hyper-heuristics are not subjected to special tuning. In fact, in our case, we only have to deal with only population size and maximum iteration. Secondly, hyper-heuristics allow adaptive decision (and learning) to decide on the best search operator at hand. Finally, hyper-heuristics allow a flexible and "plug-and-play" approach of search operators from different meta-heuristics allowing for more search diversity of solutions.

Concerning the individual performance, FIS, generally outperforms most other strategies as far as obtaining the best average test sizes are concerned. As far as statistical analysis is concerned, FIS has statistically better performance than other strategies at 90% confidence level. However, at 95% confidence level, there is no sufficient statistical evidence that suggests the superiority of FIS as compared to other strategies. This finding is an indication that most (meta-heuristic) strategies are already capable of reaching the known best results.

To the best of our knowledge, FIS is the first fuzzy based hyper-heuristic heuristic strategy that addresses the problem of *t-way* test suite generation. The main feature of FIS is that it enhances the rules of its predecessor ISR allowing multiple degrees of membership. Like ISR, FIS relies on three operators' measures to decide on whether to maintain or change a particular running search operator. Unlike ISR where the decision is strictly based on the previous operator measures, the FIS decision is based on the weighted Fuzzy inference rules. The net effect is that FIS allows smoother transition between search operators allowing consistent best test suite generation (as evidenced by many of the best average test suite sizes).

On a negative note, in terms of average execution time, FIS and its predecessor ISR appears to be slower than EMCQ and CF. Unlike EMCQ and CF, both FIS and ISR require extra overheads to undertake the Hamming distance measure for the improvement, diversification and intensification operator.

As far as the percentage distribution of search operators is concerned, one glaring observation can be elaborated further. With the exception of the CF, the search operator distributions are nearly fair for most hyper-heuristic selection and acceptance mechanisms (indicating that the decision by each mechanism is adaptive and dynamic in nature). At a glance, apart from CF, FIS appears to favor peer learning search operators. However, a closer look reveals that the delta range is too small with the highest value of 26.83% (in Figure 14). Unlike FIS, CF demonstrates a clear preference toward the Jaya operator with all cases (reaching 30% in three cases; 37.99% in Figure 13, 34.78% in Figure 14 and 30.74% in Figure 17). Our observation indicates that such preferences arise owing to the simplicity of the Jaya search operator. As the implementation of its operator is based on simple arithmetic differences (as compared to complex arithmetic for crossover, peer learning, and global pollination), the Jaya code tends to run faster. CF, unlike EMCQ, ISR, and FIS, gives more reward to the search operator than runs faster. For this reason, the Jaya search operator will naturally have more opportunity for selection. For other hyper-heuristic selection and acceptance mechanisms, execution time is not the parameter for making decision on whether to maintain or change the current search operators. Furthermore, having given too many opportunities for the Jaya search operator (even when it is not giving the good quality solution) causes a large swing of values (from low to high) for CF in the box plot analysis given in Figure 12.

The time complexity analysis for EMCQ, CF, ISR and FIS can be analyzed by considering the combined structures of all the hyper-heuristic algorithms and its four defined operators as described in Section 3. The structures for EMCQ and CF as well as ISR and FIS are shown in Figure 20. Assuming all other operations can be performed in a constant time, the time complexity for EMCQ and CF is $O(J\text{x}K\text{x}L) \approx O(n^3)$ when J, K, and L are approaching large n. In a similar manner, the time complexity for ISR and FIS is $O(J\text{x}K\text{x}(L+M)) \approx \text{O}(n^3)$ when J, K, L+M are approaching large n.

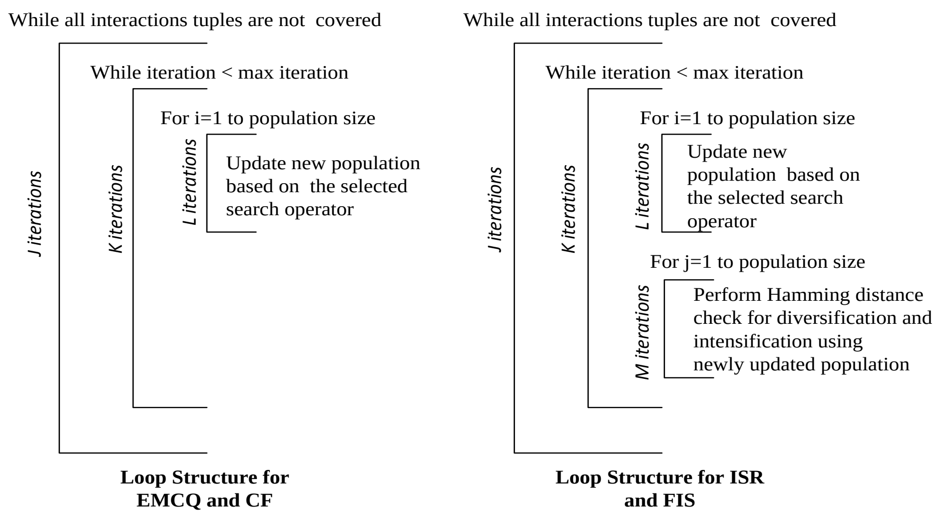

While all interactions tuples are not covered
While iteration < max iteration
For i=1 to population size
Update new population based on the selected search operator
J iterations
K iterations
L iterations
Loop Structure for EMCQ and CF
While all interactions tuples are not covered
While iteration < max iteration
For i=1 to population size
Update new population based on the selected search operator
For j=1 to population size
Perform Hamming distance check for diversification and intensification using newly updated population
M iterations
Loop Structure for ISR and FIS

Figure 20. General Structures for EMCQ, CF, ISR and FIS

As our findings have been encouraging, we are planning to benchmark our work further. In fact, we have summarized the application of FIS for test redundancy reduction problem as the supplementary material for the current paper (refer to [50]). Apart from its application for test redundancy reduction problem, another useful avenue is to consider the benchmark of FIS within the HyFlex framework [1] (since the framework captures most if not all the state-of-the-art on hyper-heuristic). Ideally, within the HyFlex framework, the performance of FIS for general optimization problems can be objectively evaluated.

Currently, we are also interested in adopting FIS for testing cloud and service-oriented architecture (SOA) solutions. Given the potentially large interaction of components within an overall integrated solution, a strategy such as FIS can be useful to systematically minimize the test data for testing considerations (i.e., based on the given interaction strength) and give indication of the developed solution's quality.

Additionally, we are also looking into adopting Case based Reasoning (CBR) for search operator selection and acceptance mechanism. The current fuzzy rules can be our initial starting point. The output of the search operators can be clustered accordingly based on some dynamic centroid approach (i.e. using K-means or Fuzzy C-Mean clustering algorithm). With this approach, we could also investigate the ensembles of new search operators.

Finally, adaptation of FIS for dynamic multi-objective problems with multi-population models could be another avenue for future work. In this respect, FIS needs to allow effective information sharing between populations and proper adaptation to tackle changes in the functional landscapes.

Acknowledgement

The work reported in this paper is funded by the Science Fund Grant for the project titled: Constraints T-Way Testing Strategy with Modified Condition/Decision Coverage from the Ministry of Science, Technology, and Innovation (MOSTI), Malaysia.

References

1. *HypeFlex Framework: Cross-Domain Heuristic Search Challenge, last accessed on 15 September, 2016.* Available from: http://www.asap.cs.nott.ac.uk/external/chesc2011.

2. Ahmed, B.S., T.S. Abdulsamad, and M. Potrus, *Achievement of Minimized Combinatorial Test Suite for Configuration-Aware Software Functional Testing using the Cuckoo Search Algorithm.* Information and Software Technology, 2015. **66**: p. 13-29.
3. Ahmed, B.S. and K.Z. Zamli, *A Variable-Strength Interaction Test Suites Generation Strategy Using Particle Swarm Optimization.* Journal of Systems and Software, 2011. **84**(12): p. 2171-2185.
4. Ahmed, B.S., K.Z. Zamli, and C.P. Lim, *Application of Particle Swarm Optimization to Uniform and Variable Strength Covering Array Construction.* Applied Soft Computing, 2012. **12**(4): p. 1330-1347.
5. Alazzam, A. and H. W. Lewis III, *A New Optimization Algorithm For Combinatorial Problems.* International Journal of Advanced Research in Artificial Intelligence, 2013. **2**(5): p. 63-68.
6. Alsewari, A.R.A. and K.Z. Zamli, *Design and Implementation of a Harmony Search based Variable Strength t-way Testing Strategy with Constraint Support.* Information Software Technology, 2012. **54**: p. 553-568.
7. Asmuni, H., E.K. Burke, and J.M. Garibaldi. *Fuzzy Multiple Heuristic Ordering for Course Timetabling*. in *Proceedings of the 5th United Kingdom workshop on computational intelligence (UKCI 2005)*. 2005. Citeseer.
8. Asta, S., et al., *Combining Monte-Carlo and Hyper-Heuristic Methods for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem.* Information Sciences, 2016. **373**: p. 476-498.
9. Ayob, M. 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.
10. Burke, E.K., et al., *Hyper Heuristics: A Survey of the State of the Art.* Journal of the Operational Research Society, 2013. **64**: p. 1695–1724
11. Burke, E.K., G. Kendall, and E. Soubeiga, *A Tabu-Search Hyperheuristic for Timetabling and Rostering.* Journal of Heuristics, 2003. **9**(6): p. 451-470.
12. Camastra, F., et al., *A Fuzzy Decision System for Genetically Modified Plant Environmental Risk Assessment using Mamdani Inference.* Expert Systems with Applications, 2015. **42**(3): p. 1710-1716.
13. Cano-Belmán, J., R.Z. Ríos-Mercado, and J. Bautista, *A Scatter Search based Hyper-Heuristic for Sequencing a Mixed-Model Assembly Line.* Journal of Heuristics, 2010. **16**(6): p. 749-770.
14. Chen, X., et al. *Variable Strength Interaction Testing with an Ant Colony System Approach*. in *2009 16th Asia-Pacific Software Engineering Conference*. 2009.
15. Cohen, D.M., et al., *The AETG System: An Approach to Testing based on Combinatorial Design.* IEEE Transactions on Software Engineering, 1997. **23**(7): p. 437–443.
16. Cohen, M.B., *Designing Test Suites for Software Interaction Testing*, in *Department of Computer Science*. 2004, University of Auckland: New Zealand. p. 185.
17. Cohen, M.B., C.J. Colbourn, and A.C.H. Ling, *Constructing Strength Three Covering Arrays with Augmented Annealing.* Discrete Mathematics, 2008. **308**(13): p. 2709-2722.
18. Cohen, M.B., 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*. 2007. London, UK: ACM.
19. Cordón, O., *A Historical Review of Evolutionary Learning Methods for Mamdani-type Fuzzy Rule-based Systems: Designing Interpretable Genetic Fuzzy Systems.* International Journal of Approximate Reasoning, 2011. **52**(6): p. 894-913.
20. Drake, J.H., E. Özcan, and E.K. Burke. *A Modified Choice Function Hyper-Heuristic Controlling Unary and Binary Operators*. in *Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC)*. 2015. IEEE.
21. Garvin, B.J., M.B. Cohen, and M.B. Dwyer, *Evaluating Improvements to a Meta-Heuristic Search for Constrained Interaction Testing.* Empirical Software Engineering, 2011(16): p. 61-102.
22. Ghanem, T.F., W.S. Elkilani, and H. M.Abdul-Kader, *A Hybrid Approach for Efficient Anomaly Detection Using Meta-heuristic Methods.* Journal of Advanced Research, 2015. **6**(4): p. 609-619.
23. Gudino-Penaloza, F., et al. *Fuzzy Hyperheuristic Framework for GA Parameters Tuning*. in *Artificial Intelligence (MICAI), 2013 12th Mexican International Conference on*. 2013. IEEE.
24. Harman, M. and B.F. Jones, *Search-Based Software Engineering.* Information and Software Technology, 2001. **43**(14): p. 833-839.
25. Holland, J.H., *Adaptation in Natural and Artificial Systems* 1975: University of Michigan Press.
26. Holm, S., *A Simple Sequentially Rejective Multiple Test Procedure.* Scandinavian Journal of Statistics, 1979. **6**: p. 65-70.
27. Jia, Y., et al. *Learning Combinatorial Interaction Test Generation Strategies Using Hyperheuristic Search*. in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. 2015.
28. Kendall, G., P. Cowling, and E. Soubeiga. *Choice Function and Random Hyperheuristics*. in *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*. 2002.
29. Kennedy, J. and R. Eberhart. *Particle Swarm Optimization*. in *Proceedings of the IEEE International Conference on Neural Networks*. 1995.
30. Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, *Optimization by Simulated Annealing.* Science, 1983. **220**(4598): p. 671-680.

31. Lei, Y. 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)*. 1998. Washington, DC, USA: IEEE Computer Society.
32. Mahmoud, T. and B.S. Ahmed, *An Efficient Strategy for Covering Array Construction with Fuzzy Logic-based Adaptive Swarm Optimization for Software Testing Use.* Expert Systems with Applications, 2015. **42**: p. 8753-8765.
33. McCaffrey, J. *An Empirical Study of Pairwise Test Set Generation Using a Genetic Algorithm*. in *Proceedings of the 7th International Conference on Information Technology*. 2010. IEEE Computer Society.
34. Mizumoto, M., *Fuzzy Controls under Various Fuzzy Reasoning Methods.* Information Sciences, 1988. **45**(2): p. 129-151.
35. Nie, C. and H. Leung, *A Survey of Combinatorial Testing.* ACM Computing Surveys, 2011. **43**(2).
36. Pedrycz, W. and X. Wang, *Designing Fuzzy Sets With the Use of the Parametric Principle of Justifiable Granularity.* IEEE Transactions on Fuzzy Systems, 2016. **24**(2): p. 489-496.
37. Press, W.H., et al., *Numerical Recipes in C- The Art of Scientific Computing*. 1992: Cambridge University Press.
38. Rao, R.V., *Jaya: A Simple and New Optimization Algorithm for Solving Constrained and Unconstrained Optimization Problems.* International Journal of Industrial Engineering Computations, 2016. **7**(1): p. 19-24.
39. Rao, R.V., V.J. Savsani, and D.P. Vakharia, *Teaching-Learning-based Optimization: A Novel Method for Constrained Mechanical Design Optimization Problems.* Computer Aided Design, 2011. **43**: p. 303-313.
40. Sabar, N.R. and G. Kendall, *Population based Monte Carlo Tree Search Hyper-Heuristic for Combinatorial Optimization Problems.* Information Sciences, 2015. **314**: p. 225-239.
41. Shiba, T., 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)*. 2004. Hong Kong: IEEE Computer Society.
42. Stevens, B. and E. Mendelsohn. *Efficient Software Testing Protocols*. in *Proceedings of the 8th IBM Centre for Advanced Studies Conference (CASCON '98)*. 1998. Toronto, Ontario, Canada: IBM Press.
43. Terashima-Marín, H., E. Flores-Alvarez, and P. Ross. *Hyper-Heuristics and Classifier Systems for Solving 2D-Regular Cutting Stock Problems*. in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 2005. ACM.
44. Wolpert, D.H. and W.G. Macready, *No Free Lunch Theorems for Optimization.* IEEE Transactions on Evolutionary Computation, 1997. **1**(1): p. 67-82.
45. Wu, H., et al., *A Discrete Particle Swarm Optimization for Covering Array Construction.* IEEE Transactions on Evolutionary Computation, 2015. **19**(4): p. 575-591.
46. Yang, X.-S., *Nature-Inspired Metaheursitic Algorithm*. 2010: Luniver Press.
47. Yang, X.-S. and S. Deb. *Cuckoo Search via Lévy Flight*. in *Proceedings of World Congress on Nature and Biologically Inspired Computing*. 2009. IEEE.
48. Yang, X.S., *Flower Pollination Algorithm for Global Optimization*, in *Unconventional Computation and Natural Computation, Lecture Notes in Computer Science*. 2012, Springer. p. 240-249.
49. Zamli, K.Z., B.Y. Alkazemi, and G. Kendall, *A Tabu Search Hyper-Heuristic Strategy for t-way Test Suite Generation.* Applied Soft Computing, 2016. **44**: p. 57-74.
50. Zamli, K.Z., et al. *Supplementary Material for the Information Sciences Paper: An Experimental Study of Hyper-Heuristic Selection and Acceptance Mechanism for Combinatorial t-way Test Suite Generation, last accessed on 16 February, 2017.*; Available from: http://arxiv.org/abs/1702.04501.