Journal of Engineering Science and Technology
Vol. 14, No. 6 (2019) 3213 - 3233

MEMETIC APPROACH FOR MULTI-OBJECTIVE OVERTIME PLANNING IN SOFTWARE ENGINEERING PROJECTS

HAMMED A. MOJEED*, AMOS O. BAJEH,
ABDULLATEEF O. BALOGUN, HAMMID O. ADELEKE

Department of Computer Science, University of Ilorin PMB 1515, Ilorin, Nigeria
*Corresponding Author: mojeed.ha@unilorin.edu.ng

Abstract

Software projects often suffer from unplanned overtime due to uncertainty and risk incurred due to changing requirement and attempt to meet up with time-to-market of the software product. This causes stress to developers and can result in poor quality. This paper presents a memetic algorithmic approach for solving the overtime-planning problem in software development projects. The problem is formulated as a three-objective optimization problem aimed at minimizing overtime hours, project makespan and cost. The formulation captures the dynamics of error generation and propagation due to overtime using simulation. Multi-Objective Shuffled Frog-Leaping Algorithm (MOSFLA) specifically designed for overtime planning is applied to solve the formulated problem. Empirical evaluation experiments on six real-life software project datasets were carried out using three widely used multi-objective quality indicators. Results showed that MOSFLA significantly outperformed the existing traditional overtime management strategies in software engineering projects in all quality indicators with 0.0118, 0.3893 and 0.0102 values for Contribution (I_C), Hypervolume (I_{HV}) and Generational Distance (I_{GD}) respectively. The proposed approach also produced significantly better I_{HV} and I_{GD} results than the state of the art approach (NSGA-II$_V$) in 100% of the project instances. However, the approach could only outperform NSGA-II$_V$ in approximately 67% of projects instances with respect to I_C.

Keywords: Memetic algorithms, Multi-objective optimization, Overtime planning, Project management, Search-based software engineering.

1. Introduction

Software Engineering is concerned with optimization problems that seek to develop software that is faster, cheaper, more reliable, scalable, responsive, maintainable, and testable [1]. Consequently, software engineering projects are hardly completed on schedule and within budget unless good software project management techniques are enforced [2]. Software Project Management (SPM) involves carrying out several activities such as cost estimation, project planning (including Project scheduling and staffing) and quality management, which are critical to the success of a software project [1]. These activities often involve finding an appropriate balance between competing and usually conflicting objectives. SPM is commonly modelled as project scheduling and staffing problem [3], which is usually solved using Search-Based Software Engineering (SBSE) approach. In SBSE a software engineering problem is seen as a search problem whose aim is to find the most appropriate solution that conforms with some adequacy requirements. It seeks to reformulate software engineering problems as search-based optimization problems and applies a variety of meta-heuristics algorithms to solve them.

Software Project Scheduling (SPS) is a kind of optimization problem that seeks to find an optimal schedule for a software project so that the precedence and resource constraints are satisfied and the final project cost consisting of personnel salaries and project duration is minimized [4]. SPS has been tackled using meta-heuristic optimization algorithms [3, 5-8]. Meta-heuristic algorithms search for a suitable solution in a typically large input space, which is guided by a fitness function. The fitness function expresses the goals and leads the exploration into potentially optimal areas of the search space.

These algorithms automate and suggest appropriate schedules and solutions for software engineers to guide their decision in a planning software project. However, the assumption that automated tools should produce the initial project plan may not always be realistic. Ferrucci et al. [9] reported that experience with practitioners revealed their preference for their own judgments for the initial project plan because the allocation of staff to work packages requires several human and domain-specific decisions for which, an automated approach is inadequately equipped to handle. Nevertheless, unguided initial project plans by engineers often subject developers to working overtimes- usually without previous plans - in order to meet up with the project deadlines. Several factors may contribute to the need for overtime. Some of these factors are; late and changing requirements, compressed schedules, difficulties associated with project progress measurement, and the need to meet up with the time-to-market of new features. However, the most reported factors contributing to unplanned overtime are a late change in requirements and reduced time to market [1, 10].

Expectedly, it has been shown that excessive and unplanned overtime allocations have detrimental effects on developers' health and the software they build. It was observed by Nishikitani et al. [11] and Karita et al. [12] that overtime work and long shifts have a positive correlation with stress indexes such as depression, anger, hostility and increase in equilibrium and motor-related problems on IT professionals. Also regarding the effect on the software that developers produce, Akula and Cusick [13] found out that longer overtime work causes increased stress, which in turn translates to higher defect counts in software projects. The implication of these effects is bad quality software since developers

continue to focus more on quick and rapid completion of tasks rather than quality. These shreds of evidence from the literature underscore the need for an effective decision support approach for overtime planning that can balance staff productivity, cost, project duration and product quality against overtime.

To mitigate the bad effects of long overtime on software development projects, several Overtime Management Strategies (OMS) are deployed by engineers in the industry. Ferrucci et al. [9] presented three commonly used overtime strategies by software engineers in the industry: Margarine (MAR) management, Critical Path Management (CPM) and Second Half (SH) management strategies. In MAR, overtime hours are spread evenly across all activities in the schedule. CPM, on the other hand, loads overtime onto the schedule's critical path while in SH, overtime hours are loaded onto the latter half of the schedule to make up for delays introduced in the former half. However, few studies have been carried out to optimally solve Overtime Planning Problem (OPP) through the application of SBSE approach. Current approaches employ genetic multi-objective evolutionary algorithms - mainly NSGA-II and its variants [9, 10, 14]. To the best of our knowledge, no work has been carried out to apply other Multi-Objective Optimization algorithms to the problem of software projects' overtime planning. Therefore, there is a need for more studies on overtime planning techniques that could completely outperform the current overtime management strategies and the state of the art approaches.

This study investigates the effectiveness of the memetic multi-objective evolutionary algorithm in solving software project overtime planning problem using Multi-Objective Shuffled Frog Leaping Algorithm (MOSFLA). With respect to optimization problems, memetic algorithms, which can be viewed as a union between a population-based global technique and a local search made by each individual in the population - have been shown to be more efficient and more effective than traditional evolutionary algorithms for some problem domains [2, 15, 16]. As a result, memetic algorithms are gaining wide acceptance, particularly in well-known combinatorial optimization problems where large instances have been solved to optimality and where other meta-heuristics have failed to produce comparable results.

The remaining part of this paper is organized as follows: Section 2 presents some related works in software projects' overtime planning. Section 3 focused on the formulation of the OPP problem. Section 4 is based on the proposed framework, which presents the employed computational search approach for solving OPP and the evaluation metrics employed to measure the performance of the approach. Section 5 presents the experimental setup and discusses the results, and the work is concluded in Section 6.

2. Related Works

Akula and Cusick [13] carried out a statistical survey on four real-life software projects to analyze the impact of overtime on software quality. The number of scheduled work hours, actual overtime hours, and a number of defects were recorded on each of the projects. Statistical analysis showed that defect count is directly proportional to the actual overtime hours in all considered projects. It was observed that defect count increases when overtime is extensive and stress on workers and the project develops during excessive overtime.

In spite of this evidence from the literature, there have been very few studies on the application of search-based meta-heuristic algorithms to software projects overtime planning. This study could only find three works that apply SBSE to the problem. Ferrucci et al. [9] introduced a search-based optimization approach to software overtime planning. The study presented the formulation of software project overtime planning as a multi-objective optimization problem. Their objective is to investigate the effects of choices of overtime allocations on the project schedule, each of which, seeks to minimize project duration, the amount of overtime and risk of overrun based on three risk assessment models.

A new variant of NSGA-II called NSGA-II$_v$ that exploits a crossover operator specifically designed for overtime planning problem was employed as the computational search method. To analyse the effectiveness of the approach they reported an empirical study on six real-life software projects. Results from experiments revealed that the approach is significantly better than standard NSGA-II in 76% of experiments and significantly outperformed standard overtime planning strategies used in the industry. However, their formulation failed to assess the negative effect of overtime on software quality even when empirical studies in the literature [11, 13] have revealed that the effort required to correct additional errors introduced by developers working overtime outweighs the productivity gains.

Recently, Sarro et al. [14] extended the work of Ferrucci et al. [9] by proposing an adaptive method for meta-heuristic operator selection to improve algorithmic performance under the same problem formulation. The study presented a new variant of NSGA-II, namely Adaptivevsc, which systematically combines the crossover operator proposed by Ferrucci et al. [9] and adaptive genetic operators of NSGA-II$_a$, proposed by Nebro et al. [17]. In addition to that, the study introduced an adaptive strategy for the selection of genetic operators during the search. Results from experimental studies on eight real-life software projects showed that the proposed adaptive outperformed the state-of-the-art approaches in 93% of the experiments and significantly performed better than the current overtime planning strategies in 100% of the experiments.

Barros and Araujo [10] advanced the formulation proposed by Ferrucci et al. [9] by considering the reported effect of overtime work on the quality of software. The study introduced a new formulation for the OPP that make use of simulation to propagate the effects of an increased number of defects due to overtime to the project's duration and cost. The main objective of their work is to use optimization and simulation to learn the effects of different overtime planning policies on the project duration and cost. NSGA-II was used as the computational search method to optimally plan over time with the aim of minimizing the overall project duration, project makespan, and cost.

The approach was compared with overtime strategies used in industry and a similar model devoid of the negative effects of overtime on product quality. Experimental results showed that NSGA-II conveniently outperformed the margarine and critical path overtime strategies. However, the Second-Half overtime management strategy produced competitive results with NSGA-II under this formulation. This observation calls for a better search approach that could completely outperform all the overtime strategies under the same formulation.

This study is close to the work of Barros and Araujo [10] as it adopts the same problem formulation but employs different computational search approach (memetic) using MOSFLA.

3. Problem Formulation

This study adopts a three-objective decision problem model presented by Barros and Araujo [10] for overtime planning, which captures the dynamics of error generation and propagation due to overtime using simulation. This approach estimates the impact of increased defect counts due to overtime on the project duration and cost.

Let a project schedule be represented as a Directed Acyclic Graph (DAG) consisting of a node-set WP = {wp_1, wp_2,pt wp_n} of Work Packages and an edge set DP = { (wp_i, wp_j) : i :j; 1 or$i \leq n$, 1 or$j$ 1n } of dependencies among WPs. Each wp is characterized by the amount of effort (measured in function points (FP)), which is required to complete it. Each dependency in DP depicts that development of a work package (wp_j) can only start after the completion of the analysis of another work package (wp_i).

Given the size of a work package in FP and average productivity of 27.8 FP/developer-month as estimated empirically [18] for Information Systems (IS) projects, the expected duration of the work package can be calculated as:

$$Duration(wp_i) = effort(wp_i)/(27.8) \quad (1)$$

The shortest possible duration of the project is given by any maximal length path called Critical Path. The critical path has been used for decades to build software projects schedules. However, the main problem is to analyse the effects of overtime assignments on the schedule, which seek to minimize project makespan, cost and the amount of overtime deployed.

With respect to error propagation during software development projects, Jones [18] found out that developers introduced an average of 4 errors/FP through analysis, design and coding activities of a component. This dynamic affects the duration of testing activities. Also regarding error generation due to overtime, Abdel-Hamid and Madnick [19] modelled the relationship between the amount of overtime spent by developers and the number of defects introduced into the software they produce. This model estimates an introduction of 20% more errors by a developer working 10 hours/day than those working on regular shifts of 8 hours/day and 50% more errors by developers working 12 hours/day. This non-linear relationship between overtime and error generation rates, coupled with the error propagation dynamics form a compound model on which, an optimization technique driven by simulation is required. Therefore, this study combines optimization and simulation to optimally solve the software overtime planning problem.

Optimization objectives

The OPP formulation in this study tries to minimize:

- Amount of Overtime Hours (OH) = $\sum_{i=1}^{n} overtime(wp_i)$ (2)
- Project MakeSpan (PMS) = $\sum_{wp \in CP} duration(wp_i)$ (3)

- Project Cost (PC) = $\sum_{i=1}^{n} C_r(wp_i) + C_o(wp_i)$ (4)

Subject to the constraints:

$0 \leq overtime(wp_i) \leq maxovertime(wp_i),\ 1 \leq i \leq n$

$DP_{violation} < 1$.

Overtime Hours (OH) is the sum of overtime hours spent on all activities in the schedule as calculated by the computational search algorithm. The Project Makespan (PMS) is defined by the longest precedence-respecting path in the graph and it is calculated as the sum of the duration of activities in the path. CP is the Critical Path and duration (wp_i) is the duration of an activity. The duration of an activity is computed from the simulation since the error generation and propagation dynamics is integrated into the project's schedule. The Project Cost (PC) is the sum of each activity's cost, which depends on the amount of regular and overtime hours its developer spends to come up with its expected outcomes. The Brazilian cost law presented by Barros and Araujo [10] is used to model the cost of an activity. It follows that if a regular working hour costs C, each of the first two overtime hours cost 120% of C and each of the next two hours cost 150% of C. C_r in objective (iii) is the cost of regular working hours and C_o is the cost of overtime hours.

A candidate solution to the OPP is modelled as a set of integer values denoting the number of overtime hours to be spent daily on each Work Package (WP) in the schedule. It is represented as a linear array of integers representing the overtime hours to be spent on activities denoted by the corresponding indexes of the array. It is important to note that the length of regular working hours and the maximum overtime hours are country-based parameters. In this study by Barros and Araujo [10], the working hours are set to 8 hours and maximum overtime of 4 hours per day.

4. Proposed Framework

This section presents the computational search approach employed to solve the overtime planning problem modelled in section 3 and the evaluation metrics to be used to measure the performance of the proposed computational approach.

4.1. Computational search approach

A Multi-Objective Shuffled Frog-Leaping Algorithm (MOSFLA) - with the same framework as the original SFLA algorithm proposed by Eusuff and Lansey [20] - that employs an archiving strategy based on adaptive niche technique proposed by Cui [21] is employed as the computational search in this study. The niche technique is used to maintain the non-dominated solutions. The algorithm improves on population sorting technique and memetic evolution process to adapt to Multi-Objective Optimization (MOO).

Due to parallel evolution mechanism of the algorithm, the solutions evolve toward different directions. This makes MOSFLA be specifically fit for MOO problems like the one considered in this study. The MOSFLA algorithm is illustrated in Fig. 1 as adapted by Yinghai et al. [22]. MOSFLA has been applied successfully to various problems in different domains such as standard optimization test problem instances [23], reservoir flood control problem [22] and robot path planning problem [24].

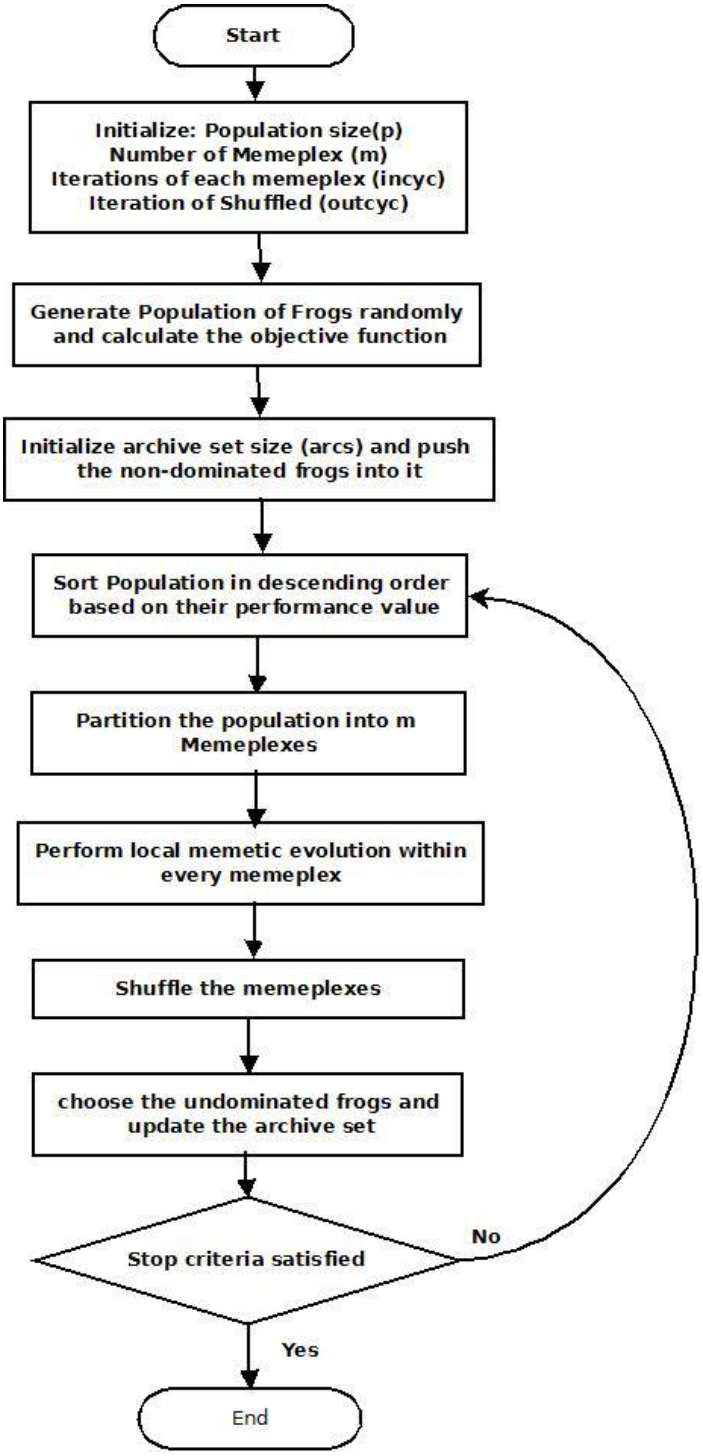

Fig. 1. Flowchart of MOSFLA [22].

4.2. MOSFLA design for overtime planning problem

To adapt the MOSFLA to overtime planning problem the algorithm is designed as follows:

- **Population creation and sorting strategy**

Because MOSFLA is a stochastic algorithm, the population of feasible solutions (frogs) is generated randomly. The acceptance and rejection criterion for any solution is the satisfaction of the maximum overtime constraints. In SFLA, frogs are sorted in decreasing order of their performance values calculated through a fitness function. For single-objective problems, performance value is usually set directly as the objective function value. This cannot be applied to multi-objective problems. Various approaches have been used in the literature to measure the performance value for multi-objective problems.

Some of the approaches used are Pareto dominance relation of the solutions, crowding distance of individual solution, a combination of crowding distance within the non-dominated solutions and the hamming distance between dominated and non-dominated solutions in the population [22, 23, 25]. In this study, a new hybrid multi-objective fitness function that is based on both the crowding distance (of the individual frogs) and the rank (obtained by ordering Pareto fronts) is employed to measure the fitness of each solution in the population.

More specifically, the rank is calculated using the following procedure:

Step 1. The non-dominated solutions in the initial population are included in the first rank (i.e., rank 0) and then removed.

Step 2. The next non-dominated solutions are determined from the remaining solutions, and included in the next rank (rank 1). These solutions are also removed accordingly.

Step 3. The procedure is continued until no more solutions exist in the population.

After the ranking, crowding distance (C_d) is computed for all solutions in the same rank. Supposing that the MOO has *r* goals, the crowding distance of each frog is calculated as:

$$C_d = \sum_{k=1}^{r} |P[i+1] \cdot f_k - P[i-1] \cdot f_k| \quad (5)$$

C_d is the crowding distance of the i^{th} frog in the rank set, $P[i+1]$ f_k and $P[i-1]$ f_k are k^{th} objective function values of two adjacent frogs.

Alejandro et al. [24] proposed the overall multi-objective fitness function, which is calculated using the formula. It is defined as:

$$MOFit = \frac{1}{2^{rank} + \frac{1}{1+C_d}} \quad (6)$$

- **Memplex formation**

The sorted frogs are stored in an array X = {*P(i), i = 1, . . . ,n}*. X is then partitioned into *m* memeplexes, i.e., $Y^1, Y^2, \ldots, Y^m$, each containing *n* frogs, such that:

$$Y^k = [\, P(i)^k \mid P(i)^k = P[\, k + m(i-1)\,]\,],\ i = 1,..n\ \ k = 1,..m \quad (7)$$

In this way, for m = 4, frog in position 1 goes to memeplex 1, position 2 to memeplex 2, position 3 to memeplex 3, position 4 to memeplex 4. Then frog in position 5 goes to memeplex 1, and so on.

- **Memetic evolution in memeplex**

Within each memeplex, the virtual frogs go through a memetic evolution where the worst performing fog is improved via meme transfer and sharing. In the original SFLA, the worst frog in each memeplex is improved using the evolution step:

$$d = rand * (x_b - x_w), \quad newxw = oldx_w + d \tag{8}$$

where x_b = local best frog in the memeplex and *xw* = worst frog in the memeplex.

This evolution step is inefficient as it limits the location of the new frog new*xw* in the area between *xw* and x_b. This only enforces a local bound improvement. In order to extend the evolution area of new*xw* out of the local bound, the following evolution step proposed by Yinghai et al. [22] is employed:

$$d = 2 * rand * (x_b - x_w), \quad newx_w = oldx_w + d \tag{9}$$

This step doubles the evolution area so that the position of new*xw* can reach $2(xb-x_w)$. In essence, the evolution step can produce solutions that are better than the current best solution within each memplex.

The following steps are taken to achieve local evolution:

- The local best frog (x_b) is set as the first frog *yk* [1] and the worst frog (x_w) as the last frog *yk*[n] of k^{th} memeplex. For frogs to evolve toward Pareto optimal, the global best frog (x_g) is set as a solution randomly chosen from the current archive set.
  - The position of the worst frog is adjusted using Eq. (9). The objective function values are calculated and the Pareto dominance relationship between new*xw* and old*xw* are compared:
    - If $newx_w$ dominates $oldx_w$, then $y^k[n]$ is replaced with new*xw*.
    - If $oldx_w$ dominates $newx_w$, then go to iii;
  - Step two is recomputed by substituting x_b with xg in Eq. (9).
    - If $newx_w$ dominates the $oldx_w$, then $y^k[n]$ is replaced with new*xw* and
    - If $oldx_w$ dominates $newx_w$, then go to iv*;*
  - A new solution is generated randomly to replace the worst frog. To guide this process towards an evolutionary direction, the new solution is produced by randomly generating a new frog in the neighbourhood of the global best frog x_g.
  - After` the memetic evolution step, the memeplex Y^k is updated and resorted.
  - For each memeplex, steps *i* to *v* are repeated for a specific number of iterations.
  - Shuffling and archiving strategy.

After a number of memetic evolution, the memeplexes are combined and sorted in descending order of their MOfit value. Non-dominated solutions are determined and added to the archive set. In many MOO algorithms, the archiving strategy is used for maintaining the set of non-dominated solutions. Niche method is an

effective approach to ensure diversity of the non-dominated solutions within the set. In the niche-based archiving method, niche radius is used to calculate sharing fitness of non-dominated solutions. The sharing of fitness is calculated as follows:

$$F(i) = 1/\sum_{j=1}^{q} sh(d_{ij}) \tag{10}$$

where:

$$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^{\alpha} & d_{ij} < \sigma_{share} \\ 0 & d_{ij} > \sigma_{share} \end{cases} \tag{11}$$

$F(i)$ is the sharing fitness of i^{th} non-dominated solution; q is the number of solutions in the archive set; sh(d_{ij}) represents the sharing function of i^{th} and j^{th} non dominated solution; d_{ij} is the Euclidean distance of objective space between i^{th} and j^{th} non-dominated solutions; α is a constant coefficient and σ_{share} is the niche radius.

It can be observed that the niche radius directly affects $F(i)$. An unfitted niche radius may cause an ununiformed spread of the non-dominated solutions. Considering the dependence of niche radius (σ_{share}) on the number (and distribution) of solutions in the archive set, and the difficulty of specifying it a priori, this study employs a self-adaptive calculation method for σ_{share} by automatically computing and adjusting it in the iteration procedure as proposed by Chui [21]. The calculation procedure is given in Eqs. (12) and (13).

$$\sigma_{share} = \begin{cases} C & if\ q < 2 \\ \sum_{i=1}^{q} {}^{d_i}/_{q} & if\ q \geq 2 \end{cases} \tag{12}$$

$$d_i = min\ (\ ||\ F_i(x) - F_j(x)\ ||\ )\ for\ i, j = 1,2,\ldots,q\ \ such\ that\ j \neq I \tag{13}$$

where q represents the number of solutions in the archive set; d_i denotes the minimum Euclidean distance in the objective space between the i^{th} non-dominated solution and others, and C is a positive constant usually set as 1. With this, the niche radius is calculated as the mean value of d_i of all non-dominated solutions in the archive set.

4.3. Multi-objective evaluation measures used

This study employs the use of three quantitative solution set quality measures: Contributions (I_C), Hypervolume (I_{HV}), and Generational Distance (I_{GD}) adopted by Ferrucci, et al. [9] and Barros and Araujo [10]. They are measured in the [0, 1] interval.

I_C: Is a convergence measure that computes the proportion of solutions produced by an algorithm, A, that lies on the reference front RS. It calculated as the ratio of the solutions in RS produced by A [26] and it is defined as follows:

$$I_C(A/RS) = \frac{\frac{|C|}{2} + |W_A| + |N_A|}{|RS|} \tag{14}$$

where C is the Pareto solution set common to both A and RS, W_A is the solutions sets in A, which dominate other solutions in RS and N_A is the set of solutions in A have dominance relation with no solution in *RS.* A good Pareto front should have high I_C value and contribute greatly to the reference front.

I_{HV}: Calculates the volume within the objective space covered the set of non-dominated solutions from an algorithm of interest. This measure mixes convergence and diversity. Optimal 3D Hypervolume Algorithm described by Paquete et al. [27] is employed in this study. The algorithm sweeps along a front sorted in one objective, maintaining an overall 2D area for the points considered so far. For each point, *p*, in the front, a height-balanced binary tree is queried to determine the position of p in the remaining objectives. If p is dominated, it is discarded. If *p* dominates other points, they are deleted from the tree. If needed, the 2D area is then updated in constant time. The height from *p* to the next point down (i.e., the slice depth) is then multiplied by the area and the result is added to the overall volume. The pseudo-code for the algorithm is shown in Fig. 2. The higher the value of I_{HV}, the better the algorithm.

I_{GD}: Is a convergence measure that calculates the average distance between solution set S, from the algorithm of interest and the reference set RS. The distance between S and RS in an N objective space is computed as the average N-dimensional Euclidean distance between each point in S and its nearest neighbouring point in RS. It is defined as follows [28]:

$$I_{GD}(S) = \frac{1}{|RS|} \sum_{y \in RS} min \{d_{xy} \mid x \in S\} \quad (15)$$

The d_{xy} is the distance between a solution *x* in S and a reference solution *y* in RS in the *N*-dimensional objective space as defined in Eq. (16):

$$d_{xy} = \sqrt{(f_1(y) - f_1(x))^2 + \cdots + (f_N(y) - f_N(x))^2} \quad (16)$$

where f_i (*x*) is the i^{th} objective function values of a solution *x*. Good fronts possess low I_{GD} and thus, are closer to the reference front.

INPUT: Pareto Front (PS) from an algorithm of interest
OUTPUT: Hypervolume Value I_{HV}

Begin:
Step 1: Initialize tree, sort PS in 3rd objective and set Volume to 0
Step 2: Set *p* = head (PS), *ps* = tail (PS), area = p[0] * p[1], *z* = *p*
Step 3: For each *p* in PS
Step 4: Search tree for point *q* to the right of *p*
Step 5: If *p* is not dominated
Step 6: Increase volume by slice between *z* and *p*
Step 7: *z* = *p*
Step 8: For each point *s* in tree dominated by *p*
Step 9: Remove *s* from tree
Step 10: Decrease area by contribution of *s*
Step 11: End for
Step 12: Increase area by contribution of *p*
Step 13: Insert *p* in tree
Step 14: Increase volume by area * z[2]
Step 15: End for

End

Fig. 2. Optimal 3D hypervolume algorithm [27].

5. Results and Discussion

This section discusses the design of our empirical experimental study comprising of the dataset used, parameter settings, and the results of experiments.

5.1. Software project data used

This study makes use of six (6) real-life software project data collected by Barros and Araujo [10] and made publicly available for study replication and validation. The dataset is summarized in Table 1. OMET is an application that manages meteorological information. WAMS is an air traffic routing control system that manages traffic control messages. PARM stores and manages user profiles configuration settings used by many applications.

PSOA is a personnel management system that manages users' authentication and authorization from enterprise systems. ACAD is an academic portal system that manages university students and staff records. WMET manages meteorological information in a database.

Table 1. Instances used for empirical studies.

Name	Activities	Dependencies	Function points
OMET	84	63	635
WAMS	60	45	381
PARM	108	91	451
PSOA	72	84	290
ACAD	40	39	185
WMET	44	33	225

5.2. Parameter setting

Due to the non-deterministic nature of MOSFLA, three major parameters were set experimentally. The values of the parameters were varied in subsequent runs and the best performing values were selected for the actual experiment.

The shuffling iteration that determines the stopping criteria is tested with values 500, 1000, 1500 and 2000. The randomness incumbent to MOSFLA was catered for by executing the optimization process 30 times for each value and instance. Thereafter, a reference front was built from non-dominated solutions collated from all the 30 cycles for each instance. Then, IGD was computed for the front generated from each cycle to determine the most appropriate and effective value.

For the most adequate initial population size, a similar method was used. The population sizes of 2*m*, 3*m*, and 4*m*, (*m* is the number of activities for each instance) were compared. Table 2 shows the preliminary result of this experiment on instance ACAD. For all experiments, the number of memeplexes was set to 5. For the number of iteration of memetic evolution within each memeplex, the same procedure was also applied for each instance. Since the value is dependent on the number of frogs' *n* in each memeplex, the value is set to 2*n*, 4*n*, and 8*n*. Table 3 shows the preliminary result of this experiment on instance ACAD.

From the preliminary results, it can be inferred that the shuffling iteration 1500, population size 4*m* (m is the number of activities of the instance) and evolution iteration value 4*n* (*n* is the number of frogs in each memeplex) produced best results

in all the considered combinations. Therefore, the three major parameters are set accordingly for the actual experiment.

Table 2. I_{GD} value for different combination of shuffling iteration and population size.

Population/ iterations	500	1000	1500	2000
2*m*	0.122	0.134	0.023	0.102
3*m*	0.022	0.130	0.022	0.026
4*m*	0.024	0.025	0.012	0.023

Table 3. I_{GD} value for different combination of shuffling iteration and evolution iteration.

Evolution iterations /shuffle iterations	500	1000	1500	2000
2*n*	0.041	0.044	0.102	0.122
4*n*	0.031	0.035	0.002	0.032
8*n*	0.044	0.042	0.22	0.034

5.3. Experimental results

MOSFLA implemented in java was applied to all the project instances with all parameters set as discussed in the previous subsection. Each experiment is run 30 times to cater for the stochastic nature of the algorithm and results are averaged. The multi-objective quality indicators Contributions (I_C), Hypervolume (I_{HV}), and Generational Distance (I_{GD}) are measured for each instance. The reference front is built from all the fronts generated from each run. Table 4 shows the results of I_C, I_{HV}, and I_{GD} for all the instances considered.

From Table 4, it is obvious that the proposed memetic algorithm performed effectively with very high hypervolume value close to 0.7, a high contribution value close to 0.4 and considerably low Generational distance, which underscores the robustness and effectiveness of the algorithm. It can also be deduced that the algorithm performed better for large projects as it produces the best hypervolume and generational distance value for the instances with the highest number of activities of 108 (PARM), except for Contribution where it produced the best result in a medium-scale project with 84 activities (OMET).

It can be concluded that MOSFLA is most suitable for overtime planning problem in large-scale software engineering projects. To further evaluate the effectiveness of the algorithm, its performance was compared with the typical overtime management strategies in software industries. The OPP formulation with MOSFLA as search method is compared with three OMS strategies presented by Ferruci et al. [9]: "margarine" (MAR), Critical Path (CPM) and Second Half (SH). Fronts produced by MOSFLA over 30 optimization cycles were compared with the front generated by each OMS based on the quality indicators. The reference front was built from all the Pareto fronts generated by all the OMS strategies and the MOSFLA. Results of the OMS strategies are adapted from [10]. Table 5 presents the results of MOSFLA compared with all OMS strategies.

Table 4. Quality indicators results of MOSFLA for all instances.

Instances	I_C	I_{HV}	I_{GD}
OMET	0.3851	0.5512	0.0010
WAMS	0.3014	0.5210	0.0012
PARM	0.2185	0.6502	0.0009
PSOA	0.2654	0.4870	0.0048
ACAD	0.1320	0.6740	0.0023
WMET	0.2170	0.6098	0.0023

Table 5. Quality indicator values for MOSFLA and OMS.

Instances	I_C				I_{HV}				I_{GD}			
	MOSFLA	**MAR**	**SH**	**CPM**	**MOSFLA**	**MAR**	**SH**	**CPM**	**MOSFLA**	**MAR**	**SH**	**CPM**
OMET	0.0119	0.0076	0.0115	0.0076	0.4342	0.2545	0.3544	0.2089	0.0101	0.2346	0.0300	0.1356
WAMS	0.0086	0.0053	0.0079	0.0053	0.3722	0.2547	0.3547	0.1718	0.0076	0.1511	0.0418	0.5151
PARM	0.0354	0.0137	0.0342	0.0137	0.4489	0.2308	0.3579	n/a	0.0104	0.1518	0.0215	n/a
PSOA	0.0073	0.0063	0.0063	0.0063	0.3303	0.1496	0.2069	n/a	0.0123	0.2309	0.2309	n/a
ACAD	0.0016	0.0024	0.0048	0.0072	0.3402	0.2303	0.3589	n/a	0.0145	0.1783	0.0488	n/a
WMET	0.0060	0.0055	0.0055	0.0055	0.4102	0.2544	0.3538	0.1913	0.0065	0.2420	0.0418	0.1530

With respect to Contributions (I_C) quality indicator, it can be seen that MOSFLA outperforms all the overtime management strategies in all the six instances except for ACAD instance where it recorded the lowest value of 0.0016. Figure 3 shows a comparison of the results of the Contribution (I_C) indicator.

For Hypervolume (I_{HV}) quality indicator, it can be observed that MOSFLA performs significantly better than all the overtime management strategies across all the instances except for ACAD instance where SH recorded the highest value of 0.3589 with MOSLA trailing closely with 0.3402 value. This is clearly presented in Fig. 4.

The low performance of MOSFLA in ACAD instance might be due to the small size of the project as the algorithm ultimately performs better as with the larger project as shown in Table 4. Considering Generational Distance (I_{GD}), MOSFLA completely outperforms all the overtime management strategies recording the lowest values in all the six project instances.

Figure 5 shows the Generational Distances **(I_{GD})** of MOSFLA and overtime management strategies. MOSFLA produces significantly better values in I_{GD} and I_{HV} for all instances. For I_C MOSFLA only produced slightly better values compared to the other OMS.

For proper comparison, the average Contribution (I_C), Hypervolume (I_{HV}) and Generational Distance (I_{GD}) values recorded by MOSFLA and other OMS strategies are presented in Table 6.

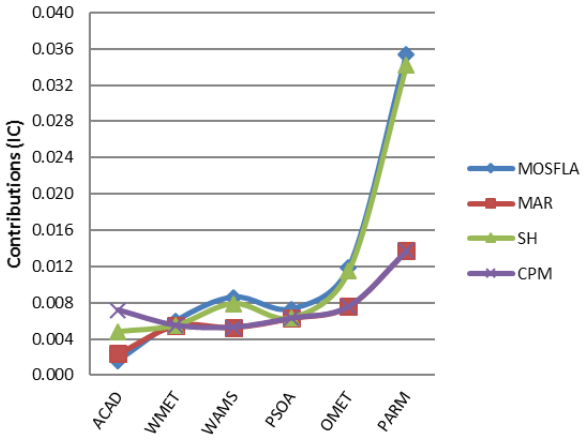

Fig. 3. Contributions of MOSFLA and other OMS.

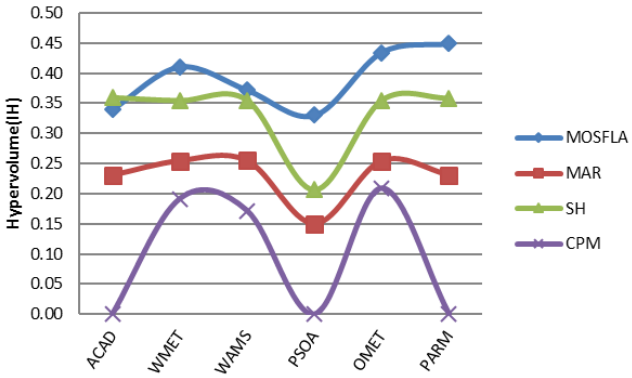

Fig. 4. Hypervolumes of MOSFLA and other OMS.

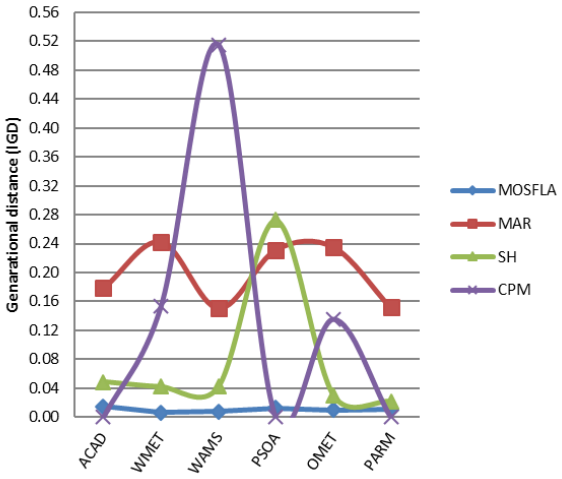

Fig. 5. Generational Distances of MOSFLA and other OMS.

Table 6. Comparison of MOSFLA with other OMS based on average I_C, I_{HV}, and I_{GD}.

	Average contribution (I_C)	Average hypervolume (I_{HV})	Average generational distance (I_{GD})
MOSFLA	**0.0118**	**0.3893**	**0.0102**
MAR	0.0068	0.2291	0.1981
SH	0.0117	0.3311	0.0761
CPM	0.0076	0.1906	0.2679

It can be observed that the proposed algorithm MOSFLA under the current formulation outperformed all other overtime strategies. On the average, it had the highest values of 0.0118 and 0.389 in Contribution(I_C) and Hypervolume (I_{HV}) quality indicators respectively and the lowest value of 0.0102 in Generational Distance (I_{GD}) quality indicator.

Generally, on average, each run of MOSFLA contributed to the reference front with 5.8 solutions and OMS contributed overall of 6.83 solutions. This indicates that the proposed approach contributed up to 46% of all the solutions generated and approximately just 1 solution less than those produced by the OMS strategies altogether, which depict the superiority of the MOSFLA algorithm overall currently practised overtime management strategies in industries.

Lastly, in order for the proposed memetic approach to be adopted, it must also outperform the state of the art approach for the problem in hand. In this case, current solution approaches in SBSE for the problem of planning software projects' overtime employ NSGA-II and its variance [9, 10, 14]. As a means of evaluation, the results of MOSFLA are compared with NSGA-II$_V$ results recorded by Barros and Araujo [10] since our work is based on the same dataset. Table 7 presents the results of the comparison.

With respect to Contribution (I_c), it can be observed that NSGA-II$_v$ produced higher values of 0.2636 and 0.0848 in ACAD and WMET instances. However, as the size of the project instances increase, MOSFLA overtakes NSGA-II$_v$ in terms of performance value as it recorded higher values of Contribution (I_c) in WAMS, PSOA, OMET and PARM with the margin increasing in direct proportion.

It can be observed that as the project size increases the performance of MOSFLA increases while the contrary is the case of NSGA-II$_v$, which diminishes in value as the projects grow bigger. The observed change is graphically presented in Fig. 6. This result underscores the effectiveness of MOSFLA in handling large scale problems.

Considering Hypervolume (I_{HV}) and Generational Distance (I_{GD}) MOSFLA significantly outperformed NSGA-II$_v$ in all project instances recording highest I_{HV} of 0.4489 in PARM and lowest I_{GD} value of 0.0065 in WMET. Figure 7 presents the graphical representation of the I_{HV} of MOFLA and NSGA-II$_v$.

Table 7. Comparison of MOSFLA and NSGA-II$_V$.

Instances	I_C		I_{HV}		I_{GD}	
	MOSFLA	**NSGA-II$_V$ [10]**	**MOSFLA**	**NSGA-II$_V$ [10]**	**MOSFLA**	**NSGA-II$_V$ [10]**
OMET	0.0119	0.0006	0.4342	0.2729	0.0101	2.0360
WAMS	0.0086	0.0060	0.3722	0.3080	0.0076	2.0528
PARM	0.0354	0.0006	0.4489	0.4211	0.0104	2.8130
PSOA	0.0073	0.0024	0.3303	0.2320	0.0123	0.8928
ACAD	0.0016	0.2636	0.3402	0.3075	0.0145	1.5962
WMET	0.0060	0.0848	0.4102	0.3136	0.0065	2.3436

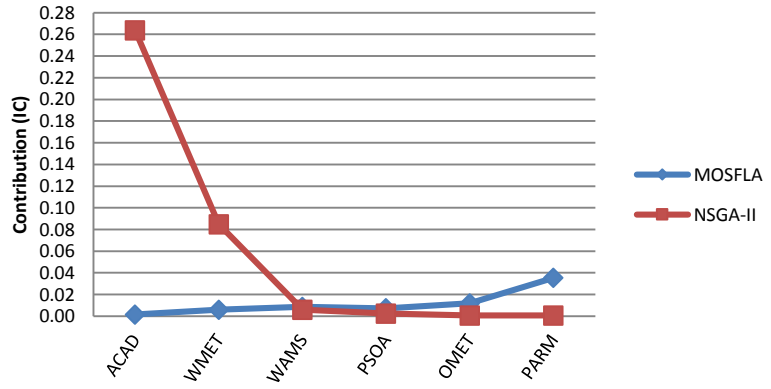

Fig. 6. Contributions of MOSFLA and NSGA-I.

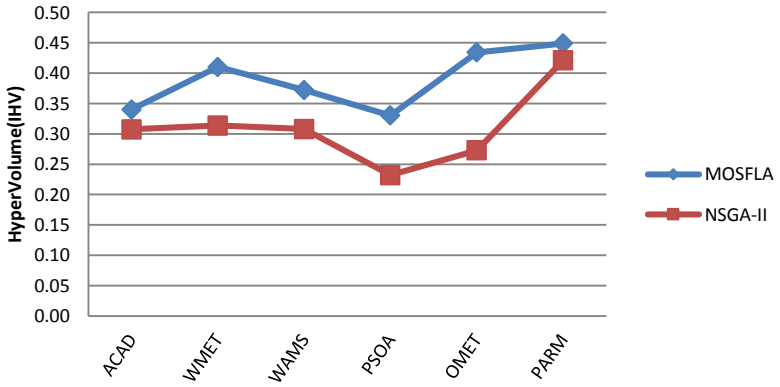

Fig. 7. Hypervolumes of MOSFLA and NSGA-II$_V$

6. Conclusion

A memetic algorithm based on MOSFLA for multi-objective overtime planning in software engineering projects has been developed. The Overtime Planning Problem is formulated as a three-objective optimization problem capturing error generation and propagation dynamics due to overtime using simulation. A variant of MOSFLA

specifically designed for overtime planning is applied to solve the formulated problem. The algorithm incorporates a self-adaptive niche-based archiving strategy to maintain the non-dominated solution. An effective sorting and memetic evolution procedures were applied to adapt the algorithm to MOO. The performance of the algorithm was evaluated empirically on real-life software project dataset and results showed that the approach is effective for managing medium and large-scale software development as it outperformed all currently used overtime management strategies in all quality indicators. The memetic approach also outperforms competitively the state of the art approach (NSGA-II$_v$) in all quality indicators. In the future, we planned to further investigate the effect size and significance of the result using some inferential statistical methods for in-depth analysis. We also plan to measure empirically the effect of overtime on the quality of the software being built using software quality prediction techniques [29-31].

Nomenclatures

C_d	Crowding distance
C_o	Cost of overtime hours
C_r	Cost of regular hours
d_{ij}	Euclidean distance of objective space between the i^{th} and j^{th} non-dominated solutions
$F(i)$	Sharing Fitness
I_C	Contribution
I_{GD}	Generational Distance
I_{HV}	Hypervolume
$MOFit$	Multiobjective Fitness Function
$Sh(d_{ij})$	Sharing function of i^{th} and j^{th} non-dominated solutions
x_b	Local best frog
x_g	Global best frog
x_w	Worst frog
Y^k	K^{th} memeplex

Greek Symbols

α	Sharing constant-coefficient
σ_{share}	Niche radius

Abbreviations

CPM	Critical Path Management
DAG	Directed Acyclic Graph
DP	DePendency
FP	Function Points
MAR	MARgarine Management
MOO	Multi-Objective Optimization
MOSFLA	Multi-Objective Shuffled Frog-Leaping Algorithm
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OH	Overtime Hours
OMS	Overtime Management Strategies
OPP	Overtime Planning Problem

PC	Project Cost
PMS	Project Make Span
SBSE	Search Based Software Engineering
SH	Second Half Management
SPM	Software Project Management
SPS	Software Project Scheduling
WP	Work Packages

References

1. Ferrucci, F.; Harman, M.; and Sarro, F. (2014). Search-based software project management. *Software Project Management in a Changing World*. Chapter 15, 373-399.
2. Oladele, R.O.; and Mojeed, H.A. (2014). A shuffled frog-leaping algorithm for optimal software project planning. *African Journal of Computing and ICTs*, 7(1), 147-152.
3. Ren, J. (2013). *Search based software project management*. Ph.D. Thesis. University College London, London, United Kingdom.
4. Patil, N.; Sawanti, K.; Warade, P.; and Shinde, Y. (2014). Survey paper for software project scheduling and staffing problem. *International Journal of Advanced Research in Computer and Communication Engineering*, 3(1), 215-324.
5. Chang, C.; Christensen, M.; and Zhang, T. (2001). Genetic algorithms for project management. *Annals of Software Engineering*, 11(1), 107 - 139.
6. Alba, E.; and Chicano, F. (2007). Software project management with GA's. *Information Sciences*, 177(11), 2380-2401.
7. Gueorguiev, S.; Harman, M.; and Antoniol, G. (2009). Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. Montral, Canada, 1673-1680.
8. Stylianou, C.; and Andreou, A.S. (2013.) A multi-objective genetic algorithm for intelligent software project scheduling and team staffing. *Intelligent Decision Technologies*, 7(1), 59-80.
9. Ferrucci, F.; Harman, M.; Ren, J.; and Sarro, F. (2013). Not going to take this anymore: Multi-objective overtime planning for software engineering projects. *Proceedings of the International Conference on Software Engineering (ICSE)*. Piscataway, New Jersey, United States of America, 462-471.
10. Barros, M.d.O.; and Araujo, L.A.O.d.J. (2016). Learning overtime dynamics through multiobjective optimization. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. Denver, Colorado, United States of America, 1061-1068.
11. Nishikitani, M.; Nakao, M.; Karita, K.; Nomura, K.; and Yano, E. (2005). Influence of overtime work, sleep duration, and perceived job characteristics on the physical and mental status of software engineers. *Industrial Health*, 43(4), 623-629.
12. Karita, K.; Nakao, M.; Nishikitani, M.; Iwata, T.; Murata, K.; and Yano, E. (2006). Effect of overtime work and insufficient sleep on postural sway information-technology workers. *Journal of Occupational Health*, 48(1), 65-68.

13. Akula, B.; and Cusick, J. (2008). Impact of overtime and stress on software quality. *Proceedings of the 4th International Symposium on Management, Engineering, and Informatics (MEI).* Orlando, Florida, United States of America, 9 pages.
14. Sarro, V.; Ferrucci, F.; Harman, M.; Mannay, A.; and Ren, J. (2017). Adaptive multi-objective evolutionary algorithms for overtime planning in software projects. *IEEE Transactions on Software Engineering*, 43(10), 898-917.
15. Deng, J.; and Wang, L. (2017). A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm and Evolutionary Computation*, 32, 121-131.
16. Poonam, G. (2009). A comparison between memetic algorithm and genetic algorithm for the cryptanalysis of simplified data encryption standard algorithm. *International Journal of Network Security and its Applications (IJNSA)*, 1(1), 34-42.
17. Nebro, A.J.; Durillo, J.J.; Machin, M.; Coello, C.A.C.; and Dorronsoro, B. (2013). A study of the combination of variation operators in the NSGA-II algorithm. *Lecture Notes in Computer Science*, 8109.
18. Jones, C. (2000). *Software assessments, benchmarks, and best practices*. Boston, Massachusetts, United States of America: Addison-Wesley Longman Publishing Co.
19. Abdel-Hamid, T.; and Madnick, S.E. (1991). *Software project dynamics: An integrated approach*. Upper Saddle River, New Jersey, United States of America: Prentice-Hall.
20. Eusuff, M.; and Lansey, K. (2003). Optimization of water distribution network design using the shuffled frog leaping algorithm. *Journal of Water Resources. Planning and Management*, 129(3), 210-225.
21. Cui, X.X. (2006). *Multi-objective evolutionary algorithms and their applications*. Beijing, China: National Defence Industry Press.
22. Yinghai, L.; Jianzhong, Z.; Yongchuan, Z.; Hui, Q.; and Li, L. (2010). Novel multiobjective shuffled frog leaping algorithm with application to reservoir flood control operation. *Journal of Water Resources Planning and Management*, 136(2), 217-226.
23. Jie, Z.; and Wei, N. (2014). Novel multi-objective optimization algorithm. *Journal of Systems Engineering and Electronics*, 25(4), 697-710
24. Alejandro, H.; Miguel, A.V.; Joaquín, F.; and Nieves, P. (2015). MOSFLA-MRPP: Multi-objective shuffled frog-leaping algorithm applied to mobile robot path planning. *Engineering Applications of Artificial Intelligence*, 44, 123-136.
25. Rahimi-Vahed, A.; and Mirzaei, A.H. (2007). A hybrid multi-objective shuffled frog-leaping algorithm for a mixed-model assembly line sequencing problem. *Computer & Industrial Engineering*, 53(4), 642-666.
26. Talbi, E.G. (1999). Métaheuristiques pour l'optimisation combinatoire multi-objectif. *Etat de l'art, C.N.E.T Report* (France Telecom).
27. Paquete, L.; Fonseca, C.M.; and Lopez-Ibanez, M. (2006). An optimal algorithm for a special case of klee's measure problem in three dimensions. *Technical Report CSI-RT-I-01*, CSI, Universidade do Algarve.
28. Elena, S.N. (2007). Performance measures for multi-objective optimization algorithms. *Matematică - Informatică - Fizică*, 109(1), 19-28.

29. Orenyi, B.A.; Basri, S.; and Jung, L.T. (2012). Object-oriented software maintainability measurement in the past decade. *Proceedings of the International Conference on Advanced Computer Science Applications and Technologies (ACSAT).* Kuala Lumpur, Malaysia, 257-262 .
30. Bajeh, A.O. (2015). *Object-oriented software design maintainability measurement*. PhD Thesis. Universiti Teknologi Petronas, Perak, Malaysia.
31. Balogun, A.O.; Bajeh, A. O.; Orie, V.A.; and Yusuf-Asaju, A.W. (2018). Software defect prediction using ensemble learning: An ANP based evaluation method. *FUOYE Journal of Engineering and Technology (FUOYEJET),* 3(2); 50-55.