Azerbaijan Journal of High Performance Computing, Vol 1, Issue 1, 2018, pp. 87-112
https://doi.org/10.32010/26166127.2018.1.1.87.112

Multi-Start Jaya Algorithm for Software Module Clustering Problem

Kamal Z. Zamli[1], Abdulrahman Alsewari[2], Bestoun S. Ahmed[3]

[1] IBM Centre of Excellence, Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, Pahang, Malaysia, kamalz@ump.edu.my
[2] Faculty of Computer Systems and Software Engineering,Universiti Malaysia Pahang, Pahang, Malaysia, alsewari@ump.edu.my
[3] Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University, Prague, Czech Republic, albeybes@fel.cvut.cz

*Correspondence: Kamal Z. Zamli, IBM Centre of Excellence, Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, Pahang, Malaysia, kamalz@ump.edu.my

Abstract

Jaya algorithm has gained considerable attention lately due to its simplicity and requiring no control parameters (i.e. parameter free). Despite its potential, Jaya algorithm is inherently designed for single objective problems. Additionally, Jaya is limited by the intense conflict between exploration (i.e. roams the random search space at the global scale) and exploitation (i.e. neighborhood search by exploiting the current good solution). Thus, Jaya requires better control for exploitation and exploration in order to prevent premature convergence and avoid being trapped in local optima. Addressing these issues, this paper proposes a new multi-objective Jaya variant with a multi-start adaptive capability and Cuckoo search like elitism scheme, called MS-Jaya, to enhance its exploitation and exploration allowing good convergence while permitting more diverse solutions. To assess its performances, we adopt MS-Jaya for the software module clustering problem. Experimental results reveal that MS-Jaya exhibits competitive performances against the original Jaya and state-of-the-art parameter free meta-heuristic counterparts consisting of Teaching Learning based Optimization (TLBO), Global Neighborhood Algorithm (GNA), Symbiotic Optimization Search (SOS), and Sine Cosine Algorithm (SCA).

Keywords: Search based Software Engineering, Software Module Clustering Problem, Parameter Free Meta-Heuristic Algorithm, Jaya Algorithm, Computational Intelligence

1. Introduction

Solving complex multi-objective optimization problems can be painstakingly difficult endeavor considering multiple and conflicting design goals. A growing trend in utilizing meta-heuristic algorithms to solve these problems has been observed as they have shown considerable success in dealing with tradeoffs between conflicting design goals. Many multi-objective meta-heuristic algorithms have been developed in the past 25 years. Most of these algorithms have merits, but they require tuning of their specified control parameters. For example, Genetic Algorithm [1] require substantial tuning for population size, mutation and cross over rate. The same issue also arises in the case of Particle Swarm Optimization [2] which depends on population size, inertia weight, social and cognitive parameters as parameters. In similar manner, Harmony Search [3] requires tuning of harmony size, harmony memory consideration rate, and pitch adjustment. As for Ant Colony [4], the calibration of evaporation rate, pheromone influence, and heuristic influence are essential. Concerning Cuckoo Search [5], there is a need to tune the elitism probability. In many cases, improper tuning for all of these specific parameters undesirably increases computational efforts as well as yields sub-optimal solutions. As a result, many researchers have advocated the adoption of parameter free meta-heuristic algorithms [6-9].

Jaya algorithm [6] is a recently developed parameter free population based meta-heuristic algorithm. Despite its potential, Jaya algorithm is inherently designed for the single objective problem. To this end, there are two common methods for dealing with multi-objective problems within a meta-heuristic algorithm: aggregative and non-aggregative [10]. The former method combines all the objectives into one weighted function. This main drawback of this method is that only one solution is provided. To gain the benefit and tradeoffs of multiple sets of solutions, the method needs to be run multiple times. Another disadvantage of this method is that it is unable to deal with concave Pareto front in multi-objective problems[11]. Unlike the former method, the latter method optimizes all the objectives simultaneously. Here, a set of optimal solutions is formed, which has a better value for at least one objective (i.e. non-dominating), forming a Pareto front set. As such, this method provides the multiple tradeoffs (of Pareto-optimal) solutions (even in a single run) that can be chosen based on the order of importance of objectives. Thus, in this paper, we consider a non-aggregative method for our work.

Apart from the need to effectively deal with multi-objective problems, the current Jaya algorithm is limited by the intense conflict between exploration (i.e. roams the random search space at the global scale) and exploitation (i.e. neighborhood search by exploiting the current good solution). For this reason, Jaya requires better control for exploitation and exploration to prevent premature convergence and avoid being trapped in local optima.

Addressing the aforementioned issues, the contributions of this paper can be summarized as follows:

- A new Jaya algorithm variant, called MS-Jaya with multi-start adaptive capability and Cuckoo search like elitism scheme [5].
- Performance comparison of MS-Jaya with other state-of-the-art parameter free multi-objective algorithms (including the original Jaya [6], Teaching Learning based Optimization (TLBO)[7], Global Neighborhood Algorithm (GNA) [12], Symbiotic Optimization Search (SOS) [8], and Sine Cosine Algorithm (SCA)[9]) for software module clustering problem.

The paper is organized as follows. Section 2 presents the theoretical framework for software module clustering as a multi-objective optimization problem. Section 3 describes the related work on software module clustering. Section 4 highlights the original Jaya algorithm along with its known variants and the general design of MS-Jaya. Section 5 outlines our adaptation of MS-Jaya for software module clustering problem. Section 6 presents our benchmarking experiments. Section 7 discusses our experimental observations. Finally, Section 8 gives our concluding remarks along with the scope for future work.

2.Software Module Clustering Problem as Multi-Objective Optimization Problem

Software module clustering problem can be defined as the problem of partitioning modules into clusters based on some predefined quality criterion. Typically, the quality criterion for software module clustering problem relates to the concept of coupling and cohesion. Coupling is a measurement of dependency between module clusters whilst cohesion is the measurement of the internal strength of a module cluster. Thus, a good cluster distribution aids in functionality-cluster-module traceability provides easier navigation between sub-systems and enhances source code comprehension.

To evaluate the cluster distribution, a software system is usually represented as a Module Dependency Graph (MDG)[13]. The MDG is a directed graph in which modules are shown as nodes, dependencies are shown as edges, and clusters are partitions. Sometimes, weights are assigned to edges to denote the strength of the connection between the edge source and target nodes. For unweighted MDG, the weight is always set to 1. The coupling of a cluster can be calculated by summing the weight of external edges leaving or entering a cluster partition (termed inter-edges). Meanwhile, cohesion is calculated by summing the internal edges where the source and target modules belonging to the cluster partition (termed intra-edges). Combining coupling and cohesion, Mancoridis and Mitchell [13] (and later refined by Praditwong et al [14]) define Modularization Quality measure (MQ) as the sum of the ratio of intra-edges and inter-edges in each cluster, called Modularization Factor (MFk) for cluster k. MFk can be formally defined as follows:

$$MF_k = \begin{cases} 0 & if\ i = 0 \\ \dfrac{i}{i + \frac{1}{2}j} & if\ i > 0 \end{cases} \qquad \text{(Eq. 1)}$$

where i is the weight of intra-edges and j is that of inter-edges. The term 1/2j is to split the penalty of inter-edges across the two clusters that are connected by that edge. The MQ can then be calculated as the sum of MFk as follows:
when n is the number of clusters.

$$MQ = \sum_{k=1}^{n} MF_k \qquad \text{(Eq. 2)}$$

Hypothetical software comprising of 8 modules is depicted in Figure 1 to illustrate how the clustering of software modules works. In this case, the eight modules are clustered into two set of clusters. Given the clusters on the left, the MQ is 1.466 with n=2. On the other hand, given the clusters on the right, the MQ is 1.500 with also n=2. Based on a single objective of maximizing MQ, it can be said that the clusters on the right are better than those on the left. The challenge is to find the combination of clusters (i.e. from 2 to the number of modules -1) that would give the highest MQ value. It should be noted that the MQ measure is to find a balance between coupling and cohesion, but not to completely remove them. For instance, one can have only 1 cluster or n completely independent single module clusters to have zero coupling, but such an approach does not aid in functionality-cluster-module traceability or source code navigation and comprehension.

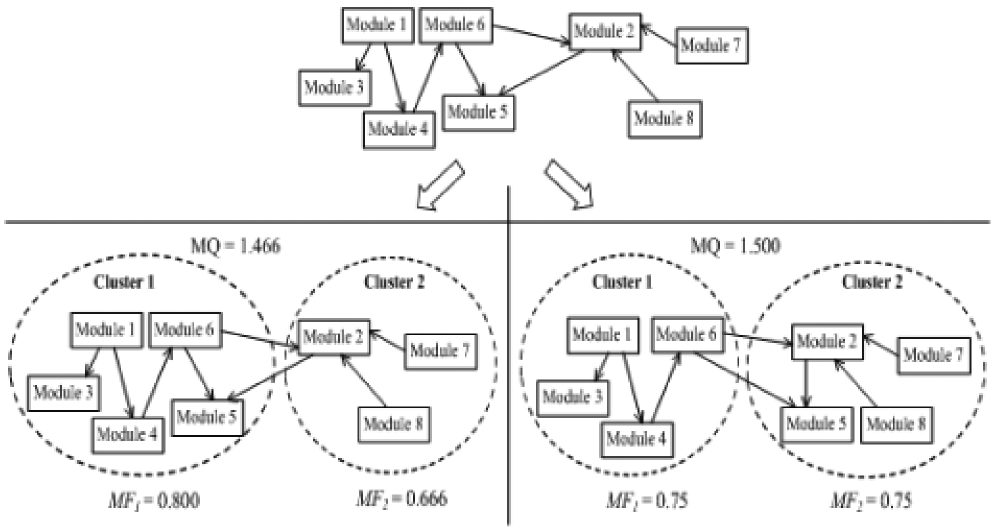


Figure 1. Software Module Clustering Problem

As maximizing cohesion and minimizing coupling are two conflicting objectives, software module clustering can be reformulated into multi-objective problem [14] as maximizing cluster approach (MCA) and equal-cluster approach (ECA). The objectives identified under MCA are:

Maximizing the sum of intra-edges of all clusters

Minimizing the sum of inter-edges of all clusters

Maximizing the number of clusters

Maximizing MQ

Minimizing the number of isolated clusters (i.e. getting as much clusters as possible).

The MCA approach focuses on achieving high cohesion and low coupling while maximizing the number of clusters.

Meanwhile, the objectives identified under ECA are:

Maximizing the sum of intra-edges of all clusters

Minimizing the sum of inter-edges of all clusters

Maximizing the number of clusters

Maximizing MQ

Minimizing the difference between the minimum and maximum number of modules in a cluster (i.e. getting equal size clusters).

The ECA approach also focuses on achieving high cohesion and low coupling. In doing so, the approach targets equal size clusters.

3. Related Work on Software Module Clustering

Software module clustering has been used in the literature within software reverse engineering to assess the software comprehension, evolution, and maintenance [15-17]. In particular, the interest on software module clustering problem has been advocated by recent emergence of the new field called Search based Software Engineering (SBSE)[18-21]. Much work has been done in the maintenance part to help to identify and group (i.e., cluster) those modules (sometimes called sub-systems) with common features. Evidence shows that modularized software could lead to better development and maintenance process [22, 23]. In fact, the non-proper and badly modularized code could be a strong potential reason for schedule deviation of maintenance and testing in the software project.

With the different complex functionalities and features that the modern software systems provide, the size of the system (concerning lines of code and module numbers) has been growing dramatically. This, in turn, leads to having dramatically many different arrangements (i.e., solutions) of the modules. At this stage, software module clustering has been considered as a graph-partitioning problem. The graph-partitioning problem, in fact, is an NP-hard problem [13, 14]). Hence, there is no exact method to solve this problem. To this end, different algorithms have been designed and implemented to find near optimal solution with reasonable execution time.

Some early attempts toward the solution followed the classical techniques for clustering such as K-means combined with greedy approach [24]. However, these attempts seem to be infeasible due to the poor exploration of all solutions that prevent the improvement of the current solution towards better solutions. Meta-heuristic based algorithms showed impressive results as compared to the classical approach due to their ability for exploration and exploitation towards a better solution. Hill climbing (HC) is one of the first algorithms used by Mancoridis and Mitchell for this purpose (i.e. based on a tool called Bunch [13]). The algorithm starts by a random partition (i.e., search space) of MDG. Then, the algorithm rearranges the module in different ways to find a better arrangement with higher MQ. Here, the better partition is always kept and the algorithm iterates and updates the partitions based on this better partition. The algorithm stops when better partition could not be found after several iterations. This procedure of work is almost the basic procedure for most of the meta-heuristic algorithms with the variation of the update mechanism of the search space.

HC has produced better results as compared to classical greedy clustering due to the randomness and better exploration and exploitation, in turn, permitting good quality solutions. HC has also undergone different improvements in other implementation later by Mahdavi et al [25]. In fact, HC produced superior results other concerning the solution quality and execution time. However, it suffers from the premature convergence that leads to local optima. Genetic algorithms (GA) have also been used for software module clustering [26, 27]. However, it suffers from the parameter tuning of the algorithm that in turn to produce poor results in different experiments.

In fact, these algorithms have treated the software module clustering as a single-objective optimization problem. Here, the low coupling and high cohesion demand are combined into one objective of MQ. However, as previously illustrated in Section 2, we can formulate the problem as a multi-objective optimization problem. Praditwong et al [13] has pioneered this approach to generate more robust results than the single objective by using Pareto optimality concept. This new multi-objective nature of the problem has motivated other researchers to implement hyper-heuristic based algorithms for module clustering. For instance, Kumari and Srivinas [28] has implemented a multi-objective hyper-heuristic Genetic Algorithm (MHypGA) for software module clustering. The research produced promising results as compared to other algorithms in term of quality of solutions and computational time.

More recently, Huang e al [29] has proposed a multi-agent evolutionary algorithm to solve this problem. The research adopts three evolutionary operators for the agents, namely competition, cooperation, and self-learning. The algorithm has been compared with two single-objective and two multi-objective algorithms. The algorithm outperforms the other algorithms in many cases. However, the algorithm needs more evaluation thought more case studies. In addition, the algorithm uses only MQ as the objective to solve, while there could be other factors that affect the quality towards even better results.

4. Introducing MS-Jaya

The following subsections highlight the original Jaya along with its existing variant as well as the new multi-start multi-objective adaptive version, MS-Jaya.

4.1. The Original Jaya

Let f(x) be the objective function to be minimized (or maximized). With f(x)$_{best}$ as the best solution and f(x)$_{worst}$ as the worst solution so far and $X_i^{(t)}$ is the value of ith variable (i.e. i=1,2…n), then the Jaya algorithm defines the next $X_i^{(t+1)}$ update as follows:

$$X_i^{(t+1)} = X_i^{(t)} + r_1(X_{best}\text{-}|X_i^{(t)}|)\text{-}\ r_2(X_{worst}\text{-}|X_i^{(t)}|) \qquad (\text{Eq. }3)$$

where X_{best} is the value of the variable for f(x)$_{best}$ and X_{worst} is the value of the variable for f(x)$_{worst}$ and $X_i^{(t+1)}$ is the updated ith value of $X_i^{(t)}$. Here, r_1 and r_2 are the two random scaling factor in the range [0,1]. The value $X_i^{(t+1)}$ is accepted only if it gives better objective function value than $X_i^{(t)}$. Figure 2 outlines the pseudo code for the Jaya algorithm.

```
Input: the population  X = {X1, X2… Xn}
Output: Xbest and the updated population X ´= {X1´, X2´… Xn´}
[1].    Begin
[2].  Initialize random populations X
[3].  Set initial $X_{best} = X_0$ and initial $X_{poor} = X_{best}$
[4].  Set population size, n
[5].  While (stopping criteria not met (i.e. t < T))
[6].      For population count, i =1 to population size, n
[7].          Generate random values of $r_1$ and $r_2$ between [0,1]
[8].          Update the current population using $X_i^{(t+1)}$ using Eq. 3
[9].          If $f(X_i^{(t+1)}) > f(X_i^{(t)})$ then    // assuming maximization problem
[10].                 $X_i^{(t)} = X_i^{(t+1)}$
[11].             If $f(X_i^{(t+1)}) > f(X_{best})$ then
[12].                 $X_{best} = X_i^{(t+1)}$
[13].             EndIf
[14].         Else
[15].             If $f(X_i^{(t)}) > f(X_{best})$ then
[16].                 $X_{best} = X_i^{(t)}$
[17].             EndIf
[18].             If $f(X_i^{(t)}) < f(X_{poor})$ then
[19].                 $X_{worst} = X_i^{(t)}$
[20].             EndIf
[21].         EndIf
[22].     EndFor
[23].  EndWhile
[24].  Return the updated population, X and the best result (Xbest)
[25]. End
```

Figure 2. Pseudo Code for Jaya Algorithm

As seen in line 10 in Figure 2, the Jaya algorithm exploits both X_{best} and X_{worst} as part of its transformation equation. Unlike other meta-heuristic algorithm, the Jaya algorithm uses only a single transformation equation to perform the search updates. Thus, the main strength of Jaya algorithm is its simplicity.

To allow exploration of the search space, the terms "$r_1(X_{best}\text{-}|X_i^{(t)}|)$- $r_2(X_{worst}\text{-}|X_i^{(t)}|)$" needs to be sufficiently large. Similarly, the terms "$r_1(X_{best}\text{-}|X_i^{(t)}|)$- $r_2(X_{worst}\text{-}|X_i^{(t)}|)$" must be small enough to allow steady exploitation. Analysing these aforementioned terms, the Jaya algorithm seems to provide poor control of exploration and exploitation. To be specific, when the difference between "$r_1(X_{best}\text{-}|X_i^{(t)}|)$- $r_2(X_{worst}\text{-}|X_i^{(t)}|)$" is small, the search process tends to get trapped in local optima hindering further exploration. In the current form, Jaya does not provide any mechanism to allow jumping out of local optima.

Additionally, although useful to ensure diversity updates, the improper random combination of the scaling factors r_1 and r_2 may unnecessarily make the Jaya algorithm wander back and forth (i.e. creating intense competition) between exploration and exploitation. This phenomenon can be counter-productive especially when the current search is converging.

Tackling these issues, the next sub-section reviews some multi-objective Jaya variants along with their applications followed by our proposed multi-objective MS-Jaya.

4.2. Review of Jaya Variants and their Applications

Since its inception in 2016, there are at least four known Jaya variants to date. Similar to other meta-heuristic variants [30], the main Jaya variants available in the literature can be divided into three categories: modified, hybrid-, and cooperative-based algorithms.

The modified-based category refers to Jaya variants that alter the original structure of the algorithm. Rao et al introduce the modified-based category called multi-objective Jaya (MO-Jaya) [31] integrating the NSGA-II algorithm. MO-Jaya has been successfully adopted to address multi-objective standard benchmark functions and solve well-known engineering problems involving wire-electric discharging, electro-chemical machining, and beam micro-milling process.

Indeed, the modified-based Jaya algorithm (e.g. MO-Jaya) produces sound results. The main issue for modified-based Jaya is to maintain it as parameter free algorithm. To be specific, introducing specific parameter controls (i.e. to manage exploration and exploitation) is not a feasible option as it would weaken the strength of Jaya (i.e. of being parameter free). For this reason, there is always an inherent limit on the modification that can be introduced.

The hybrid-based category refers to the integration of one or more meta-heuristic algorithms with Jaya. Zamli et al explore the hyper-heuristic algorithm based on Mamdani fuzzy approach (called Fuzzy Inference Selection (FIS)) [32] adopting Jaya with three other meta-heuristic algorithms (i.e. Teaching Learning based Optimization (TLBO), Flower Pollination Algorithm (FPA) and Genetic Algorithm (GA)). FIS has been adopted to solve the interaction testing problem.

Although useful for capitalizing Jaya strength and compensating its deficiencies, the main limitation of hybrid-based Jaya algorithm (e.g. FIS) is that its implementation is bulky and computationally heavy. In turn, this may be the limiting factor for its adoption in other optimization problems.

Finally, cooperative-based category refers to Jaya variants that adopt multi-swarm populations. Tasks are split in k sub-problems for simultaneous optimization before combining them as the final result. Rao and More [33] explore self-adaptive multi-population Jaya algorithm for optimal design of thermal devices. The population of Jaya can be dynamically modified during the searching process using the simple rule based selection. Later, Rao and Saroj [34] adopt similar approach to solve a series of engineering design problems involving welded beam, pressure vessels, tension spring compression and design speed reducer.

Given its potential, cooperative-based Jaya algorithm can be a useful approach for solving large NP hard optimization problem. However, there are two potential issues when dealing with cooperative-based approach. Firstly, deciding on the right level of task abstraction for k sub-problems can be problematic and often problem dependent. Secondly, coordination problem of tasks with its corresponding swarm may lead to unwarranted resource starvation causing deadlock or livelock situation.

4.3. The Design of MS-Jaya

Initially, the search process needs to roam through the search space in an effort to increase the probability of finding the best solution (i.e. exploration). Towards the end of the iteration, the search process needs to settle down and exploit the current best solution (i.e. exploitation). Ideally, when there is no improvement, the search should be able move to other location (so as to re-explore the search space with the hope to obtain better solution).

In the context of the current work, the MS-Jaya algorithm enhances the original Jaya algorithm in four aspects. Firstly, MS-Jaya changes the update $X_i^{(t+1)}$ by scaling the whole term of "$r_1(X_{best}\text{-}|X_i^{(t)}|)\text{-}\ r_2(X_{worst}\text{-}|X_i^{(t)}|)$" as follows:

$$X_i^{(t+1)} = X_i^{(t)} + \alpha\left[r_1\left(X_{best}\text{-}\left|X_i^{(t)}\right|\right)\text{-}\ r_2\left(X_{worst}\text{-}\ \left|X_i^{(t)}\right|\right)\right]\ (\text{Eq. }4)$$

where α is the actual scaling factor. Initially, the value α is large (i.e. MS-Jaya is exploring the search space). As the iteration progresses (i.e. MS-Jaya is exploiting), the value of α will be adaptively reduced as follows:

$$\alpha = M\left(1 - \frac{t}{T}\right) \qquad (\text{Eq. }5)$$

where t is the current iteration, T is the maximum number of iterations and M is the upper bound constant of the problem at hand.

Secondly, MS-Jaya allows multiple-start of the whole search process again by resetting the t value (similar to earlier work in [35]). In the case of MS-Jaya, resetting of t value makes the value of α large again and hence allowing Jaya to jump out of its current location (and permit re-exploration of the new search location). Potentially, multi-start approach in MS-Jaya increases the chance to obtain better solution within the new location in the search space.

Thirdly, MS-Jaya integrates with the Cuckoo search like elitism scheme [5] to ensure sufficient exploration of the search space. Here, a random fraction of the worst solutions is abandoned and replace with newly generated solutions.

Finally, MS-Jaya also incorporates the non-dominated approach to address multi-objective problem. For any two solutions, $X_i^{(t+1)}$ is said to dominate $X_i^{(t)}$ if these conditions hold:

- $X_i^{(t+1)}$ is not worse than $X_i^{(t)}$ in all objectives
- $X_i^{(t+1)}$ is strictly better than $X_i^{(t)}$ in at least one objective

Unlike single objective problem where the decision can be straightforwardly made (i.e. on either maximizing or minimizing from a sole objective), deciding on multi-objective best solution from the set of solutions is difficult. Utilizing the non-dominated approach, there can be potentially more than one conflicting solutions forming the Pareto front set. Based on the Pareto front set, the preferred Pareto optimal solution can be decided. In the case of MS-Jaya, we adopt the same approach in Pareto-Archived Evolution Strategy (PAES)[36] to generate the Pareto front set. MS-Jaya replaces the current solution with the new solution if the former is dominated by the latter or add the new solution to the Pareto front set (termed as PF archive) if it is dominated by no solution contained in the archive. Here, the purpose of the PF archive is to keep a historical record of the non-dominated solution found along the search process. At the start, the PF archive is empty and current (non-dominated) candidate solution is added to it. Then, the current (non-dominated) candidate solution found will be compared to the solution stored in the PF archive one-by-one in each iteration. If a particular candidate solution is dominated by one or more individuals in the PF archive, then that candidate solution must be discarded (i.e. remove from PF archive). Similarly, a particular candidate solution dominates any individual in the PF archive, that individual must be discarded.

Summing up, Figure 3 outlines the pseudo code for the multi-objective MS-Jaya algorithm.

```
    Input: the population X = {X_1, X_2... X_n}
    Output: the updated population X ´= {X_1´, X_2´... X_n´} and the Pareto Front archive,
PF = {X_p1, X_p2,... X_pn}
    [1].    Begin
    [2].  Initialize random populations , X
    [3].  Set initial $X_{best} = X_0$ and initial $X_{poor} = X_{best}$
    [4].  Set population size, n
    [5].  While (stopping criteria not met (i.e. t < T))
    [6].      Set threshold $\Delta =$ random between [0,1] × n
    [7].      For population count, i =1 to n
    [8].          If (non_improvement_count == $\Delta$) then
    [9].              t =1
    [10].             non_improvement_count =0
    [11].         EndIf
    [12].         Generate random values of $r_1$ and $r_2$ between [0,1]
    [13].         Calculate $\alpha$ using Eq. 5
    [14].         Update the current population using Eq. 4
    [15].         If ($X_i^{(t+1)}$) dominates ( $X_i^{(t)}$) then
    [16].                 $X_i^{(t)} = X_i^{(t+1)}$
    [17].                 If ($X_i^{(t+1)}$) dominates ($X_{best}$) then
    [18].                     $X_{best} = X_i^{(t+1)}$
    [19].                     Add $X_{best}$ into PF archive
    [20].                 EndIf
    [21].         Else
    [22].                non_improvement_count++;
    [23].                If ($X_i^{(t)}$) dominates ($X_{worst}$) then
    [24].                     $X_{worst} = X_i^{(t)}$
    [25].                 EndIf
    [26].         EndIf
    [27].         For i=1 to random[0,1] × n    // Cuckoo elitism scheme
    [28].               Find the worst, $X_i^{(t)}$
    [29].               Generate randomly $X_i^{(t+1)}$
    [30].               If ($X_i^{(t+1)}$) dominates ($X_i^{(t)}$) then
    [31].                   $X_i^{(t)} = X_i^{(t+1)}$
    [32].               EndIf
```

```
[32].           EndIf
[33].        EndFor
[34].          If $X_{best}$ is dominated by any member of existing {$X_{best\_1}$, $X_{best\_2}$,… $X_{best\_n}$} from the PF archive
[35].             Discard and remove $X_{best}$ from the PF archive
[36].          Else If $X_{best}$ dominates any member of existing {$X_{best\_1}$, $X_{best\_2}$,… $X_{best\_n}$} from the PF archive
[37].             Discard and remove $X_{best\_n}$ from the PF archive
[38].        EndIf
[39].     EndFor
[40].  EndWhile
[41].  Return the updated population, X. Report the PF archive and select the preferred solution
[42]. End
```

Figure 3. Pseudo Code for Multi-Objective MS-Jaya Algorithm

5. Adapting MS-Jaya for Software Module Clustering Problem

In order to adopt the MS-Jaya algorithm for software module clustering problem, there is a need to devise suitable solution representation. In the current work, the solution is initially represented by a set consisting of the random sequence of modules. To illustrate, consider Figure 4 depicting a random set {1,3,6,5,4,2,7,8} representing the 8 module MDG sequence (i.e. the sequence set can be changed by the MS-Jaya operator during the iteration process). To obtain the suitable number of clusters, the sequence set is to be partitioned accordingly. Here, the minimum cluster = 2 and the maximum cluster = 8/2 = 4 can be deduced. The rationale for the minimum cluster =2 is that there is a need for more than one cluster (i.e. no clustering is necessary if the minimum cluster = 1). With the maximum cluster = 4, a single module cluster can be avoided. For the sequence set {1,3,6,5,4,2,7,8}, a randomized partition is generated from 2 to 4 clusters as shown Figure 4a) till Figure 4b). In Figure 4a), the partition generates 2 clusters consisting of {{1,3,6,5},{4,2,7,8}}. In similar manner, in Figure 4b), the partition generates 3 clusters consisting of {{1,3},{6,5,4},{2,7,8}}. Finally, in Figure 4c), the partition generates 4 clusters consisting of {{1,3},{6,5},{4,2},{7,8}}.

There are a total of 2,794 possibilities of clustering solution (i.e. from 2 to 4 clusters) for this problem (referred to as the generation of Stirling number of the second kind). However, generating all exhaustive partitions is practically meaningless (i.e. as some partitions will also produce a single module cluster). As such, the way that the partition is organized also influences the results (e.g. not allowing single module partition). By manipulating the sequence using MS-Jaya and adopting the preferred clustering approach (i.e. ECA or MCA), the best clustering solution can be located accordingly.

Taking into account the need to partition the sequence representation of MDG, Figure 5 depicts the complete MS-Jaya pseudo code for addressing the software module clustering problem. In particular, the two round shaded boxes represent the actual partitioning of the MDG sequence as well as the steps for Pareto front generation.

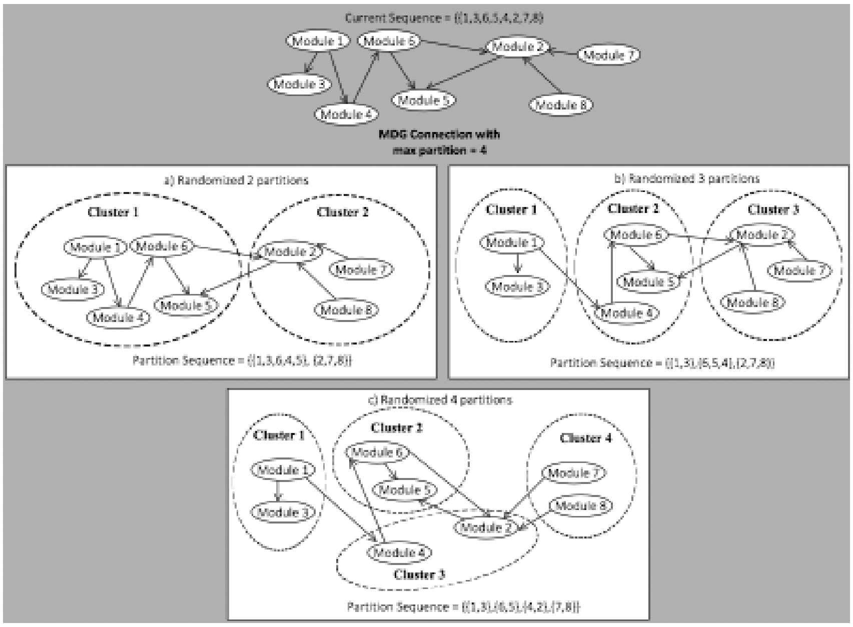

Current Sequence = {{1,3,6,5,4,2,7,8}
Module 1
Module 2
Module 3
Module 4
Module 5
Module 6
Module 7
Module 8
MDG Connection with max partition = 4
a) Randomized 2 partitions
Cluster 1
Cluster 2
Partition Sequence = {{1,3,6,4,5}, {2,7,8}}
b) Randomized 3 partitions
Cluster 1
Cluster 2
Cluster 3
Partition Sequence = {{1,3},{6,5,4},{2,7,8}}
c) Randomized 4 partitions
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Partition Sequence = {{1,3},{6,5},{4,2},{7,8}}

Figure 4. MDG Sequence Representation and Partitioning

Taking into account the need to partition the sequence representation of MDG, Figure 5 depicts the complete MS-Jaya pseudo code for addressing the software module clustering problem. In particular, the two round shaded boxes represent the actual partitioning of the MDG sequence as well as the steps for Pareto front generation.

```
Input: the population $X = \{X_1, X_2 \dots X_n\}$
Output: the updated population $X' = \{X_1', X_2' \dots X_n'\}$ and the Pareto Front archive, $PF = \{X_{p1}, X_{p2}, \dots X_{pn}\}$
[1].  Begin
[2].    Initialize and randomize partition of populations , X
[3].    Set initial $X_{best} = X_0$ and  initial $X_{poor} = X_{best}$
[4].    Set population size, n
[5].    Set max_partition  =  no of modules/2
[6].    While (stopping criteria not met (i.e. t < T))
[7].        Set threshold  $\Delta =$  random between [0,1] $\times$ n
[8].        Set  $X_{best}$  randomly from any $X_i$ in the population
[9].        For population count, i =1 to  n
[10].           If (non_improvement_count == $\Delta$) then
[11].               t =1
[12].               non_improvement_count =0
[13].           EndIf
[14].           Generate random values of  $r_1$ and $r_2$ between [0,1]
[15].           Calculate  $\alpha$ using Eq. 5
[16].           Update the current population using Eq. 4
[17].           For partition = 2 to max_partition
```

```
[18].                 Create partition $X_i^{(t+1)}$
[19].                 If ($X_i^{(t+1)}$) dominates ( $X_i^{(t)}$) then
[20].                     $X_i^{(t)} = X_i^{(t+1)}$
[21].                     If ($X_i^{(t+1)}$) dominates ($X_{best}$) then
[22].                        $X_{best} = X_i^{(t+1)}$
[23].                        Add $X_{best}$ into PF archive
[24].                     EndIf
[25].                 Else
[26].                     non_improvement_count++;
[27].                     If ($X_i^{(t)}$) dominates ($X_{worst}$) then
[28].                        $X_{worst} = X_i^{(t)}$
[29].                      EndIf
[30].                  EndIf
[31].            EndFor
[32].            For i=1 to random[0,1] × n    // Cuckoo elitism scheme
[33].                 Find the worst, $X_i^{(t)}$
[34].                 Generate randomly $X_i^{(t+1)}$
[35].                 For partition = 2 to max_partition
[36].                      Create partition ($X_i^{(t+1)}$)
[37].                      If ($X_i^{(t+1)}$) dominates ($X_i^{(t)}$) then
[38].                         $X_i^{(t)} = X_i^{(t+1)}$
[39].                      EndIf
[40].                  EndFor
[41].            EndFor
[42].            If $X_{best}$ is dominated by any member of existing {$X_{best\_1}$, $X_{best\_2}$,… $X_{best\_n}$}
        from the PF archive
[43].               Discard and remove $X_{best}$ from the PF archive
[44].             Else If $X_{best}$ dominates any member of existing {$X_{best\_1}$, $X_{best\_2}$,… $X_{best\_n}$}
        from the PF archive
[45].               Discard and remove $X_{best\_n}$ from the PF archive
[46].            EndIf
[47].          EndFor
[48].    EndWhile
[49].    Return the updated population, X. Report the PF archive and select the
        preferred solution
[50].  End
```

Figure 5. Pseudo Code for Multi-Objective MS-Jaya Algorithm for Software Clustering Problem

6. Benchmarking Experiments

Our experiments focus on two related goals: (1) to benchmark MS-Jaya against the original Jaya algorithm and other state-of-the-art parameter free meta-heuristic algorithms; (2) to assess the performance of MS-Jaya in terms of modularisation quality, cohesion and coupling for both unweighted and weighted MDGs.

We divide our experiments into two parts. In the first part, we compare MS-Jaya against existing parameter free algorithm (i.e. consisting of the original Jaya [6], Teaching

Learning based Optimization (TLBO)[7], Global Neighborhood Algorithm (GNA) [12], Symbiotic Optimization Search (SOS) [8], and Sine Cosine Algorithm (SCA)[9])) based on the benchmark case studies of unweighted MDGs (available in GitHub [37]) that has been adopted in Mitchell and Maconridis [13], Praditwong et al [14], and Kumari and Srinivas [28] respectively. The descriptions of all the case studies are given in Table 1. We consider both the MCA and the ECA approach for each case study. As the specifications of the weighted MDGs for the abovementioned six MDGs are no longer available (despite contacting the authors), the weighted MDGs as part of Praditwong et al were not experimented.

In the second part, we also compare against parameter free algorithm (i.e. consisting of the original Jaya [6], Teaching Learning based Optimization (TLBO)[7], Global Neighborhood Algorithm (GNA) [12], Symbiotic Optimization Search (SOS) [8], and Sine Cosine Algorithm (SCA)[9])). Unlike the first part, we compare both weighted and unweighted MDGs side-by-side using MCA and ECA approach. Our case studies consist of three MDGs as defined in Hall et al [38]. The descriptions of the three MDGs are given in Table 2 (and depicted in Figures 6 till 8). Unlike the earlier case studies in Table 1, the case studies in Table 2 involve state machine representation whereby clustering of MDGs represent the identification of super states.

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. As all the algorithms are parameter free, we do not need to do any control parameter settings with the exception of the population size and iteration. According to the observation from Kumari and Srinivas [28], given the number of modules (N), the population size must be set at least with 10xN and the number of generation at 200xN to cope with the increased in the complexity in clustering more number of modules. To ensure fair comparison, we also set the same maximum fitness function evaluation for all the case studies. Additionally, we develop all the algorithms adopted in the experiments using the Java programming language based on the same approach to cater for Pareto front generation. Table 3 highlights the settings for all algorithms for each case study highlighted in both part 1 and 2. Meanwhile, the results of all the experiments are tabulated in Tables 4 till 11. Shaded cells indicate the best performance for a set of best solution for each of the case studies (e.g. maximizing MQ, minimizing inter-edges, and maximizing intra-edges) selected from the Pareto lists.

TABLE 1. MDGs for Part 1

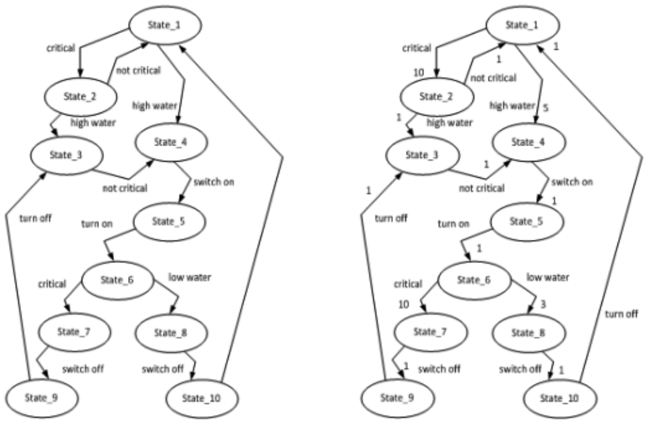
System Name	Modules	Links	Description
Mtunis	20	57	An operating system for educational purpose written in the Turing language
Ispell	24	103	Software for spelling and typographical error correction in files
Rcs	29	163	Revision control system used to manage multiple revisions of files
Bison	37	179	General purpose parser generator for converting grammar description into C program

TABLE 2. MDG for Part 2

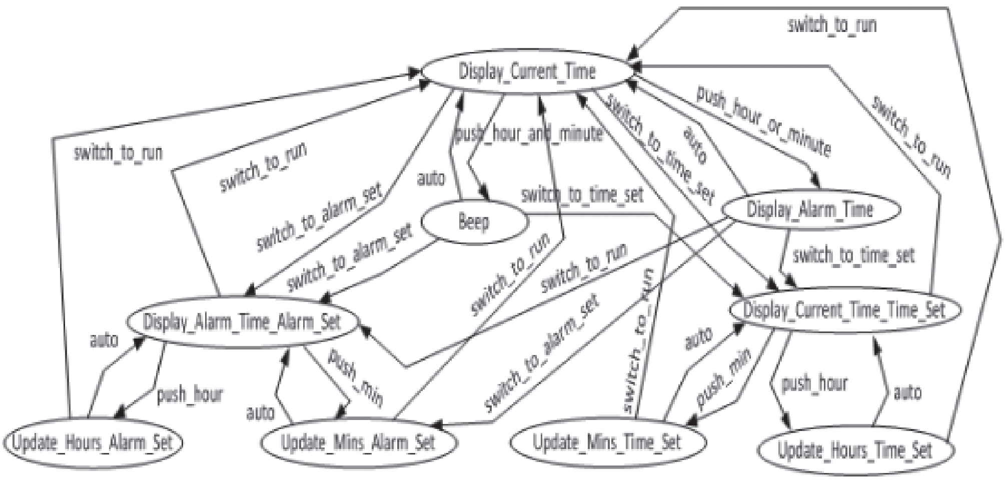
State Machines	Modules	Links	Description
Water Pump Controller [39]	10	14	A state machine representation for the water pump controller with unweighted and assigned weight (i.e. refer to Figure 6)
Alarm Clock of Romera [40]	9	25	A state machine representation for an alarm clock based on a MSc thesis with unweighted and assigned weight (i.e. refer to Figure 7)
Tamagotchi [38]	14	38	A state machine representation of a Tamagotchi pet toy as part of the student project with unweighted and assigned weight (i.e. refer to Figure 8)

TABLE 3. Maximum Fitness Function Evaluation

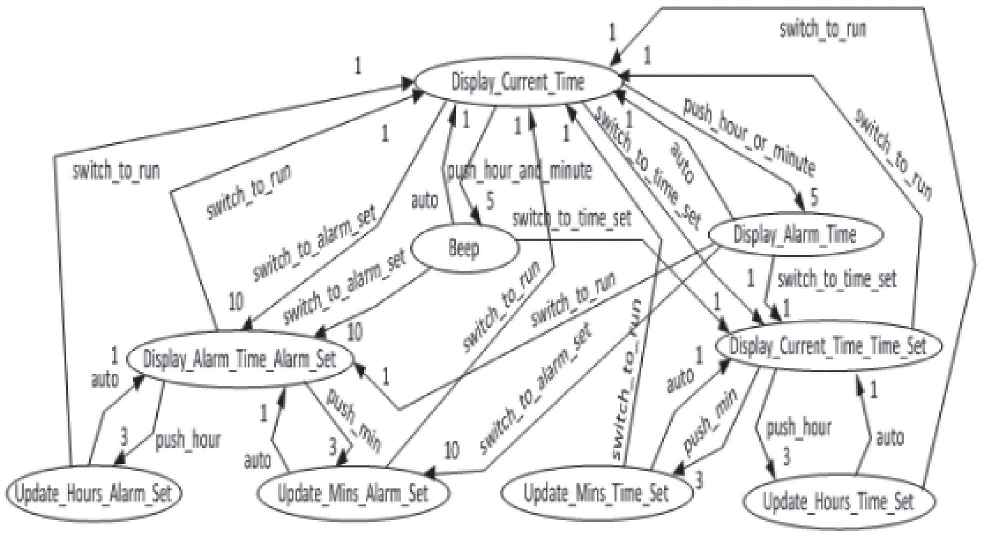
	System Name	Maximum Fitness Function Evaluation as suggested by [14]
MDG for Part I	Mtunis	800000
	Ispell	115200
	Rcs	1682000
	Bison	2738000
MDG for Part 2	Water Pump Controller	200000
	Alarm Clock of Romera	162000
	Tamagotchi	392000



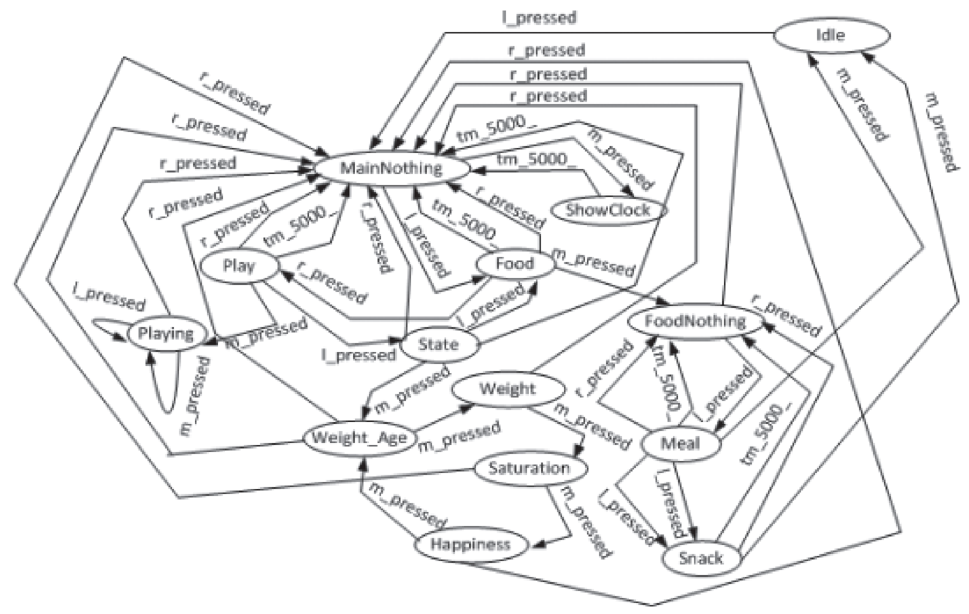
6(a) Unweighted State Machine 6(b) Weighted State Machine
Figure 6. State Machines for Water Pump Controller



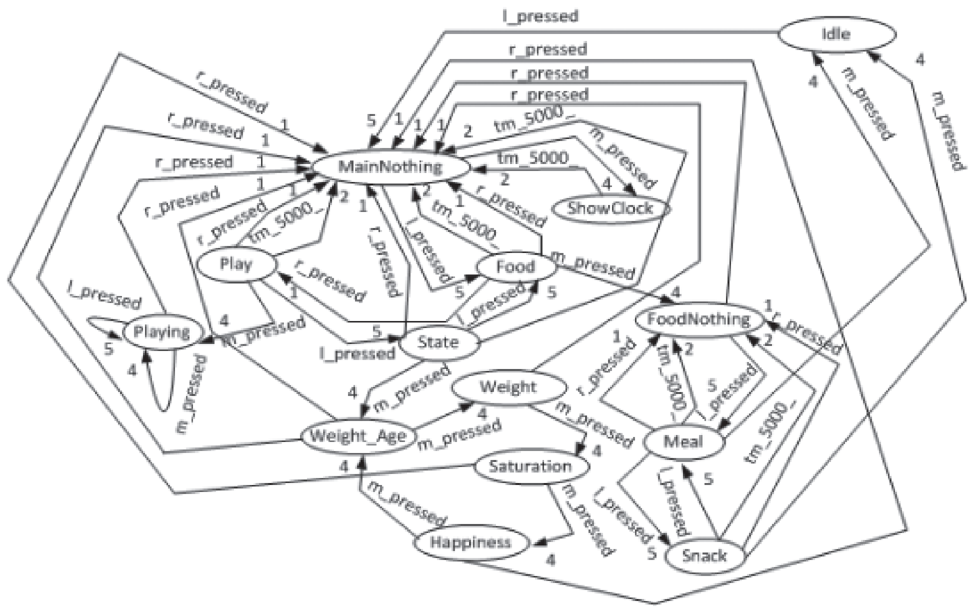
7(a) Unweighted State Machine



7(b) Weighted State Machine
Figure 7. State Machines for Alarm Clock of Romera



8(a) Unweighted State Machine



8(a) Weighted State Machine
Figure 8. State Machines for Tamagotchi

TABLE 4. Maximizing of MQ values with MCA Approach

	Teaching Learning based Optimization [7]		Global Neighborhood Algorithm [12]		Symbiotic Optimization Search [8]		Sine Cosine Algorithm [9]		Original Jaya Algorithm [6]		MS-Jaya Algorithm	
Case Studies	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev.	Mean	Std. Dev.
Mtunis	2.161	0.051	2.179	0.042	2.131	0.064	2.179	0.049	2.2385	0.034	2.239	0.037
Ispell	2.194	0.041	2.237	0.037	2.158	0.06	2.199	0.055	2.277	0.035	2.287	0.032
Bison	1.976	0.075	2.016	0.115	1.913	0.126	1.992	0.097	2.191	0.065	2.207	0.067
Rcs	2.017	0.053	2.036	0.061	1.983	0.069	2.032	0.051	2.113	0.036	2.115	0.045

TABLE 5. Minimizing inter-edges values with MCA Approach

	Teaching Learning based Optimization [7]		Global Neighborhood Algorithm [12]		Symbiotic Optimization Search [8]		Sine Cosine Algorithm [9]		Original Jaya Algorithm [6]		MS-Jaya Algorithm	
Case Studies	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev.	Mean	Std. Dev.
Mtunis	69.011	7.253	69.110	5.848	72.011	6.812	67.011	5.273	67.021	4.494	64.011	5.061
Ispell	150.021	14.550	155.022	9.788	147.034	22.086	151.022	13.66	155.011	7.47	151.056	8.637
Bison	294.021	29.264	285.000	23.862	289.033	33.278	271.054	42.645	277.011	33.654	269.044	35.005
Rcs	190.035	46.946	221.012	36.885	199.020	46.272	177.011	33.465	223.023	39.164	218.034	38.556

Table 6. Maximizing intra-edges values with MCA Approach

Case Studies	Teaching Learning based Optimization [7]		Global Neighborhood Algorithm [12]		Symbiotic Optimization Search [8]		Sine Cosine Algorithm [9]		Original Jaya Algorithm [6]		MS-Jaya Algorithm	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev.	Mean	Std. Dev.
Mtunis	22.012	3.606	22.011	2.958	21.111	3.406	23.011	2.655	23.011	2.280	25.011	2.530
Ispell	27.011	7.287	25.012	4.914	29.012	11.045	27.011	6.823	25.023	3.735	27.012	4.307
Bison	32.023	29.264	36.123	11.925	34.560	16.646	44.560	21.326	40.560	16.821	43.023	17.803
Rcs	67.045	25.475	52.043	23.447	63.023	38.135	74.024	31.733	51.012	19.586	53.002	19.281

TABLE 7. Maximizing MQ values with ECA Approach

Case Studies	Teaching Learning based Optimization [7]		Global Neighborhood Algorithm [12]		Symbiotic Optimization Search [8]		Sine Cosine Algorithm [9]		Original Jaya Algorithm [6]		MS-Jaya Algorithm	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev.	Mean	Std. Dev.
Mtunis	2.159	0.049	2.188	0.052	2.129	0.065	2.168	0.05	2.229	0,029	2.246	0.034
Ispell	2.175	0.06	2.224	0.054	2.126	0.069	2.199	0.051	2.272	0.045	2.288	0.035
Bison	1.995	0.08	2.063	0.117	1.918	0.11	2.004	0.081	2.183	0.072	2.205	0.074
Rcs	2.032	0.05	2.051	0.035	1.993	0.073	2.046	0.071	2.109	0.036	2.102	0.034

TABLE 8. Minimizing inter-edges values with ECA Approach

	Teaching Learning based Optimization [7]		Global Neigh-borhood Algo-rithm [12]		Symbiotic Opti-mization Search [8]		Sine Cosine Algorithm [9]		Original Jaya Algorithm [6]		MS-Jaya Algo-rithm	
Case Stud-ies	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev.	Mean	Std. Dev.
Mtunis	68.011	6.542	68.011	6.066	72.011	6.132	70.011	5.727	67.011	4.171	66.012	6.341
Ispell	150.123	13.327	157.221	5.422	141.020	17.176	147.112	19.519	158.111	3.493	149.111	9.349
Bison	287.112	35.718	275.011	22.424	285.011	29.206	276.012	45.843	275.023	29.513	290.213	15.754
Rcs	198.011	28.996	219.089	21.107	187.112	28.205	221.333	24.433	219.650	32.213	210.213	27.582

TABLE 9. Maximizing intra-edges values with ECA Approach

	Teaching Learning based Optimization [7]		Global Neigh-borhood Algo-rithm [12]		Symbiotic Opti-mization Search [8]		Sine Cosine Algorithm [9]		Original Jaya Algorithm [6]		MS-Jaya Algo-rithm	
Case Studies	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev.	Mean	Std. Dev.
Mtunis	22.023	3.362	22.022	3.162	20.012	3.131	21.011	3.001	23.011	2.088	23.013	2.074
Ispell	28.123	6.663	24.112	2.711	32.111	8.588	29.111	9.767	23.110	1.936	28.004	4.701
Bison	35.111	17.862	41.112	21.216	36.051	14.605	40.099	22.939	41.111	14.761	33.113	7.896
Rcs	63.232	26.511	53.210	20.551	69.011	29.104	52.123	27.221	53.213	26.105	57.123	23.795

TABLE 10. Comparison of best MQ values for Unweighted and Weighted State Machine Superstate Identification with MCA Approach

		Teaching Learning based Optimization [7]		Global Neighborhood Algorithm [12]		Symbiotic Optimization Search [8]		Sine Cosine Algorithm [9]		Original Jaya Algorithm [6]		MS-Jaya Algorithm	
	Case Studies	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev.
Unweighted	Water Pump Controller	2.305	0.000	2.305	0.000	2.305	0.000	2.305	0.000	2.305	0.000	2.305	0.000
	Alarm Clock of Romera	1.377	0.002	1.377	0.002	1.377	0.002	1.394	0.050	1.394	0.050	1.402	0.055
	Tamagotchi	2.584	0.009	2.582	0.016	2.581	0.016	2.557	0.034	2.585	0.017	2.587	0.016
Weighted	Water Pump Controller	2.514	0.000	2.514	0.000	2.514	0.000	2.514	0.000	2.514	0.000	2.514	0.000
	Alarm Clock of Romera	2.123	0.000	2.123	0.000	2.123	0.000	2.123	0.000	2.123	0.000	2.123	0.000
	Tamagotchi	3.193	0.01	3.195	0.013	3.193	0.013	3.182	0.013	3.194	0.003	3.198	0.013

TABLE 11. Comparison of best MQ values for Unweighted and Weighted State Machine Superstate Identification with ECA Approach

		Teaching Learning based Optimization [7]		Global Neighborhood Algorithm [12]		Symbiotic Optimization Search [8]		Sine Cosine Algorithm [9]		Original Jaya Algorithm [6]		MS-Jaya Algorithm	
	Case Studies	Mean	Std. Dev	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev.
Unweighted	Water Pump Controller	2.305	0.002	2.305	0.002	2.305	0.002	2.305	0.002	2.305	0.002	2.305	0.002
	Alarm Clock of Romera	1.377	0.0	1.377	0.0	1.377	0.0	1.402	0.059	1.385	0.036	1.410	0.066
	Tamagotchi	2.586	0.005	2.586	0.021	2.569	0.031	2.563	0.029	2.586	0.005	2.588	0.001
Weighted	Water Pump Controller	2.514	0.000	2.514	0.000	2.514	0.000	2.514	0.000	2.514	0.000	2.514	0.000
	Alarm Clock of Romera	2.123	0.000	2.123	0.000	2.123	0.000	2.123	0.000	2.123	0.000	2.123	0.000
	Tamagotchi	3.194	0.015	3.193	0.014	3.191	0.013	3.189	0.017	3.192	0.003	3.198	0.006

7. Experimental Observation

Based on the experiments undertaken, a number of observations can be elaborated based on the obtained results.

Considering the MCA approach, MS-Jaya outperforms all algorithms for all case studies as far as the mean of maximizing MQ in concerned in Table 4 (with small standard deviations swing). In the case minimizing inter-edges in Table 5, MS-Jaya also outperforms other algorithms in two case studies involving Mtunis and Bison. Here, SoS gives the best mean for Ispell and TLBO for Rcs, respectively. The standard deviations for all cases are large indicating large swing of values. Considering maximizing intra-edges in Table 6, SCA outperforms all other algorithms with the best mean for Bison and Rcs. SoS and MS-Jaya obtain the best mean for Ispell and Mtunis, respectively. Similar to Table 5, the standard deviations for all cases are also large. Often, large standard deviations are caused by the potentially many possible combinations of clusters that yield similar results.

Referring to Table 7 for ECA approach, MS-Jaya gives the best mean for three-out-of-four cases (involving Mtunis, Ispell and Bison) as far as maximizing the MQ is concerned. MS-Jaya, SoS, GNA, and TLBO share the best mean for each case study in Table 8. Similar observation can be deduced in Table 9 as far maximizing intra-edges. It is worth mentioning that the standard deviations are similar to that of the MCA approach given earlier (i.e. in Tables 4 till 6).

Concerning the unweighted and weighted state machine clustering with MCA approach in Table 10, MS-Jaya outperforms all other algorithm as far as the best mean of MQ. For the weighted Water Pump Controller and Alarm Clock of Romera, all algorithm get the best result with no standard deviation indicating that all exhaustive possibilities have been exercised by the specified iteration.

Finally, as far as the unweighted and weighted state machine clustering with ECA approach in Table 11, similar observation can be deduced. MS-Jaya outperforms all other algorithm in terms of the best mean of MQ. For the weighted Water Pump Controller and Alarm Clock of Romera, all algorithm get the best result with no standard deviation. Concerning Tamagotchi case study, the same mean MQ is achieved for ECA and MCA approaches.

8. Concluding Remarks

Reflecting on the earlier given results from the comparative experiments, the usefulness of our approach can be debated further. Firstly, the fact that MS-Jaya has consistently outperformed the original Jaya gives clear indication that the exploration and exploitation have been improved. Although MS-Jaya has not singly outperformed other algorithms in the case of minimizing inter-edges and maximizing intra-edges, the obtained mean results have been competitive. In fact, MS-Jaya gives the overall best as far as obtaining the best MQ values for both MCA and ECA in almost all cases (except in the case of Rcs in Table 7).

Secondly, our modification of the original algorithm (including that of restarting mechanism, Cuckoo search like elitism, and multi-objective capabilities) has not in any way degraded Jaya. Specifically, we have maintained MS-Jaya as parameter free meta-heuristic algorithm. The fact that MS-Jaya is also parameter free retains the attractiveness of Jaya in terms of not requiring significant tuning for adoption in any optimization problems.

Thirdly, given that all the meta-heuristic algorithms are relying on random operators to generate new solution, fairness of the benchmark experiments and their comparisons can be an issue. Additionally, the restarting mechanism within MS-Jaya (i.e. as a way to avoid getting trap in local minima) may also potentially increase iteration. In our case, we define the same maximum fitness function evaluation for all algorithms. In this manner, all the algorithms will terminate with same the maximum fitness function evaluation. As such, the comparisons are fair for all algorithms.

Finally, as the scope for future work, we are looking to investigate the application of MS-Jaya for other optimization problems. In particular, we are interested to contribute in the Search based Software Engineering (SBSE) field whereby meta-heuristic algorithms are being sought to solve complex software engineering optimization problems.

Acknowledgement

The work reported in this paper is funded by Fundamental Research Grant from Ministry of Higher Education Malaysia titled: A Reinforcement Learning Sine Cosine based Strategy for Combinatorial Test Suite Generation (grant no: RDU170103).

References

[1]. Holland, J. H. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press.

[2]. Kennedy, J., Eberhart, R. (1995) Particle Swarm Optimization, Proceedings of the IEEE International Conference Neural Networks, pp. 1942-1948. Perth, Australia.

[3]. Geem, Z. W. (2009) Music-Inspired Harmony Search Algorithm: Theory and Applications, Studies in Computational Intelligence, Springer.

[4]. Dorigo, M., Maniezzo, V., Colorni, A. (1996) Ant System: Optimization by a Colony of Cooperating Agents, IEEE Transactions on Computes Systems, Man, and Cybernetics, Part B: Cybernetics, 26, 29-41.

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

[6]. Rao, R. V. (2016) Jaya: A Simple and New Optimization Algorithm for Solving Constrained and Unconstrained Optimization Problems, International Journal of Industrial Engineering Computations, 7, 19-24.

[7]. Rao, R. V., Savsani, V. J., Vakharia, D. P. (2012) Teaching-Learning-Based Optimization: An Optimization Method for Continous Non-linear Large Scale Problem, Information Sciences, 183, 1-15.

[8]. Cheng, M.Y., Prayogo, D. (2014) Symbiotic Organisms Search: A New Meta-Heuristic Optimization Algorithm, Computers and Structures, 139, 98-102.

[9]. Mirjalili, S. (2016) SCA: A Sine Cosine Algorithm for Solving Optimization Problems, Knowledge Based Systems, 96, 120-133.

[10]. Bashiri, M., Amiri, A., Doroudyan, M. H., Asgari, A. (2013) Multi-Objective Genetic Algorithm for Economic Statistical Design of X Control Chart, Scientia Iranica Transactions E: Industrial Engineering, 20, 909-918.

[11]. Flemming, P. J. (1985) Computer Aided Control Systems Using a Multi-Objective Optimization Approach, Proceedings of the IEEE Control'85 Conference, pp. 174-179.

[12]. Alazzam, A., Lewis, H. W. (2013) A New Optimization Algorithm For Combinatorial Problems, International Journal of Advanced Research in Artificial Intelligence, 2, 63-68.

[13]. Mitchell, B. S., Mancoridis, S. (2006) On the Automatic Modularization of Software Systems using the Bunch Tool, IEEE Transactions on Software Engineering, 32, 193-208.

[14]. Praditwong, K., Harman, M., Yao, X. (2011) Software Module Clustering as a Multi-Objective Search Problem, IEEE Transactions on Software Engineering, 37, 264-282.

[15]. Burd, E., Munro, M. (1998) Investigating Component-based Maintenance and the Effect of Software Evolution: A Reengineering Approach using Data Clustering, Proceedings of the International Conference on Software Maintenance, pp. 199-207.

[16]. Lucca, G. A. D., Fasolino, A. R., Pace, F., Tramontana, P., Carlini, U. D. (2002) Comprehending Web Applications by a Clustering-based Approach, Proceedings 10th International Workshop on Program Comprehension, pp. 261-270.

[17]. Jahnke, J. H. (2004) Reverse Engineering Software Architecture using Rough Clusters, Proceedings of the IEEE Annual Meeting of the Fuzzy Information Processing, pp. 4-9

[18]. Zamli, K. Z., Alkazemi, B. Y., Kendall, G. (2016) A Tabu Search Hyper-Heuristic Strategy for t-way Test Suite Generation, Applied Soft Computing Journal, 44, 57-74.

[19]. Zamli, K. Z., Din, F., Ahmed, B.S., Bures, M. (2018) A Hybrid Q-learning Sine-cosine-based Strategy for Addressing the Combinatorial Test Suite Minimization Problem, PLoS ONE, 13.

[20]. Nasser, A. B., Zamli, K.Z., Alsewari, A.R.A., Ahmed, B.S. (2018) Hybrid Flower Pollination Algorithm Strategies for t-way Test Suite Generation, PLoS ONE, 13.

[21]. Ahmed, B. S., Zamli, K. Z., Afzal, W., Bures, M. (2017) Constrained Interaction Testing: A Systematic Literature Review, IEEE Access, 5, 25706 - 25730.

[22]. Taylor, R. N., Medvidovic, N., Dashofy, E. M. (Ed.) (2009) Software Architecture: Foundations, Theory, and Practice, New-York: John Wile and Sons.

[23]. Sommerville, I. (Ed.) (2001) Software Engineering, Boston: Addison-Wesley.

[24]. Gordon, A. D. (Ed.) (1999) Classification, Boca Raton: Chapman and Hall/CRC.

[25]. Mahdavi, K., Harman, M., Hierons, R. M. (2003) A Multiple Hill Climbing Approach to Software Module Clustering, Proceedings of the International Conference on Software Maintenance, pp. 315-324.

[26]. Praditwong, K. (2011) Solving Software Module Clustering Problem by Evolutionary Algorithms, Proceedings of the 8th International Joint Conference on Computer Science and Software Engineering, pp. 154-159.

[27]. Doval, D., Mancoridis, S., Mitchell, B. S. (1999) Automatic Clustering of Software Systems using a Genetic Algorithm, Proceedings of the Software Technology and Engineering Practice, pp. 73-81.

[28]. Kumari, A. C., Srinivas, K. (2016) Hyper-heuristic Approach for Multi-Objective Software Module Clustering, Journal of Systems and Software, 117, pp. 384-401.

[29]. Huang, J., Liu, J., Yao, X. (2017) A Multi-Agent Evolutionary Algorithm for Software Module Clustering Problems, Soft Computing, 21, 3415-3428.

[30]. Zamli, K. Z., Din, F., Baharom, S., Ahmed, B. S. (2017) Fuzzy Adaptive Teaching Learning-based Optimization Strategy for the Problem of Generating Mixed Strength t-way Test Suite, Engineering Applications of Artificial Intelligence, 59, 35-50.

[31]. Rao, R. V., Rai, D. P., Balic, J. (2017) A Multi-Objective Algorithmfor Optimization of Modern Machining Processes, Engineering Applications of Artificial Intelligence, 61, 103-125.

[32]. Zamli, K. Z., Din, F., Kendall, G., Ahmed, B. S. (2017) An Experimental Study of Hyper-Heuristic Selection and Acceptance Mechanism for Combinatorial t-way Test Suite Generation, Information Sciences, 399, 121-153.

[33]. Rao, R. V., More, K. C. (2017) Design Optimization and Analysis of Selected Thermal Devices using Self-Adaptive Jaya Algorithm, Energy Conversion and Management, 140, 24-35.

[34]. Rao, R. V., Saroj, A. A Self-Adaptive Multi-Population based Jaya Algorithm for Engineering Optimization, Swarm and Evolutionary Computation, 37, 1-26.

[35]. Salwani, A., Laleh, G., Mohd Zakree, A. N. (2011) Re-Heat Simulated Annealing Algorithm for Rough Set Attribute Reduction, International Journal of the Physical Sciences, 6, 2083-2089.

[36]. Knowles, J. D., Corne, D. W. (1999) The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multi-Objective Optimization, Proceedings of the Congress on Evolutionary Computation (CEC '99), pp.98-105.

[37]. Barros, M. (June 2017) MDGs Benchmark. Retrieved from https://github.com/cmsp/ils.

[38]. Hall, M., McMinn, P., Walkinshaw, N. (2010) Superstate Identification for State Machines using Search-Based Clustering, Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 1381-1388.

[39]. Damas, C., Lambeau, B., Dupont, P., Lamsweerde, A. V. (2005) Generating Annotated Behavior Models from End-User Scenarios, IEEE Transactions on Software Engineering, 31, 1056-1073.

[40]. Romera, M. E. (2000) Using Finite Automata to represent Mental Models, MSc Thesis, Department of Psychology, San Jose State University.

Submitted 20.02.2018
Accepted 27.05.2018