

OPTIMAL PROJECT SCHEDULING AND STAFFING WITH HYBRID SFLA-TABU ALGORITHM: A PERFORMANCE EVALUATION

¹Mojeed H. A., ²Jimoh R. G., ³Sadiku P. O. & ⁴Salihu S. A.

Department of Computer Science, University of Ilorin, Ilorin, Nigeria

¹Mojeed.ha@unilorin.edu.ng, ²Jimoh_rasheed@unilorin.edu.ng, ³peterposdoy@gmail.com, ⁴salihusa1980@gmail.com

ABSTRACT

Planning a large-scale software project involves the objectives of optimal ordering of a set of activities and an allocation of staff to activities. Current adopted method presents difficulty in reaching optimal good solutions when the two objectives are combined. This study proposes a hybrid SFLA-TABU search algorithm for solving project scheduling and staffing problem with the two objectives combined. The hybrid algorithm retains the framework of SFLA algorithm but within the local memetic evolution, employs the neighborhood structure method of Tabu search and its avoidance of already explored area in the solution space to move towards optimal solution. The performance of the algorithm was evaluated on 12 standard problem instances using hit rate, average fitness and length. Experimental results showed that the proposed algorithm outperforms original SFLA and Tabu search in 100% of the considered instances based on hit rate and quality of solution (measured as average fitness). It however consumes more time than the other two algorithms in about 60% of the problem instances considered. Our findings revealed that the algorithm can be applied to handle various sizes of software projects if given enough time. The results prove the effectiveness, robustness and scalability of the proposed approach.

Keywords: Software Project Management, Software Project scheduling, Staffing, Optimization.

1.0 INTRODUCTION

Software Engineering as a systematic approach for developing software is concerned with optimization problems [1]. Consequently, software engineering projects cannot be 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]. One important activity within SPM is Software Project Scheduling (SPS).

SPS deals with estimation, planning, cost control, budget management and resource allocation in software development projects [3]. Software Project Scheduling is a problem of finding an optimal schedule for a software project so that the precedence and resource constraints are satisfied, and the final project cost and project duration are minimized [4]. It is one of the challenging problems faced by software project managers in the highly competitive software industry.

Developing effective and efficient project scheduling tool for project managers becomes desirable. Also, software project development requires the coordination of the efforts of a team of experts with varying skills. Staffing involves optimal allocation of team to tasks with the aim of minimizing cost and project duration [5]. Studies showed that major challenges in software project management lies in staffing [6].

There have been a number of studies employing various approaches to tackle the problem of Software Project Scheduling using population-based search methods such as Genetic Algorithm [3], Ant Colony Optimization [7], Shuffled Frog Leaping Algorithm (SFLA) [2], Differential Evolutionary Algorithm [8],

[9]. These studies however formulate the problem as only task scheduling problems. Moreover, Stylianou and Andreou [5] proposed a better formulation that adopts an optimization technique to construct a project's optimal schedule (first objective) and to assign the most experienced employees to tasks (second objective) as separate objectives using GA. However, the adopted method presents difficulty in reaching optimal solutions if the objectives are combined especially when having preference to assign the most experienced employees over the project's duration.

This study focused on reformulating the project scheduling and staffing problems as one objective optimization problem and applying a hybrid algorithm based on Shuffled frog-Leaping Algorithm (SFLA) and Tabu search to optimally solve the problem with the aim of improving the overall quality of the solution.

1.1 Shuffled Frog-Leaping Algorithm

Shuffled Frog-Leaping Algorithm (SFLA) proposed by Eusuff and Lansey[10] is a memetic metaheuristic algorithm which is based on evolution of memes carried by interactive individual and a global exchange of information among themselves [11]. The shuffle frog leaping algorithm progresses by transforming frog (solution) in a memetic evolution. In the SFLA, an initial population consist of a set of frogs (solutions) is partitioned into subsets referred to as memeplexes, the different memeplexes are considered as different culture of frogs, each performing a local search, within each memeplexes, the individual frogs hold ideas of other frogs and evolve through a process of memetic evolution. After a defined number of memetic evolution step, ideas are passed among memeplexes in a shuffling process, the local search and the shuffling processes continue until defined convergence criteria are satisfied [12].

Specifically, an initial population of frog is created randomly and then the fitness of the frog is evaluated. The frogs are then sorted in descending order of their fitness and divided into different memplexes after which a local search is performed within each memplex. Let the best frog be X_b , the worst frog be X_w , and the global best frog be X_g , the worst frog is improved thus:

Change in Frog position:

$$(D_i) = rand() \cdot (X_b - X_w) \quad (1)$$

New position: $X_n = X_w + D_i$

$$(D_{max} \geq D_i \geq -D_{max}) \quad (2)$$

Where $rand()$ is a random number between 0 and 1; and D_{max} is the maximum allowed change in a frog's position. If this process yields a better frog (solution), it replaces the worst frog. Elsewise, the calculations in equations (1) and (2) are repeated with respect to the global best frog (that is X_g replaces X_b). If this process is still unable to generate a better performance, the position is randomly generated for the worst frog. The iteration continues for a predefined number G of memetic evolutionary times within each memplex and then the entire population is mixed together in the shuffling process. The local evolution and global shuffling continues for some specific number of

iterations [13]. The SFLA algorithm is given in listing 1.

1.2 Tabu Search Algorithm

Tabu search is a metaheuristic search method employing local search methods used for mathematical optimization. It imitates human memory, starts from an initial feasible solution, chooses a series of specific search direction (neighborhood space) as a temptation, and uses Tabu list that makes sure the areas that have been searched are not searched again.

The Tabu search procedure is generally simple. The procedure starts with a feasible initial solution and stores it as the current seed and the best solution.

The neighbours of the current seed are then produced by a neighbourhood structure. These are candidate solutions. They are evaluated by an objective function, and a candidate which is the best not Tabu or satisfies the aspiration criterion is selected as new seed solution. This selection is called a move and added to Tabu list, another (the oldest one) move is removed from Tabu list if it is overloaded. If the new seed solution is better than the current best solution, it is stored as new best solution. Iterations are repeated until a stop criterion is satisfied. [14]. Listing 2 shows the pseudocodes for the Tabu search.

SFLA ALGORITHM

Input: Initial Population (p), Number of memplex(m), iteration within memplex(q)

Output: Best solutions

1. Begin;
2. Generate random population of P frogs;
3. For each individual i in P: calculate fitness (i);
4. Sort the population P in descending order of their fitness;
5. Divide P into m memplexes;
6. For each memplex:
7. Determine the best and worst frogs;
8. Improve the worst frog position (Local search)
9. Repeat for a specific number of iterations;
- End for;
10. Combine (shuffle) the evolved memplexes;
11. Sort the population P in descending order of their fitness;
12. Check if termination criterion is satisfied;
13. Output best solution(s)
14. End;

Listing 1: The SFLA Algorithm

TABU SEARCH

```

1: Input: Initial solution  $S_0$ , neighborhood  $N(s)$ , search depth  $\beta$ 
2: Output: The best solution  $S_b$  found during the Tabu search process
3    $S = S_0$  /*  $S$  is the current solution */
4    $S_b = S$  /*  $S_b$  is the best solution found so far */
5:  $d = 0$  /*  $d$  counts the consecutive iterations where  $S_b$  is not updated */
6: repeat
7: Choose a best eligible neighboring solution  $s' \in N$  /*  $s'$  is eligible if it is not in Tabu list or better
   than  $S_b$  */
8:  $S = s'$ 
9: Update Tabu list
10: if fitness( $s$ ) < fitness ( $S_b$ ) then
11:  $S = S_b$ 
12:  $d = 0$ 
13: else
14:  $d = d + 1$ 
15: end if
16: until  $d = \beta$ 
17: return  $S_b$ 

```

Listing 2: Pseudocodes for TABU Search Algorithm

2.0 LITERATURE REVIEW

Karova, et al. [3] applied genetic algorithm to software project scheduling and planning problem with an attempt to minimize the overall completion time. Their method was applied on randomly generated problems and results showed that project manager can use this technique in order to better simulate realistic situations and, if needed, reorder the activities and delay the project deadline. This work focuses only on task scheduling without considering staff allocation.

In another study, Oladele and Mojeed [2] extended the work of Karova et al. [3], by applying SFL algorithm to an SPSP by determining only the optimal work packages ordering with a view to minimize the completion time of the software project (mono-objective). It was shown that the SFL algorithm works better than the GA according to their experimental results. Their study also did not consider staff allocation but suggests a multi-objective formulation of the problem.

Gerasimou *et al.* [15] investigated the application of a Particle Swarm Optimization (PSO) algorithm to software project scheduling and effective team staffing. The study aims to create optimal project schedules by specifying the best sequence for executing a project's tasks to minimize the total project duration and seeks to form skilful

and productive working teams with the best utilization of developer skills. A combination of Constriction-PSO and Binary-PSO variations were employed to solve the problem. Results from empirical experiments showed that PSO was able to generate feasible solutions with feasibility rate of approximately 100% and hit rate of virtually 100% in all of considered problems. However, as the complexity and size of the problems increase a progressive decrease in these percentages is observed reaching as low as 30%. This shows that the employed algorithm still encounters difficulties in producing optimal solution as project complexity increases.

Chen and Zhang [16] developed an approach based on an event-based scheduler (EBS) and an ant colony optimization (ACO) algorithm for optimal project scheduling and staffing. The model employed the event based scheduler to simplify the restricted flexibility of human resource allocation. The project plan was model as task list and employee allocation matrix, then Ant Colony Optimization (ACO) algorithm was applied to solve the problem.

Experimental results on 83 problem instances showed that the representation scheme with the EBS is effective, and the proposed algorithm manages to yield better plans with lower costs and more stable workload assignments compared with other existing

approaches such as the Tabu Search (TS) algorithm for the multiskill scheduling Problem, the knowledge-based GA (KGA) and the time-line-based GA. The study however considered not the employee experience in the formulation.

Amiri and Barbin [8] applied Differential Evolutionary Algorithm (DEA) to a resource constrained Project Scheduling problem which is intended to program a group of activities of minimal duration while considering precedence and resource constraints, their objectives are to minimize completion time and overall cost of the project. Results gotten from their approach were compared with the ones generated using GA and it was found that DEA perform better than GA with lesser error rate.

In [17], Maenhout, and Vanhoucke evaluated different scheduling policies and practices for different personnel resource types in integrated project Scheduling and personnel staffing problem. The study examined the impact on the staffing cost when the personnel resources are scheduled in a cyclic versus a non-cyclic manner for different patterns. The problem is modeled using Master Problem formulation and Integer Linear Programming is employed as computational solution approach. Experimental results showed that non-cyclic scheduling leads to a considerable lower staffing cost under all circumstances compared to cyclic scheduling. It was found that flexible temporary resources are essential on top of the regular personnel resources to respond to the variability in demand.

Stylianou and Andreou [5] adopted an approach as an optimization technique to construct a project's optimal schedule (first objective) and to assign the most experienced employees to tasks (second objective). Their study adopts a separate objective function (fitness function) to handle each objective (constraint). The result obtained from their study when using either of the objective functions show that the GA is capable of finding optimal solutions for projects of varying sizes but when the objective functions were combined as one, their adopted GA presents difficulty in reaching optimal solutions. Their work formed the bases of this study as we apply a hybrid SFLA-TABU search algorithm to the same problem combining the objectives with preference to assigning tasks to the most experienced personnel.

3.0 METHODOLOGY

3.1 Problem Modelling

Planning a large scale software project involves a set of activities called Work Packages (WPs), and an allocation of staff to work packages. Given a fixed number of Work Packages, there exists an optimal WP ordering and optimal people distribution into teams. Such resource allocation problems belong to the class of Non-deterministic Polynomial-time hard (NP-hard) problems. Given a project that consists of a set $WP's = \{wp_1, wp_2, \dots, wp_n\}$ of tasks to be performed, a set of dependency constraints $DP =$

$\{(wp_i, wp_j), \dots, (wp_m, wp_n)\}$, such that $0 < i < n$ and $0 < j < n$ and $j \neq i$ of dependencies between tasks, where wp_j requires wp_i to be completed first. A Schedule is modeled using Design Structure Matrix (DSM) also referred to as Dependency Structured Method which enforces the dependencies among tasks (WP's).

It is represented in a jagged array; a two-dimensional array maintained as a one-dimensional array in which each element refers to a one-dimensional array. The row indices denote Work Package (WP) ids. Using a scheduling problem consisting of a software project

containing seven WPs which denote the tasks involved in the development of the project, the corresponding DSM can be represented as shown in figure 1.

WP 1						
WP 2	1					
WP 3	1	2				
WP 4	1	2				
WP 5	1	2	3	4		
WP 6	4	5				
WP 7	5					

Figure 1: DSM model for dependencies

From figure 1, it can be deduced that work package 1 does not rely or depend on any task before it can start, before work package 2 can start, it depends on work package 1 to finish, before work package 3 can also start, work packages 1 and 2 must have finished, before work package 4 can also start, work packages 1, 2 and 3 must have finished, before work package 5 can also start, work packages 1, 2, 3, 4 must have finished. Before work package 6 can start, work packages 1, 2, 3, 4, 5 must have finished and before work package 7 can start, work packages 1, 2, 3, 4, 5, 6 must have finished.

This illustration shows how dependent each and every work package is on the work packages that precedes them. For the problem model, any scheduling problem must have number of WPs that is less than or equal to $2^n - 1$, where n is the total number of employee available for the whole software project.

The staff allocation (employee assignment) of any work package is represented by the binary equivalence (code) of any number between 1 to $2^n - 1$, in n -bit format, where n is the total number of employee available for the whole software project. Each bit position in the binary equivalence represents an employee. If the bit is 1, the associated employee is assigned for the given work package and if 0, the employee is not assigned.

The leftmost bit represent employee 1, the next bit position represent employee 2, and so on. Assuming four (4) employees were available for the example project containing seven work packages cited above, the employee assignment of any of the work packages would be binary equivalence of any number between 1 and 15 (that is $2^4 - 1$). The employee assignment representation for each work package can take the form as shown in table 1 below.

Each employee has his own inherent skill set which is represented as an n -array of class 'Skill' type. Also, each work package has its own required competence(s) which is represented as an n -array of class 'Skill' type. The team of employee assigned to a WP produces a 'total competence' which is a cumulation of all the inherent skill set possessed by each team member.

Table 1: Representation of employee assignments

Work Package	Employee assignment	Employee assignment representation (Binary equivalence)	Remarks
1	2	0010	Only employee 3, is assigned (owing to the bit position, third place from the left)
2	11	1011	Employee 1, employee 3 and employee 4 are assigned
3	15	1111	Employee 1, 2, 3 and 4 are all assigned
4	5	0101	Employee2 and employee4 are assigned.
5	4	0100	Only employee2 is assigned.
6	13	1101	Employee 1, 2 and 4 are assigned.
7	7	0111	Employee 2,3 and 4 are assigned

We seek an optimal ordering of tasks in the sequence in which they should be completed without violating dependency constraints such that the overall project completion time is minimized and optimal staff allocation to tasks such that skill mismatch is minimized.

3.2 Solution Representation and Fitness Function

A feasible solution to the problem depicts effective Work Package (WP) ordering and proper staff allocations. Each frog is encoded as an $n \times 2$ array. Each row index is associated with a WP and it indicates the position of the WP in the WPs ordering. For example, row index 0 indicates position (POS) 1, and the associated WP start first before any other WP. On each row, the value on the first column represents a WP id and the value on the second column represents employee assignment for that WP (whose binary equivalence is what is being actually used). The solution representation is shown in figure 2.

POS 1	WP1	1
POS 2	WP2	5
POS 3	WP4	6
POS 4	WP3	7
POS 5	WP5	2
POS 6	WP7	3
POS 7	WP6	7

Figure 2: Solution representation

The fitness of a solution is estimated as the sum of the total number of dependency constraint (s) violated and the total number of skill mismatched allocation(s) for each WP in the solution. The dependency constraint violation of a given WP in a frog is measured by checking if the WP(s) it depends on (according to the DSM) have all preceded it or not. For each of the WPs it depends on, if the WP does not precede it in the WP ordering, a violation occurs and one (1) is added to the number of dependency violation, V , otherwise no violation and zero (0) is added. There is a skill mismatched allocation to a W_{pi} , if for each employee, e_i assigned to W_{pi} : skill-set(e_i) $\cap$ required competence (W_{pi}) = ϕ (that is e_i does not have any of the WP's required competence(s)) or if required competence (W_{pi}) is not a subset of the set of the total competences of the team of employee(s) assigned to W_{pi} . For every skill mismatched allocation to each WP in a frog, the number of the total skilled mismatched allocation, M of the frog increases by one (1). The lower the fitness value, the better the solution. The fitness of a solution, A is mathematically represented as:

$$fitness(A) = \sum_{i=1}^n (V + M); fitness(A) \geq 0 \quad (3)$$

Where V is the number of dependency violations, M is the number of skilled mismatched allocations and n is the number of WPs for the software project.

Considering the sample project presented in table 2, the required competence(s) of each WP in the sample project, the skill set of the employees available for the project, the DSM in figure 1 and solution representation A given in figure 2, the fitness of the solution is given as:

$$V(A) = 0 \quad M(A) = 0$$

$$Fitness(A) = V(A) + M(A) = 0$$

(No violation of dependency and no skill mismatched allocation(s)).

Table 2: A Sample Project with Skill definition, Employee skill-set and WPs' required competences

ID	Skills	Skill Description
1	Skill1	Programming
2	Skill2	Object oriented design
3	Skill3	System analysis
4	Skill4	GUI Designing
5	Skill5	Database management
Employees		
ID	Name	Skill-set
1	Employee1	{Skill1, Skill2, Skill4}
2	Employee2	{Skill1, Skill2}
3	Employee3	{Skill3, Skill5}
Activities (WP)		
ID	Name	Required Competence(s)
1	Feasibility Study	{Skill3}
2	Requirement Analysis	{Skill1, Skill3, Skill4}
3	Design Database	{Skill1, Skill3, Skill5}
4	Design GUI	{Skill1, Skill4}
5	Coding	{Skill1}
6	Write user manual	{Skill1, Skill3}
7	Testing	{Skill1, Skill2, Skill3}

3.3 Hybrid SFLA-TABU Algorithm

This study employs a hybrid SFLA-TABU search algorithm as a solution approach for the SPSP. In the hybridization, the SFLA forms the framework within which Tabu search is used as the local search rather than the SFLA's inherent local search method. The algorithm capitalizes on the memetic power of SFLA and fast convergence of Tabu due to its avoidance of already exploited area in the solution space. The hybridized SFLA-Tabu framework is depicted in figure 3.

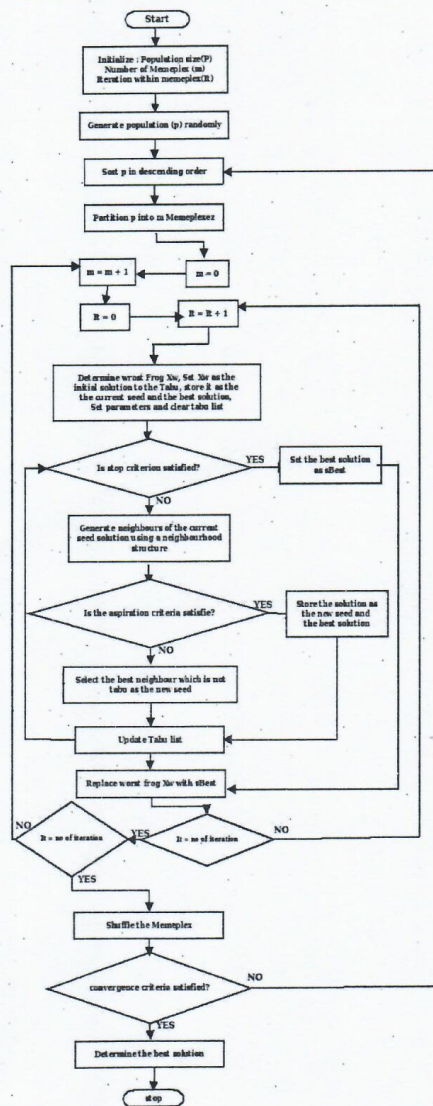


Figure 3: Flow Chart for the Hybrid SFLA-TABU Algorithm

The initial randomly generated population of frogs is sorted in a descending order of their fitness. Then, the entire population is partitioned into m memeplexes, each containing n frogs (i.e. $P = m \times n$). In this process, the first frog (the fittest frog) goes to the first memeplex, the second frog goes to the second memeplex, frog m goes to the m memeplex, and frog $m+1$ goes to the first memeplex, and so on.

In each memeplex, only the worst frog X_w is identified and it is used as the initial seed of the tabu search. The worst frog is improved using Tabu list and neighbourhood search approach of Tabu search algorithm. The Tabu list is modeled as a static circular FIFO queue of size 30 to facilitate tabu short-term memory. The neighbourhood $N(s)$ is defined by swapping any two elements of the $n \times 2$ array (frog). The number of neighbours of any solution was set to be problem specific, $2 \times$ numbers of WPs. The neighbours are evaluated based on the fitness function. The best solution is selected and the Tabu list is updated.

The solution returned by the Tabu search (i.e. the best neighbour) if found better than the worst frog replaces the worst frog. More specifically, the Tabu search is made to run for a specific number of iterations, q . If the attempt produces a new frog (X_{new}) that is better than the worst frog, it replaces it otherwise no replacement.

This operation is performed for a specific number of iteration (q). The number of iteration q here determines the time spent for local meme transference and in turn the efficiency of the local search. Since there is no generally agreed number of evolutionary iterations for a given SFLA local search, q was chosen to be dependent on the problem size with the value $q = 2n$, where n is the number of frogs in a memeplex as used in [2] and proved efficient.

After series of parallel local searches in the memeplexes, the evolved frogs are shuffled for global information exchange and the population is resorted in descending order of their fitness. The process continues until stopping criteria is satisfied. This is the condition that must hold for the local search and global search of the SFLA to halt. Listing 3 itemised the steps (as pseudo-codes) followed in the implementation of the algorithm.

3.4 Experimental Setup

In this section the implementation details, parameters settings, dataset used and the performance measures considered are discussed.

All algorithms are implemented in JAVA programming language using Netbeans IDE development environment. Experiments were run on an intel core i5 processors of 2.26 GHz speed each and a RAM of 4GB size PC running windows 10.

Input: Population (p), no of memplex (p), iteration in memplex (It), neighbourhood structure ($N(s)$), Searchdepth (β)
 Output: Best solution (s)

1. Begin;
2. Generate random population of P frogs;
3. For each individual i in P : calculate fitness (f_i);
4. Sort the population P in descending order of their fitness;
5. Divide P into m memplexes;
6. For each memplex:
 - Determine the worst frog, X_w ; $X_{best} = X_w$;
 7. $tabuList = \{ \}$
 8. $tabuList.push(X_w)$
 9. While (not (stoppingCondition()))
 10. $SNeighborhood = getNeighbors(bestCandidate)$
 11. $bestCandidate = SNeighborhood.firstElement$
 12. for ($sCandidate$ in $SNeighborhood$)
 13. if ((not $tabuList.contains(sCandidate)$ and (fitness($sCandidate$) < fitness($bestCandidate$))))
 14. $bestCandidate = sCandidate$
 - end if end for
 15. if (fitness($bestCandidate$) < fitness($sBest$))
 - $sBest = bestCandidate$
 - end if
 16. $tabuList.push(bestCandidate)$
 17. if($tabuList.size > maxTabuSize$)
 - $tabuList.removeFirstElement()$
 - end if
 - End while
18. If (fitness($sBest$) < fitness(X_w))
- $X_w = sBest$
- End;
19. Combine (shuffle) the evolved memplexes;
20. Sort the population P in descending order of their fitness;
21. Check if termination criterion is satisfied;
30. End;

Listing 3: The Pseudocodes for the Hybrid SFLA-TABU Algorithm

For parameter setting, all parameters are tested experimentally using three randomly generated instances described in table 3. The population size is varied as 100, 150, 200 and 300. The number of memplexes is also varied as 5, 10 for each case of the population size. The parameters for the local Tabu search were set as follows: size of neighbourhood structure was set at 2 x number of WPs, the size of Tabulist was set to 30 and the stop condition was set as twice the number of evolutionary iteration ($2 \times It$). This variation of population size and memplex size is due to the fact that there is no generally acceptable criterion for choosing the most appropriate population size and number of memplexes for a given problem.

Table 3: Randomly generated Instances

Instances	No of activities	Dependencies	No of employees
7W5E	7	12	5
16W10E	16	22	10
25W12E	25	30	12

These parameters have great influence together with the number of evolutionary iterations on the performance of the algorithm. The number of evolutionary iterations per memplex is set to $2N$, where N is the number of frogs in each memplex as proposed by Oladele and Mojeed (2014).

Owing to the stochastic nature of the hybrid algorithm experiments on each instance were run 20 times and results are averaged. Table 4 presents the results of this preliminary experiment based on different combinations of parameters. Results are based on overall fitness values.

From the analysis of the preliminary experimental results as shown in the table 4, it could be deduced that the algorithm performs effectively on all considered instances with the best average fitness recorded in the large sized instance. It can also be observed that the lower the number of memplex, the better the average fitness of the solution as observed in all instances except in instance 16W10E where memplex size 10 produced 0.62 a value better than 0.72 produced by memplex size 5. Moreover, it is also observed that setting high evolutionary iteration values impact positively on the fitness of the result in all the instances as iteration value 300 produced the best results for all instances. So consequently, 300 and 5 were chosen for population size and number of memplex respectively.

Table 4: The preliminary fitness results of the Hybrid approach for the Randomly generated Instances

INSTANCE 1 (7W5E)		
No of Iterations	Memeplex size	fitness
100	5	0.87
	10	1.27
150	5	0.52
	10	0.89
200	5	0.48
	10	0.66
300	5	0.44
	10	0.60
INSTANCE 2 (16W10E)		
No of Iterations	Memeplex size	fitness
100	5	0.79
	10	0.88
150	5	0.62
	10	0.73
200	5	0.56
	10	0.60
300	5	0.75
	10	0.62
INSTANCE 3 (25W12E)		
No of Iterations	Memeplex size	fitness
100	5	0.27
	10	0.49
150	5	0.23
	10	0.32
200	5	0.16
	10	0.28
300	5	0.15
	10	0.22

For the actual experiments the dataset used are 12 publicly available SPSP instances obtained from an Instance Generator developed by alba [18]. The instance generator has been used previously in [19 -21]. The dataset is composed of different combination of task sizes, number of employee and skill sets. For simplicity the instances are grouped according to number of task referred to as Work Packages, group 1 involve 10 WPs, group 2 contain 20 WPs and group 3 contain 30 WPs. Skill set per each group as well as number of employees ranges from 5 to 10. Table 5 presents the description of the dataset.

To evaluate the effectiveness and efficiency of the hybrid algorithm in solving the PSP problem, its performance is compared with original SFLA and Tabu search algorithms based on the following performance measures:

- (i) **Hit rate [20]:** is the percentage out of the total number of runs that an algorithm produces a feasible and optimal solution. It measures the efficiency and robustness of the algorithm of interest. It is calculated as:

Table 5: Actual Experimental Instances

GROUP (Work Package)	Skill Employee	5	10
Group 1	5	10t_5e_5s	10t_5e_10s
	10	10t_10e_5s	10t_10e_10s
Group 2	5	20t_5e_5s	20t_5e_10s
	10	20t_10e_5s	20t_10e_10s
Group 3	5	30t_5e_5s	30t_5e_10s
	10	30t_10e_5s	30t_10e_10s

$$HitR(A) = \frac{(\text{no of successful runs of } A)}{(\text{total number of runs})} \times 100\% \quad (4)$$

- (ii) **Length [20, 21]:** is the average execution time in seconds for the algorithm of interest to produce a solution for the specific number of runs. This measure estimates the efficiency of the algorithm. It is calculated as:

$$Len(A) = \frac{\sum_{i=1}^N \text{execution time}_i}{N} \quad (5)$$

Where N is the number of runs

- (iii) **Average Fitness:** is the mean value of the overall fitness of the algorithm for the total number of runs. It measures the effectiveness of the algorithm and quality of solutions generated by the algorithm.

It is given as:

$$AvgFit(A) = \frac{\sum_{i=1}^N Fit(A)_i}{N} \quad (6)$$

Where N is the number of times the algorithm is run

Table 6: Results of Hit rate for SFLA-TABU, SFLA and Tabu search algorithms

Algorithms Instances	SFLA- TABU	SFLA	TAB U
10t 5e 5s	91%	70%	50%
10t 5e 10s	95%	65%	45%
10t 10e 5s	97%	76%	48%
10t 10e 10s	100%	80%	35%
20t 5e 5s	65%	45%	25%
20t 5e 10s	75%	0%	0%
20t 10e 5s	84%	40%	20%
20t 10e 10s	70%	25%	0%
30t 5e 5s	40%	0%	0%
30t 5e 10s	55%	10%	0%
30t 10e 5s	60%	30%	0%
30t 10e 10s	48%	0%	0%

4.0 RESULTS AND DISCUSSION

The results of hit rate for the hybrid SFLA-TABU, ordinary SFLA and Tabu search algorithms is given in table 6. Each algorithm is run 20 times to obtain the result.

It can be seen that SFLA-TABU was able to produce feasible results in all instances recording a highest hit rate of 100% in 10t_10e_10s instance and lowest value of 40% in 30t_5e_5s instance. Ordinary SFLA produced feasible solutions only in 9 out of the 12 instances considered with the highest hit rate of 80% in 10t_10e_10s instance and a lowest hit rate of as low as 10% in 30t_5e_10s instance. It could not produce feasible solutions in 20t_5e_10s, 30t_5e_5s and 30t_5e_5s instances. Tabu search could only produce feasible solutions in 6 out of the 12 instances covering all instances in the small sized instances with 10 tasks and 2 from middle sized instances with 20 tasks. It could not produce any feasible solution in instances with 30 tasks. It recorded the highest hit rate of 50% in 10t_5e_5s. With these results, SFLA-TABU outperformed the other algorithms in terms of hit rate across all the instances.

Figure 4 present a graphical representation of the results. From figure 4 it can be observed that the more the number of tasks the lower the hit rate. This is shown in all algorithms as the value diminishes when the number of tasks increases from 10 to 30. Also the influence of increased number of employees can be observed in figure 4 as the hit rate increases when the number of employee increases from 5 to 10 in almost all the three groups considered (10, 20 30 tasks). The influence of increased number of skill sets can be observed to cause a slight decrease in the hit rate of the algorithms in across the instances except for SFLA-TABU algorithm which show increase in about 55% of the instances

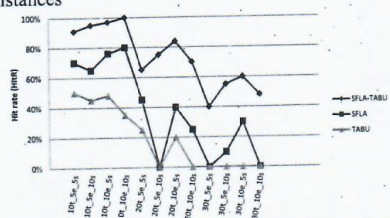


Figure 4: Hit Rate of SFLA-TABU, SFLA and Tabu search algorithms

Table 7 presents the results of average fitness of the algorithms on 20 runs. Algorithms that fail to produce feasible solution (hit rate equals 0%) have average fitness value to be NIL.

SFLA-TABU recorded the lowest average fitness values in all instances compared to ordinary SFLA and Tabu search except in 10t_10e_5s where SFLA has a lowest value of 0.77. SFLA-TABU produced the lowest fitness value of 0.65 in 30t_10e_5s instance, a result that shows the effectiveness of the algorithm in handling large sized project instances. As for the influence of increased number of employees, it is observed as show figure 5 that it caused a decrease in

the fitness value which translates into better performance.

Table 7: Average Fitness of SFLA-TABU, SFLA and Tabu search algorithms

Algorithms Instances	SFLA- TABU	SFLA	TABU
10t 5e 5s	0.91	1.11	0.95
10t 5e 10s	0.95	1.75	1.45
10t 10e 5s	0.87	0.77	0.90
10t 10e 10s	0.90	1.90	1.45
20t 5e 5s	1.65	2.65	3.25
20t 5e 10s	1.85	NIL	NIL
20t 10e 5s	0.94	1.90	4.20
20t 10e 10s	1.17	2.47	NIL
30t 5e 5s	1.40	NIL	NIL
30t 5e 10s	1.50	2.90	NIL
30t 10e 5s	0.65	0.75	NIL
30t 10e 10s	0.97	NIL	NIL

The case is the same for all the algorithms. Like the hit rate, increased number of skill sets translates into increased value of average fitness and consequently a reduced performance in terms of quality of solutions. It is worthy however to note that as the complexity of the problem increases (based on number of tasks) the average fitness increases in all algorithm across all instances except for SFLA-TABU that produced values in 30t_10e_5s and 20t_10e_5s that are lesser than at least a value in instances with 10 tasks.

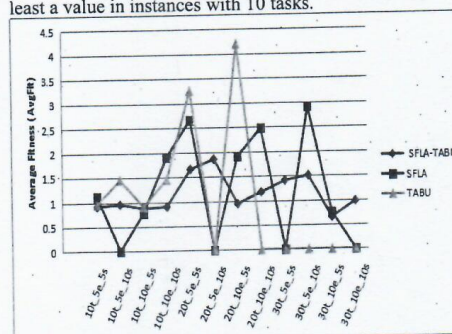


Figure 5: Average Fitness of SFLA-TABU, SFLA and Tabu search algorithms

Lastly, as a measure of efficiency, the results of average execution time (in minutes) of all the algorithms in all the instances are presented in figure 6. It can be seen that the hybrid SFLA-TABU algorithm consumes more time compared to ordinary SFLA and Tabu search in 7 out of the 12 instances. It is however important to note that despite this high running time, the hybrid SFLA-TABU algorithm managed to consume the least time in instances 20t_5e_5s, 30t_5e_10s and 30t_10e_5s.

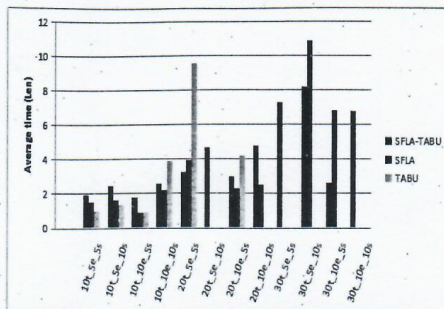


Figure 6: Average Execution time of SFLA-TABU, SFLA and Tabu search algorithms

Generally the proposed hybrid SFLA-TABU algorithm could produce feasible solutions with relatively high hit rate and high quality solutions if given enough time.

5.0 CONCLUSION AND FUTURE WORKS

In this study, a hybrid SFLA-Tabu search algorithm was applied to solve the problem of software project scheduling and staffing as a combined objectives to help project managers better simulate realistic schedules and manage human resources. The problem was modeled using DSM structure for schedules and binary representation for staffing. The hybrid algorithm employs the framework of SFLA with Tabu search incorporated as the local search within the framework. The performance of the proposed algorithm was evaluated on 12 publicly available standard problem instances. Evaluation results show that it outperforms original SFLA and Tabu search in all considered instances based on hit rate and quality of solution (measured as average fitness). It however consumes more time than the other two algorithms in about 60% of the problem instances considered. Our findings revealed that the algorithm can be applied to handle various sizes of software projects if given enough time. This work can be extended in the future through empirical evaluation and comparison of the proposed approach with other state of the art algorithms. Also other local search algorithms can be considered as a replacement for Tabu search algorithm within the framework.

REFERENCES

- [1] Ferruci F., Harman M., Ren J. and Sarro F. "Search-Based Software Project Management". In: Ruhe G., Wohin C., (eds). Software Project Management in a Changing World. 2014
- [2] Oladele, R.O. and Mojeed, H.A. "A Shuffled Frog-leaping Algorithm for optimal software project planning". African Journal of Computing & ICT. 7(1): pp 147-152. 2014
- [3] Karova, M., Petkova, J. and Smarkov, V. "A Genetic Algorithm for Project Planning Problem". Journal of Computer Engineering. 2(2):pp 66-68. 2008.
- [4] Patil, N., Sawanti, K., Warade, P., Shinde, Y. "Survey paper for software project scheduling and staffing problem". International journal of advanced research in computer and communication engineering. 2014.
- [5] Stylianou, C.S. and Andreou A.S. "Intelligent software project scheduling and team staffing with Genetic Algorithm". IFIP Advances in Information and Communication Technology (IFIPAICT), Vol. 364. 2011.
- [6] Gonsalves, T. and Itoh, K. "Multi-objective Optimization for Software Development Projects". Proceedings of International Multi Conference of Engineers and Computer Scientists 2010(I). 2010.
- [7] Vitekar, K.N., Dhanawe, S. A. and Hanchate, D.B. "Review of solving software project scheduling problem with Ant Colony Optimization". Inter-national journal of advanced research in electrical, electronics and instrumentation engineering. 2(4): pp 1177-1186. 2013.
- [8] Amiri, M., and Barbin, J.P. "New approach for solving software project scheduling problem using differential evolution algorithm". International Journal in foundations of Computer Science & Technology. 5(1):pp 1-5. 2015.
- [9] Eshraghi, A. "A new approach for solving resource constrained project scheduling problems using differential evolution algorithm". International Journal of Industrial Engineering Computations. 7(2): pp 205-216. 2016.
- [10] Eusuff, M.M., and Lansey, K.E. "Optimization of water distribution network design using the shuffled frog leaping algorithm". Journal of Water Resources Planning and Management. vol 129: pp 210-225. 2003.
- [11] Eusuff, M., Lansey, K., and Pasha, F. "Shuffled frog leaping algorithm: a memetic metaheuristic for discrete Optimization". Engineering Optimization. 38(2): 129-154. 2006.
- [12] Elbeltagi, E., Hegazy, T. and Garrison, D. "Comparison Among Five Evolutionary-Based Optimization Algorithms". Journal of Advanced Engineering Informatics, Elsevier Science. 19(1): pp 43-53. 2005.
- [13] Luo, J., Li, X. and Chen, M.R. "A novel hybrid shuffled frog leaping algorithm for vehicle routing problem with time windows". Journal of Information Sciences. 316: pp 266-292. 2014.
- [14] Zhang L., Wang L. and Zheng D. Z. "An effective hybrid genetic algorithm for flow shop scheduling with limited buffers". Journal of Computers and Operations Research. 33(10): pp 2960-2971. 2006.
- [15] Gerasimou, S., Stylianou, C. and Andreou, A. S. "An Investigation of Optimal Project Scheduling and Team Staffing in Software Development using Particle Swarm Optimization". In ICEIS (2) pp. 168-171. 2012.
- [16] Chen, W. N., and Zhang, J. "Ant colony optimization for software project scheduling and staffing with an event-based scheduler". IEEE Transactions on Software Engineering. 39(1) : pp 1-17. 2013.

- [17] Maenhout, B. and Vanhoucke, M. "A resource type analysis of the integrated project scheduling and personnel staffing problem". *Annals of Operations Research*. 252(2): pp 407-433, 2017.
- [18] Alba, E. and Chicano JF. Software project management with GAs. *Information Sciences*, 177 (11), pp 2380-401, 2007.
- [19] Crawford, B. Soto, R., Johnson, F., Monfroy, E. and Paredes F. A Max-Min Ant System algorithm to solve the Software Project Scheduling Problem. *Expert Systems with Applications*, 41, pp6634-6645, 2014.
- [20] Rachman, V. and M. Anwar Ma'sum. "Comparative analysis of ant colony extended and mix-min ant system in software project scheduling problem." In *2017 International Workshop on Big Data and Information Security (IWBIS)*, pp. 85-91. IEEE, 2017.
- [21] Xiao, J., Gao, M.L. and Huang, M.M. Empirical study of multi-objective ant colony optimization to software project scheduling problems. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 759-766, 2015.