HYBRID SFLA-TABU SEARCH ALGORITHM FOR OPTIMAL PROJECT SCHEDULING AND STAFFING

Mojeed H.A., Jimoh R.G., Sadiku P.O. & Salihu S.A.
Mojeed.ha@unilorin.edu.ng, Jimoh_rasheed@unilorin.edu.ng, peterposdoy@gmail.com, salihusa1980@gmail.com
Department of Computer Science, University of Ilorin

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 to solve the project scheduling and staffing problem with the two objective combined. The hybrid algorithm retains the framework of SFLA algorithm but employs the neighborhood structure method of tabu search and its avoidance of already explored area in the solution space to move towards optimal solution within the local memetic evolution. The algorithm was applied on three randomly generated problem instances representing small, medium and large sized problems. Results showed that the proposed algorithm was able to produce good optimal solutions with average fitness values 0.44, 0.56 and 0.15 in small, medium and large sized problems respectively. The hybrid algorithm outperformed the baseline algorithms in 100% of the problem instances and findings from the experiment revealed theoretically, the scalability of the proposed approach in handling various sizes of software project.

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

1.0 INTRODUCTION

Software Engineering, like other engineering disciplines, is concerned with optimization problems: it seek to build systems that are better, faster, cheaper, more reliable, flexible, scalable, responsive, adaptive, maintainable, and testable [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 a hybrid algorithm based on Shuffled frog-Leaping Algorithm (SFLA) and Tabu search algorithm is proposed to optimally solve the problem with the aim of improving the overall fitness of the solution.

1.1 Shuffled Frog-Leaping Algorithm

Shuffled Frog-Leaping Algorithm (SFLA) propo-sed 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 memeplexes after which a local search is performed within each memeplex. 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

$$(D_i) = rand\,().(X_b - X_w) \qquad (1)$$

New position

$$(X_n) = X_w + D_i, \qquad (D_{max} \geq D_i \geq -D_{max}) \qquad (2)$$

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 memeplex 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 pseudocode for SFLA is given below:

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 memeplexes;
6. For each memeplex:
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 memeplexes;
11. Sort the population P in descending order of their fitness;
12. Check if termination criterion is satisfied;
13. End;

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 neighbors of the current seed are then produ-ced by a neighborhood 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].
Below is the pseudocodes for the Tabu search:

Tabu Search (S_0, N(s), ß)
1: **Input:** Initial solution S_0, neighborhood N(s), search depth β
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 =: ß
17: **return** S_b

2. 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., 2008) 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 bstudy 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 conside-red not the employee experience in the formulation.

Amiri and Barbin [8] applied Differential Evolutio-nary 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 their 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 $=\{wp_1, wp_2, \ldots\ldots, wpn\}$ of tasks to be performed, a set of
dependency constraints DP = $\{(wp_i, wp_j), \ldots\ldots (wp_{in}, wp_{jn})\}$, 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-dimen-sional 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.

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 constraints

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. 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 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: Mathematical representation of employee assignments

Work Package	Employee assignment	Employee assignment representation (Binary equivalence)	Remarks
1	2	0010	Only employee 3, employee with id '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 will contain effective Work Package (WP) ordering and proper staff allocations. Each frog is encoded as an n x 2 array (that is, there are 2 columns per row). 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 (that is element, array[m][0], where m= {0, 1…, n}) represents a WP id and the value on the second column

(that is element, array[*m*][1], where *m*= {0, 1…, n}) represents employee assignment for that WP (whose binary equivalence is what is being actually used in the algorithm, as stated earlier). A typical frog schema 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: The schema of a Frog

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	Adetola Adebiyi	{Skill1, Skill2, Skill4}
2	Abdulsalam Mustapha	{Skill1, Skill2}
3	Taiwo Ogunjobi	{Skill3, Skill5}
	Work Place	
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}

The fitness of a frog 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 frog. 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 WP, W_{p_i}, if for each employee, e_i assigned to W_{p_i}: skill-set(e_i) ∩ required competence (W_{p_i}) = ϕ (that is e_i does not have any of the WP's required competence(s)) or if required competence (W_{p_i}) is not a subset of the set of the total competences of the team of employee(s) assigned to W_{p_i}. 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) = $\sum_{k=1}^{n}(V + M)$; fitness (A) >= 0 (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.

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

V (A) = 0

M (A) = 0

Fitness (A) = V (A) + M (A) = 0 (No violation of dependency and no skill mismatched allocation(s)).

3.3 Solution Approach

This study proposes 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 was 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 algorithm is as follows:

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* memeplexes;
6. For each memeplex:
   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( Xw) )
                    Xw = sBest
    End;
19. Combine (shuffle) the evolved memeplexes;
20. Sort the population P in descending order of their fitness;
21. Check if termination criterion is satisfied;
30. End;
```

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 x 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_{th} 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 neighborhood 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 neighborhood N(s) is defined by swapping any two elements of the n x 2 array (frog). The number of neighbors of any solution was set to be problem specific, 2 x numbers of WPs. The neighbors 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 neighbor) if found better than the worst frog replaces the worst frog. The hybridized SFLA-Tabu framework is depicted in figure 4. 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 by [2] Oladele and Mojeed (2014), as it 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. The solution converges when at least 1% of the frogs in the population have a fitness value of 0.

4.0 RESULTS AND DISCUSSION

The proposed hybrid SFLA-TABU for software project scheduling and staffing was tested on three randomly generated instances of software project scheduling problem described in the table 3.

The population size is varied as 100, 150, 200 and 300. The number of memeplexes is also varied as 5, 10 for each cases of the population size. The parameters for the local Tabu search were set as follows: size of neighborhood 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 x *It*). This variation of population size and memeplex size is due to the fact that there is no generally acceptable criteria for choosing the most appropriate population size and number of memeplexes for a given problem.

These parameters have great influence together with the number of evolutionary iterations on the performance of the algorithm. The number of evolutionary iterations per memeplex is set to 2*N*, where *N* is the number of frogs in each memeplex as proposed by Oladele and Majeed (2014). Owing to the stochastic nature of proposed hybrid algorithm experiment on each instance is run 20 times and results are averaged. Table 4 presents the results of the proposed hybrid algorithm based on different combinations of parameters as explained above.

From the analysis of the experimental results as shown in the table 4, it could deduced that the algorithm performs effectively on all considered instances with the best average fitness recorded the large size instances. It can also be observed that the lower the number of memeplex, the better the average fitness of the solution as

observed in all instances except in instance 16W10E where memeplex size 10 produced 0.62 a value better than 0.72 produced by memeplex 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. It is worthy to note that our approach aided by the problem formulation produces 2, 4 and 5 best solutions with no skill mismatch allocation and tasks allocated to most experience employee in the set of employees available for 7W5E, 16W10E and 25W12E instances respectively.

These results show that the proposed algorithm can conveniently produce optimal solutions when the project scheduling and staff allocation objectives are combined contrary to the study of [5] Stylianou and Andreou (2011) which employed Genetic Algorithm.

To evaluate the effectiveness of the proposed algorithm, its performance is compared with baseline algorithms. As a preliminary sanity check, the result from our formulation is compared with pure random search. If a proposed formulation or algorithm does not allow an intelligent computational search technique to outperform random search convincingly, then there is clearly something wrong with the formulation.

To achieve this, the best performing combinations of parameters for the hybrid approach was selected based on the results in table 4. Table 5 presented the result of the comparism. From the result it can be observed that the proposed approach completely outperformed random search in all instances.

It is not enough to evaluate an algorithm based on its performance with random search as this only form a preliminary basic test. There is a need compare the algorithm with other baseline approaches. Therefore, to further evaluate the performance of the algorithm, its result was also compared with original SFLA and Tabu search algorithms. Each of the algorithms is run 20 time and average fitness is determined. The result of the comparism is presented in table 6. The best performing combinations of parameters for the hybrid approach was also selected.

Table 5: SFLA-Tabu compared with Random Search

INSTANCE	**AVERAGE FITENSS**	
	SFLA-TABU	RANDOM SEARCH
7W5E	0.44	4.50
16W10E	0.75	3.14
25W12E	0.15	2.94

Table 6: Results of Hybrid Approach, SFLA and TABU search

INSTANCE	**AVERAGE FITENSS**		
	SFLA-TABU	SFLA	TABU
7W5E	0.44	1.51	1.40
16W10E	0.75	1.73	1.85
25W12E	0.15	1.62	3.15

From the results, the hybrid algorithm significantly outperformed both SFLA and Tabu search in all instances in terms of average fitness with the best result in large sized project. This revealed the scalability of the proposed algorithm in handling different sizes of software development projects. Next to the proposed approach in performance is SFLA as it performed better than Tabu search in medium and large sized project i.e (16W10E and 24W12E). Tabu search however performed better than SFLA in 7W5E instance. This result uncovers the theoretical findings on the effective-ness of the proposed hybrid algorithm.

1.0 CONCLUSION AND FUTUREWORKS

In this study, the problem of software project scheduling and staffing was solved as a combined objectives using a hybrid SFLA-Tabu search algorithm 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 proposed algorithm was applied to three randomly generated problems. The proposed algorithm was able to find optimal solution with no skill mismatch. Evaluation results show that it outperforms random search, original SFLA and Tabu search in all considered instances. Also, findings revealed that the algorithm is robust and scalable in handling various sizes of projects. This work can be extended in the future through empirical evaluation of the proposed approach by applying it on real life data or instances. 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 Inform-ation 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 Optimiza-tion. 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 optimiza-tion for software project scheduling and staffing with an event-based scheduler"*. IEEE Transac-tions 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.