A Hybrid ant Colony Tabu Search Algorithm for Solving Next Release Problems

Balogun Abdullateef Oluwagbemiga, Basri Shuib, Said Jadid Abdulkadir, Garba Mariam, AlmomaniMalek Ahmad Thabeb

Abstract: ***Next Release Problem (NRP) is a challenge in software engineering to define which set of requirements are to be developed in the next release of a software product taking in consideration several constraints such as the cost of development, user's significance, and constraints related to scheduling, dependencies between requirements and available expertise. Solving this problem will help software engineers to make decisions on the set of requirements to include as features in the next release of a software product. This paper proposes a hybrid of Ant Colony Optimization (ACO) algorithm and Tabu Search (TS) for solving NRP using a cost-value model for requirements. A fitness function with two objectives was considered to maximize users' satisfaction and to minimize the cost of developing the requirements requested by users. The hybrid Ant Colony Optimization Tabu Search (ACOTS) algorithm is based on Ant Colony Optimization (ACO) algorithm while it employs the history keeping strategy of Tabu Search (TS) when constructing new solutions (local search spaces) for each initial solution generated by each ant. The procedure of the hybrid algorithm starts by generating random solutions that serve as a pivot for all ants of the colony which is based on the pheromone information, the set objectives in the fitness function and problem specific local heuristic information associated with each of the objectives. The output of the hybrid ACOTS is a set of promising optimal values which are the total number of the set of requirements from which a subset is to be selected. The results of the experiments showed that the application of ACOTS yielded larger and better sets of results than existing methods (ACS, Ant System and Tabu Search). The application of ACOTS also enables an easier parameter tuning (budget, number of requirements).***

Keywords: Requirement Engineering, Next Release Problem, Metaheuristics, Optimization

Revised Manuscript Received on March 08, 2019.

Balogun Abdullateef Oluwagbemiga, Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak, Malaysia. Department of Computer Science, University of Ilorin, PMB 1515, Ilorin, Nigeria. abdullateef_16005851@utp.edu.my, balogun.ao1@unilorin.edu.ng
Software Quality and Quality Engineering (SQ^2E) Research Cluster, Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak, Malaysia. Malaysia.

Basri Shuib, Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak, Malaysia. Software Quality and Quality Engineering (SQ^2E) Research Cluster, Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak, Malaysia. shuib_basri@utp.edu.my

Said Jadid Abdulkadir, Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak, Malaysia. Centre for Research in Data Science (CERDAS), Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak, Malaysia.

Garba Mariam, Department of Computer Science, University of Ilorin, PMB 1515, Ilorin, Nigeria.

AlmomaniMalek Ahmad Thabeb, Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak, Malaysia. Software Quality and Quality Engineering (SQ^2E) Research Cluster, Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak, Malaysia.

I. INTRODUCTION

Software engineering is the application of a systematic approach to the development, operation, and maintenance of software(Srivastava, Varshney, Nama, & Yang, 2012). The systematic approach has made software engineering challenges due to its tractable medium and periodic requirement change. These requirements must be quantifiable, relevant and detailed as they are the new features users are expecting to see(Buede & Miller, 2016; Dick, Hull, & Jackson, 2017). Requirement analysis is used as the communication link with the software users to gather their respective expected features from an upgrade or a new solution. In other to minimize production cost and development time, software companies need to optimal satisfy users' requirements since it is unrealistic to develop all requirements demanded by the users(Braude & Bernstein, 2016; Dick et al., 2017). The optimization of software requirements is a relevant task in software engineering as well as agile methodologies, in which software products are released within a short iteration period(Layton & Ostermiller, 2017). A new set of requirements with defined development costs is generated at each iterative cycle to satisfy the demands of the users. This problem of which set of requirements to be developed in accordance to some constraints is regarded to as the Next Release Problem (NRP)(Almeida, Pereira, Reis, & Piva, 2018; L. Li, Harman, Wu, & Zhang, 2016).

NRP is an NP-Hard Problem which makes it a complex optimization problem(Almeida et al., 2018). Quite a number of approaches and algorithms have been used or proposed to find a solution to NRP. Exact methods such as integer linear programming have been deployed for such problem(Almeida et al., 2018; Veerapen, Ochoa, Harman, & Burke, 2015), also genetic algorithm such as Non-Dominated Sorting Algorithm (NSGAII) (Deb, Pratap, Agarwal, & Meyarivan, 2002) and approximate methods(Almeida et al., 2018; Cai, Cheng, Fan, Goodman, & Wang, 2017). The NRP can also be seen as a Feature Subset Selection (FSS) Problem where an optimal subset of relevant requirements will be selected in the formulation of a model (Araújo, Paixao, Yeltsin, Dantas, & Souza, 2017; Y. Li, Zhang, Yue, Ali, & Zhang, 2017).

Ant Colony Optimization algorithm (ACO) is one of the new technologies in approximately solving NP-hard

problems(Doerner & Maniezzo, 2018). With a colony of artificial ants, ACO can achieve good results for optimization problems. To improve the performance of ACO, hybrids of ACO have been proposed for solving classical optimization problems(Dorigo & Stützle, 2019; Engin & Güçlü, 2018).

Motivated by the successes of ACO in tackling NP-hard problems, this paper proposes a hybrid of ACO and Tabu Search (TS) for solving the NRP adapting the cost-value model for requirements by(Zhang, Harman, Ochoa, Ruhe, & Brinkkemper, 2018)and a real-world RALIC datasets(Finkelstein & Lim, 2012; Lim, 2011).

Several heuristics approaches for solving the NRP already exist among those of which are the Hill Climbing (HC) and Simulated Annealing (SA)(Zhang et al., 2018). Hybrid Ant Colony Optimization algorithm with Hill Climbing was also proposed to solve typical NRP instances which gave good results but their study only considered the classical datasets (Jiang, Zhang, Xuan, Ren, & Hu, 2010). Balogun, Mabayoje, Makinwa, and Bajeh (2016) also applied a hybrid of Variable Neighbourhood Search (VNS) and Tabu Search (TS) for solving bi-objective NRP, using a cost-value model for requirements with the experiments showing that the VNSTS is a 'good enough' algorithm for application on the NRP.

II. RELATED WORKS

Search Based Software Engineering

Search Based Software Engineering (SBSE) is an approach in software or requirement engineering which tends to reformulate Software Engineering problems as search problems, this was originally coined by Harman and Jones in 2001(Bagnall, Rayward-Smith, & Whittley, 2001). SBSE sought for optimal or near-optimal solutions from a search space of candidate solutions which is usually guided by fitness function which separates better solutions from worse solutions.

The following are the factors that make Software Engineering suitable to be reformulated as search problems:

1. The need to optimize conflicting objectives.
2. The need to balance competing objectives.
3. Variations of possible solutions.
4. It is impractical to get an optimal solution.
5. Variations of optimal and near-optimal solutions can be achieved.

SBSE in Requirement Analysis and Selection

The success values of a software system or product rely on how well considers the needs of its users in the form of software requirements. Requirements Engineering (RE) is the process through which requirements are elicited, modeled, analyzed, and documented(Almeida et al., 2018; Durillo, Zhang, Alba, Harman, & Nebro, 2011). Most software companies need to keep their product users happy by proficiently satisfying huge sets of requirements with respect to minimizing constraints (time and cost). It is unrealistic to develop for the software company to consider all requirements submitted by their users at once. A software engineer is therefore faced with the challenge of selecting a smaller set of these requirements that can be adequately developed with the available resources at a time(Y. Li et al., 2017; van den Akker, Brinkkemper, Diepen, & Versendaal, 2005).

By its nature, selecting a set of requirements to be developed from a pool of requirements in the release of the software is a complex problem as some factors must be considered. One of such factors is that all users want all their requirements to be developed in the system, but the effect of the cost of those requirements on the budget is fundamentally not in favor with the goal of software engineering(Veerapen et al., 2015). Each user has a different level of importance to the software company; also requirements have different levels of importance to users. Marketing concerns can also make the organization to want to satisfy their newest user, or they may want every user to have at least one of their requirements developed. Another factor is the interactions that might be present among users' requirements, that is, the fulfillment of one requirement affects the fulfillment of another(Zhang et al., 2018). This kind of interaction could be in different form of dependency; two requirements might be conflicting such that only one of them can be fulfilled (OR dependency), or the fulfillment of a requirement means another particular requirement must be fulfilled (AND dependency), or a particular requirement that has to be developed before another particular one (precedence), or a particular requirement that when it is developed, it affects the value of another requirement (value-related)(Araújo et al., 2017; Bagnall et al., 2001).

Bagnall et al. (2001) standardized the application of SBSE to requirements engineering as the Next Release Problem (NRP), proposing a model where each user has a *value* to the software company and demands the implementation of a set of features in the software system. This model aims to select a subset of users to be satisfied such that the sum of their *values* is maximized(Almeida et al., 2018). Users whose requirements are developed in the software system are considered satisfied. van den Akker et al. (2005)offered a modification to this model where requirements are seen to have importance rather than users. The importance value for each requirement is gotten by aggregating the importance values and number of the customers that require it.

Finkelstein and Lim (2012) and Lim (2011)created a way to determine the importance value of users (stakeholder) and hence, the aggregate importance of requirements by using social networks of stakeholders and collaborative filtering. The method called StakeRare picks out users, who are also asked to recommend other users and their roles, creates a network using users as nodes and their recommendations as links, and orders users using different network measures to generate a value for their influence on the project. It then asks the users to give ratings to requirements in a list, recommends other requirements that might be relevant to them using collaborative filtering, and orders their requirements by the use of their ratings whose weight is generated from their influence on the project.

Zhang et al. (2018) and Jiang et al. (2010)noted that while previous works have formulated the NRP as having a single objective, in the real world, NRP could have multiple

objectives due to the competing and conflicting nature of requirements, and gave a formulation of the NRP where two objectives – to maximize the total value of requirements or users' satisfaction and to minimize the cost of developing them, are considered.

del Sagrado, del Aguila, and Orellana (2015)also applied the Ant Colony Optimization algorithms to the NRP, while comparing results gotten with results from applying Greedy Randomized Adaptive Search Procedure (GRASP) and the Non-dominated Sorting Genetic Algorithm (NSGA-II). From their results, it was found out that the multi-objective ACS arrived at a better solution sets than NSGA-II and GRASP. ACO has also been hybridized with Hill Climbing (HC) in solving NRP while the results were compared with

GRASP, Simulated Annealing (SA), HC and Ant Colony Optimization. The conclusion of the research was that the hybridized ACO can achieve better solution quality than ACO Jiang et al. (2010).

III. THE SOLUTION APPROACH

The two important components of SBSE are the problem representation which is the model and the fitness function(Harman & Jones, 2001). This section will be on the problem representation (cost-value model for the NRP), the fitness function, and the description of the components of the hybrid ACO and Tabu Search on how it will be applied to NRP.

The Cost Value Model

The cost-value NRP model which was proposed by Zhang et al. (2018) is adopted in this paper.

Let equation (i) be a set of Users {U} for an existing software system:

$$U = \{u_1, \ldots, u_x\} \quad (1)$$

whose set of Requirements {R} are to be factored in the development of the next release of a software product.

Let equation (ii) be their set of requirements:

$$R = \{r_1, \ldots, r_y\} \quad (2)$$

For each set of requirements $r_a(1 \le a \le y)$, there is an associated Cost for its development represented as:

$$Cost = \{cost_1, \ldots, cost_z\} \quad (3)$$

Also, each User has significance to the organization which is usually associated with a weight value. The set of the associated weight of each user $u_b(1 \le b \le x)$ is:

$$Weight = \{w_1 \ldots w_m\} \quad (4)$$

where $w_j \in [0, 1]$.

An assumption is made that not all requirements are of equal importance to a given user. The rate at which a given user is satisfied is dependent on the set of feature or requirements that are developed in the next release of the software. The overall users' satisfaction provides value for the organization.

Each user $u_j(1 \le j \le x)$ associates a value to requirement $r_i(1 \le i \le y)$ which signifies their degree of "need" for the requirement which is given by value(r_i, u_j). A value of (r_i, u_j) > 0 means user *j* has requested for the development of requirement *i* in the next release of the software product. From the above, the overall importance or score of a particular requirement $ri(1 \le i \le n)$ can be computed by:

$$score_i = \sum_{j=1}^{m} w_j \ .value\left(r_i, u_j\right) \quad (5)$$

The '*score*' of a given requirement is the overall '*value*' for the organization. A software engineer seeks a result which is a decision vector that determines the set of requirements that will be developed. In the decision vector $x = \{x_1. \ldots x_n\} \in \{0, 1\}$, the value 1 for $x_i$means requirement *i* is chosen, and value 0 means otherwise.

The Fitness Function

This paper adopted fitness functions from(Zhang et al., 2018) and (Durillo et al., 2011)for the Multi-Objective NRP (MONRP). Two objectives are considered to maximize users' satisfaction (which is equivalent to the value for the organization) and to minimize the cost of developing the requirements requested by users. It must be noted that the second objective used in this study is acost which would be used to help in exploring better solutions.

The objective function for maximizing the overall value is stated below:

$$Maximize \sum_{i=1}^{n} score_i \ .x_i \quad (6)$$

This means the selection of a subset of users' requirements that gives a maximized value for the organization. Equation (vii) represents the objective function to minimize the total cost of requirements selected:

$$Minimize \sum_{i=1}^{n} c_i \ .x_i \quad (7)$$

To standardize this objective, the sum of all the costs is multiplied by -1. The objective to optimize in the fitness function is now stated below:

$$Maximizef_1(\vec{x}) = \sum_{i=1}^{n} score_i \ .x_i \quad (8)$$

$$Maximizef_2(\vec{x}) = -\sum_{i=1}^{n} c_i \ .x_i \quad (9)$$

A. *The Algorithm*

This section describes the components that make up the hybrid algorithm, the expected input format, and the expected output format.

Ant Colony Optimization

Ant colony optimization (ACO) is a nature-inspired algorithm developed based on the behavior and characteristics of ants(Dorigo & Stützle, 2019; Salhi, 2017). Ants are simple insects that in colonies with outstanding cooperative behavior for survival. They communicate via pheromones which are chemical substances ants lay down in varying quantities to mark, for example, a path. Ordinarily, ants move randomly but they can detect and follow a previously laid pheromone trail. Ants moving this way can also lay down pheromone for others which may result to a feedback mechanism; if a pheromone trail is followed by more ants, some other ants will follow the same trail in the future(Dorigo & Stützle, 2019). ACO operate on this mechanism of pheromone trail.

In general, how the pheromone trails can be used to find better solutions in subsequent iterations is a key aspect of ACO. That is, by combining solution components that have been found to have good solutions in previous iterations, even better solutions may be found(McMullen, 2017). The aforementioned characteristics make ACO be adaptive in nature by learning from past experience(s). Local search algorithms are used with ACO to improve its performance and consequently better solution(s)(Jiang et al., 2010; Poongothai & Rajeswari, 2016; Wu, Wu, & Wang, 2017). Hybrids of ACO by combining its probabilistic solution construction with a local search algorithm is said to be efficient(Drias, Kechid, & Pasi, 2016). These locally optimal solutions found by the local search algorithm are eventually used to provide a positive feedback and help in directing the search towards other promising solutions in the search space.

The ACO algorithm is defined below:

Procedure *Ant Colony Optimization*
    Initialize pheromone trails, calculate heuristic information
    while(termination condition not met) **do**
        p = **ConstructSolutions**(*pheromone trails, heuristic information*)
        p = **LocalSearch**(*p*) (*% optional)*
        GlobalUpdateTrails(*p*)
    end
end*Ant Colony Optimization*

Listing. 1 Ant Colony Optimization algorithm(Dorigo & Stützle, 2019)

Tabu Search

Tabu Search (TS) was originally proposed by(Glover, 1990) for solving combinatorial problems in operations research. It is as an extension of typical local search methods. According to (Gendreau & Potvin, 2019), Tabu Search is a strategy for controlling and guiding "inner" heuristics particularly tailored to solving combinatorial problems. The major concept behind Tabu Search is that it selectively keeps the states encountered in memory, called *tabu-list*. The *tabu-list* serves as a guide in performing an embedded search so that areas in the search space that were recently visited are no longer visited, that is, it prevents cycling by keeping a history of states that have been previously visited. This is achieved by keeping changes in recent moves within the search space and preventing future moves from reverting those changes. A Tabu Search algorithm could keep a short-term, intermediate-term or long-term memory(Glover, Laguna, & Martí, 2018). The history *H* kept on the tabu-list is used to replace the neighborhood of the current solution *N(s)* with a modified neighborhood, which could be denoted *N(H,s*) and *N(H,s)* is a subset of *N(s)*. Therefore, the history *H* determines the solutions that could be reached by performing a move from the current solution, selecting another solution s_0 from *N(H,s)*. It is assumed that elements in *N(s)* that are not in *N(H,s)* which the algorithm seeks to identify are possible high-quality local optima at different points in the search space. The history is used as part of a way of evaluating the currently accessed solutions, that is, Tabu Search replaces the fitness function *f(s)* with another function *f(H,s)*(Gendreau & Potvin, 2019). In instances where *N(H, s)* could contain many elements, it may be costly to examine the entire neighborhood of these elements, Tabu Search then uses a strategy of isolating candidate subset of the neighborhood for examination(Glover et al., 2018).

The TS algorithm is defined below:

Sbest ← ConstructInitialSolution()
TabuList← { }
while (Not StopCondition())
    CandidateList ← { }
    for(S*candidate*∈Sbest*neighbourhood*)
        if (Not ContainsAnyFeatures(S*candidate*, TabuList))
            CandidateList ← S*candidate*
        end if
    end for
    S*candidate* ← LocateBestCandidate(CandidateList)
    if (Cost(S*candidate*) ≤ Cost(Sbest))
        Sbest= S*candidate*
        TabuList = FeatureDifferences(S*candidate*, Sbest)
        while(TabuList>Tabulist*size*)
            DeleteFeature(TabuList)
        end while
    end if
end while
Return (Sbest)

Listing. 2 Algorithm for Tabu Search (TS) (Gendreau & Potvin, 2019)

VI. RESULT AND DISCUSSION

A. Proposed Hybrid Algorithm

The main body of the hybrid is made up of the main body of the Ant Colony Optimization (ACO) algorithm while it employs the history keeping strategy of Tabu Search (TS) when constructing new solutions (local search spaces) for each initial solution generated by each ant.

The procedure starts by generating random solutions that serve as apivot for all ants of the colony which is biased by the pheromone information, the set objectives in the fitness function used in this study and a problem specific local heuristic information associated with each of the objectives. The two objectives considered are value and cost, hence the solution approach is tailored to this such that two sorts are created, one by value and the other by cost. Then several variations are made between these sorts to generate new solutions by the ants, treating the cost in the pivot as a constraint such that, one of the solutions generated by the ants which has a better value (high pheromone) without exceeding the cost is seen as a local optima and added to the set of promising solutions. The solution which has a lesser value (low pheromone) is added to the tabu list so that such

solutions can be avoided while creating a new set of solutions to avoid cycling. Finally, the pheromone values associated with the set of promising solutions high pheromone) are increased while that associated with non-promising solutions (low pheromone) are decreased (pheromone evaporation) thereby updating the pheromone trails globally. This procedure is then repeated, taking each solution generated as a pivot for generating new solutions while adding each local optimum gotten to the set of promising solutions. This procedure is guaranteed to have a solution consisting of elements whose number is at least the number of solutions initially generated. This number is one of the parameters that would be set by the researcher. This means that the software engineer can set as a parameter the minimum number of elements in the set of solutions. It should be noted that the goal of this hybrid is to serve as a decision tool for the decision maker by providing a wide range of optimal or near-optimal solutions, with trade-offs between the set objectives from which an informed decision can be made, and not to provide a non-promising solution as it is with the ACO and the TS individually which is the strength of this hybrid. The output of this algorithm is a set of promising optimal solutions. The elements of the solution set are the total number of the set of requirements from which a subset is to be selected. The values of these elements are those of the corresponding requirement that is selected.

The hybrid algorithm is defined as follows:

Initialisation:

M ← No of ants
PromisingSolutions← { }
PoorSolutions← { }
TabuList ← { }
K ← No of iterations

The body:

for i ← 1 to K
for j ← 1 to M
//Solution Construction Phase
Ant M generates solution S*j*, **for** j = 1,... m
LocalOptimum ← S*j*
//Local Search Phase
Solution ← GenerateSolutions(TabuList,S*j*)
if (fit(Solution) > fit(S*j*))
PromisingSolutions[]← LocalOptimumSolution
else
TabuList[ ] ← difference(S*j*, Solution)
PoorSolutions[ ] ← TabuList[]
Endif
Endfor
//Global Pheromone Update Phase
EvaporatePheromone(PromisingSolutions[],PoorSolutions[])
Endfor
Return (PromisingSolutions)

Listing. 3 Algorithm for the hybrid ACOTS

To determine the interrelated hybrid ACOTS parameter settings, CALIBRA procedure (Adenso-Diaz & Laguna, 2006)was used to determine the interrelated ACOTS parameter settings that will improve the algorithm's performance. CALIBRA is an automated tool for finding performance optimizing parameter settings, which helps algorithm designers from manually searching the parameter space(Kleijnen, 2015). It uses Taguchi's fractional factorial experimental designs along with local search to fine-tune algorithm parameters which has a wide range of possible values. The benefit of using CALIBRA can be seen in situations where the algorithm that is being fine-tuned has parameters whose values have a significant impact on performance(Stützle & López-Ibáñez, 2019).

The hybrid algorithm (Ant Colony Optimization Algorithm and Tabu Search – ACOTS),Ant Colony System (ACS) and Ant System (AS) require the following as input to the algorithm:

1. α – An integer number, which determines the relative importance of pheromone value.
2. β – An integer number, which is used for constructing the heuristic information.
3. ρ – An integer number ranging between 0 and 1, which determines the evaporation rate for favoring the exploration of the new solution in the search space.
4. h – An integer number, representing the number of ants to be used in each iteration.
5. P – An integer number, which is the probability for an ant to choose a solution.
6. D – An integer number containing a structured list of requirements containing value and cost for each requirement.

The Tabu Search algorithm requires the following as input to the algorithm:

1. P – An integer number, which is the probability for a solution to be selected.
2. Tabu-list – An integer number, representing a list of possible moves that could be performed on a solution.

In each run of the algorithm, the cost and value of all the requirements, as well as users satisfaction, is calculated for each set of result. Each set of result contains a set of requirements which will be developed in the next release of the software product.

Table. 1 Experiment 1 with hybrid Ant Colony Tabu Search algorithm (ACOTS)

Algorithm: Hybrid ACOTS
Algorithm Parameters:α = 1.1, β = 1.5, ρ = 0.13, h = 10, P = 0.1, d = 40 (total number of requirements in the list having cost)

Instance	Run	No of requirements in results	Average Cost	Execution Time (ms)
NRP – 0.3 / 0.5 / 0.7	1	26 / 31 / 35	1460.0 / 2434.0 / 3407.0	499.0 /470.0 / 514.0
NRP – 0.3 / 0.5 / 0.7	2	27 / 32 / 35	1460.0 / 2434.0 / 3407.0	261.0 /401.0 / 263.0
NRP – 0.3 / 0.5 / 0.7	3	26 / 31 / 35	1460.0 / 2434.0 / 3407.0	381.0 /384.0 /405.0
NRP – 0.3 / 0.5 / 0.7	4	26 / 31/ 35	1460.0 / 2434.0 / 3407.0	469.0 / 358.0 /438.0
NRP – 0.3 / 0.5 / 0.7	5	26 / 31 / 35	1460.0 / 2434.0 / 3407.0	489.0 /498.0 /413.0
NRP – 0.3 / 0.5 / 0.7	6	27 / 31 / 35	1460.0 / 2434.0 / 3407.0	393.0 /466.0 / 421.0
NRP – 0.3 / 0.5 / 0.7	7	26 / 32 / 34	1460.0 / 2434.0 / 3407.0	418.0 / 668.0 /438.0
NRP – 0.3 / 0.5 / 0.7	8	26 / 31 / 35	1460.0 / 2434.0 / 3407.0	523.0 / 367.0 /437.0
NRP – 0.3 / 0.5 / 0.7	9	26 / 31 / 35	1460.0 / 2434.0 / 3407.0	329.0 / 456.0 /372.0
NRP – 0.3 / 0.5 / 0.7	10	27 / 32 / 35	1460.0 / 2434.0 / 3407.0	418.0 / 528.0 /517.0

Table. 2 Experiment 2 with Ant Colony System (ACS)

Algorithm: ACS
Algorithm Parameters: α = 1.1, β = 1.5, ρ = 0.13, h = 10, P = 0.1, d = 40 (total number of requirements in the list having cost)

Instance	Run	No of requirements in results	Average Cost	Execution Time (ms)
NRP – 0.3 / 0.5 / 0.7	1	22 / 30 / 33	1460.0 / 2434.0 / 407.0	173.0 / 169.0 / 237.0
NRP – 0.3 / 0.5 / 0.7	2	22 / 30 / 34	1460.0 / 2434.0 / 3407.0	235.0 / 254.0 / 354.0
NRP – 0.3 / 0.5 / 0.7	3	23 / 28 / 33	1460.0 / 2434.0 / 3407.0	291.0 / 183.0 / 170.0
NRP – 0.3 / 0.5 / 0.7	4	21 / 27/ 33	1460.0 / 2434.0 / 3407.0	261.0 / 238.0 / 240.0
NRP – 0.3 / 0.5 / 0.7	5	23 / 27 / 33	1460.0 / 2434.0 / 3407.0	119.0 / 272.0 / 197.0
NRP – 0.3 / 0.5 / 0.7	6	24 / 30 / 31	1460.0 / 2434.0 / 3407.0	108.0 / 197.0 /366 .0
NRP – 0.3 / 0.5 / 0.7	7	22 / 27 / 34	1460.0 / 2434.0 / 3407.0	126.0 / 121.0 /343 .0
NRP – 0.3 / 0.5 / 0.7	8	24 / 29 / 32	1460.0 / 2434.0 / 3407.0	97.0 / 440.0 / 189.0
NRP – 0.3 / 0.5 / 0.7	9	22 / 27 / 33	1460.0 / 2434.0 / 3407.0	201.0 / 208.0 / 442.0
NRP – 0.3 / 0.5 / 0.7	10	22 / 27 / 33	1460.0 / 2434.0 / 3407.0	167.0 / 238.0 /125 .0

Table. 3 Experiment 3 with Ant System (AS)

Algorithm: AS
Algorithm Parameters: α = 1.1, β = 1.5, ρ = 0.13, h = 10, P = 0.1, d = 40 (total number of requirements in the list having cost)

Instance	*Run*	*No of requirements in results*	*Average Cost*	*Execution Time (ms)*
NRP – 0.3 / 0.5 / 0.7	1	25 /31 / 35	1460.0 / 2434.0 / 3407.0	184.0 /137 .0 /211 .0
NRP – 0.3 / 0.5 / 0.7	2	25 / 31/ 35	1460.0 / 2434.0 / 3407.0	396.0/ 204.0 /380 .0
NRP – 0.3 / 0.5 / 0.7	3	25 / 30 / 35	1460.0 / 2434.0 / 3407.0	385.0/ 112.0/ 259.0
NRP – 0.3 / 0.5 / 0.7	4	24 / 31 /35	1460.0 / 2434.0 / 3407.0	448.0 /209 .0 / 0
NRP – 0.3 / 0.5 / 0.7	5	26 /31 / 35	1460.0 / 2434.0 / 3407.0	289.0 / 343.0 /237 .0
NRP – 0.3 / 0.5 / 0.7	6	25 / 30 / 34	1460.0 / 2434.0 / 3407.0	237.0 / 328.0 / 540.0
NRP – 0.3 / 0.5 / 0.7	7	24 / 29 / 35	1460.0 / 2434.0 / 3407.0	208.0 / 122.0 / 337.0
NRP – 0.3 / 0.5 / 0.7	8	24 / 30 / 35	1460.0 / 2434.0 / 3407.0	312.0 /326 .0 / 290.0
NRP – 0.3 / 0.5 / 0.7	9	25 / 31/ 35	1460.0 / 2434.0 / 3407.0	393.0 /177 .0 / 423.0
NRP – 0.3 / 0.5 / 0.7	10	26 / 31/ 35	1460.0 / 2434.0 / 3407.0	339.0 / 322.0 /284 .0

Table 4 - Experiment 4 with Tabu Search (TS)

Algorithm: TS
Algorithm Parameters: P = 0.1, tabulist= 5, d = 40 (total number of requirements in the list having cost)

Instance	*Run*	*No of requirements in results*	*Average Cost*
NRP – 0.3 / 0.5 / 0.7	1	6 / 6 / 9	833.0 / 1306.0/ 1476.0
NRP – 0.3 / 0.5 / 0.7	2	6 / 5 / 9	744.0 / 1062.0 / 1607.0
NRP – 0.3 / 0.5 / 0.7	3	7 / 9 / 8	1144.0 / 1429.0/ 1353.0
NRP – 0.3 / 0.5 / 0.7	4	7 / 8/ 9	1042.0 / 1206.0 / 1578.0
NRP – 0.3 / 0.5 / 0.7	5	5 / 7 / 10	928.0 / 1099.0 / 1616.0
NRP – 0.3 / 0.5 / 0.7	6	7 / 10 / 9	1164.0 / 1616.0 / 1427.0
NRP – 0.3 / 0.5 / 0.7	7	4 / 8 / 8	794.0 / 1326.0 / 1177.0
NRP – 0.3 / 0.5 / 0.7	8	5 / 7 / 9	1044.0 / 1195.0 / 1429.0
NRP – 0.3 / 0.5 / 0.7	9	6 / 10 / 10	917.0 / 1616.0 / 1616.0
NRP – 0.3 / 0.5 / 0.7	10	4/ 9 / 9	523.0 / 1325.0 / 1540.0

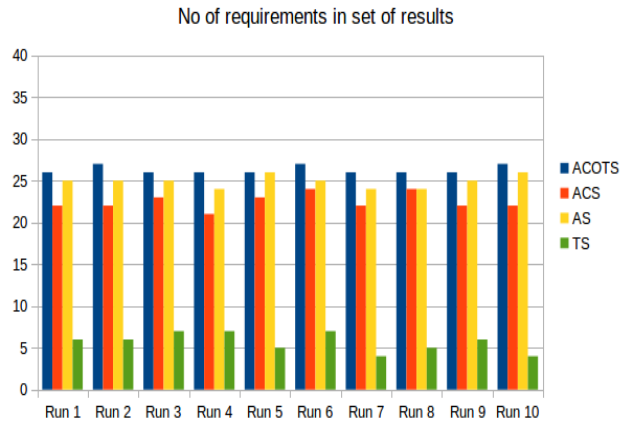

Fig. 1 Visual representation of ACOTS, ACS, AS and TS based on 30% of the budget

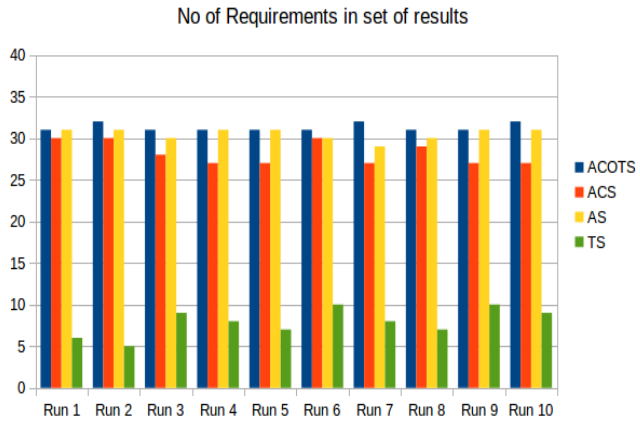

Fig. 2 Visual representation of ACOTS, ACS, AS and TS based on 50% of the budget

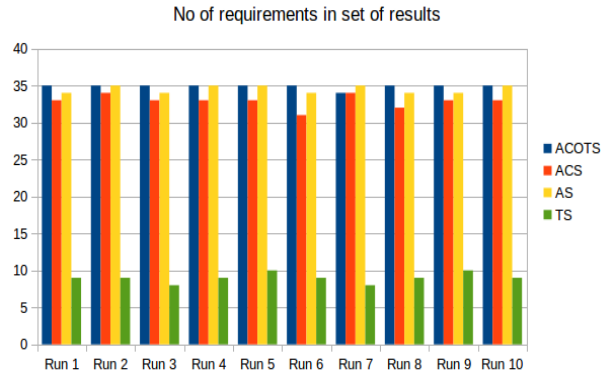

Fig. 3 Visual representation of ACOTS, ACS, AS and TS based on 70% of the budget

Comparing the results of Experiment 1 to 4 in Table 1 to 4 respectively when the objective is 0.3, 0.5 and 0.7 of the budget, requirement list is of the same size (d=40), the number of ants is of the same size (h=10) and the remaining parameters are of the same size. It can be seen that the number of requirements in the result set of Experiment 1 (Table 1) is higher than the results gotten in the other Experiments 2 to 4 at each run of the algorithm which canbe seen graphically in Figure 1 to 3 while the average cost remains constant and the execution time varies.

For hybrid ACOTS as depicted in Table 1, the number of requirements in each set of results is moderately large, hence making the range of requirements gotten predictable which can help the decision maker to assign ranges of value to the input parameters of the algorithm. It has also had the best set of requirement with varied objectives (0.3, 0.5, and 0.7) as shown in Figure 1 – 3. The AS and ACS also performed well but their respective sets of requirement

values are less than ACOTS though they all had same average cost. This is due to the usage of the same parameters from the CALIBRA. TS is inferior in comparison to other methods both in terms of cost and set of requirements.

VII.CONCLUSION

This paper proposed a hybrid Ant Colony Optimization Algorithm with Tabu Search to solve NRP. The NRP is portrayed with the decision by software engineers on what requirements to develop as new features in the next release of the software product from the set of available requirements as it would be unrealistic to develop all the features due to limited available resources and with each requirement of the set having a related value and cost associated with it. The software engineer needs to select the set of requirements to develop such that the total value of the set of requirements chosen for development is maximized alongside minimizing the total cost of development, ensuring that the available resources are used to develop the set of requirements that give the highest rating to the decision maker. The proposed hybrid ACOTS algorithm gave a better result for the NRP when compared with other methods.

REFERENCES

1. Adenso-Diaz, B., & Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. Operations research, 54(1), 99-114.
2. Almeida, J. C., Pereira, F. d. C., Reis, M. V., & Piva, B. (2018). The Next Release Problem: Complexity, Exact Algorithms, and Computations. Paper presented at the International Symposium on Combinatorial Optimization.
3. Araújo, A. A., Paixao, M., Yeltsin, I., Dantas, A., & Souza, J. (2017). An architecture based on interactive optimization and machine learning applied to the next release problem. Automated Software Engineering, 24(3), 623-671.
4. Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). The next release problem. Information and Software Technology, 43(14), 883-890.
5. Balogun, A., Mabayoje, M., Makinwa, M., & Bajeh, A. (2016). Solving the Next Release Problem using a Hybrid Metaheuristic. Annals Computer Science Series, 14(2), 101–116.
6. Braude, E. J., & Bernstein, M. E. (2016). Software engineering: modern approaches: Waveland Press.
7. Buede, D. M., & Miller, W. D. (2016). The engineering design of systems: models and methods: John Wiley & Sons.
8. Cai, X., Cheng, X., Fan, Z., Goodman, E., & Wang, L. (2017). An adaptive memetic framework for multi-objective combinatorial optimization problems: studies on software next release and traveling salesman problems. Soft Computing, 21(9), 2215-2236.
9. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation, 6(2), 182-197.
10. del Sagrado, J., del Aguila, I. M., & Orellana, F. J. (2015). Multi-objective ant colony optimization for requirements selection. Empirical Software Engineering, 20(3), 577-610.
11. Dick, J., Hull, E., & Jackson, K. (2017). Requirements engineering: Springer.
12. Doerner, K. F., & Maniezzo, V. (2018). Metaheuristic search techniques for multi-objective and stochastic problems: a history of the inventions of Walter J. Gutjahr in the past 22 years. Central European Journal of Operations Research, 26(2), 331-356.
13. Dorigo, M., & Stützle, T. (2019). Ant colony optimization: overview and recent advances Handbook of metaheuristics (pp. 311-351): Springer.
14. Drias, Y., Kechid, S., & Pasi, G. (2016). A novel framework for medical web information foraging using hybrid ACO and Tabu Search. Journal of medical systems, 40(1), 5.
15. Durillo, J. J., Zhang, Y., Alba, E., Harman, M., & Nebro, A. J. (2011). A study of the bi-objective next release problem. Empirical Software Engineering, 16(1), 29-60.
16. Engin, O., & Güçlü, A. (2018). A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems. Applied Soft Computing, 72, 166-176.
17. Finkelstein, A., & Lim, S. L. (2012). StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation. IEEE transactions on software engineering(3), 707-735.
18. Gendreau, M., & Potvin, J.-Y. (2019). Tabu Search Handbook of Metaheuristics (pp. 37-55): Springer.
19. Glover, F. (1990). Tabu search—part II. ORSA Journal on Computing, 2(1), 4-32.
20. Glover, F., Laguna, M., & Martí, R. (2018). Principles and Strategies of Tabu Search. Handbook of Approximation Algorithms and Metaheuristics: Methodologies and Traditional Applications, 1.
21. Harman, M., & Jones, B. F. (2001). Search-based software engineering. Information and Software Technology, 43(14), 833-839.
22. Jiang, H., Zhang, J., Xuan, J., Ren, Z., & Hu, Y. (2010). A hybrid ACO algorithm for the next release problem. Paper presented at the Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on.
23. Kleijnen, J. P. (2015). Design and analysis of simulation experiments. Paper presented at the International Workshop on Simulation.
24. Layton, M. C., & Ostermiller, S. J. (2017). Agile project management for dummies: John Wiley & Sons.
25. Li, L., Harman, M., Wu, F., & Zhang, Y. (2016). The value of exact analysis in requirements selection. IEEE Transactions on Software Engineering, PP (99), 1-1.
26. Li, Y., Zhang, M., Yue, T., Ali, S., & Zhang, L. (2017). Search-Based Uncertainty-Wise Requirements Prioritization. Paper presented at the Engineering of Complex Computer Systems (ICECCS), 2017 22nd International Conference on.
27. Lim, S. L. (2011). Social networks and collaborative filtering for large-scale requirements elicitation. University of New South Wales.
28. McMullen, P. R. (2017). Ant-Colony Optimization for the System Reliability Problem with Quantity Discounts. American Journal of Operations Research, 7(02), 99.
29. Poongothai, M., & Rajeswari, A. (2016). A hybrid ant colony tabu search algorithm for solving task assignment problem in heterogeneous processors. Paper presented at the Proceedings of the International Conference on Soft Computing Systems.
30. Salhi, S. (2017). Population-Based Heuristics Heuristic Search (pp. 77-128): Springer.
31. Srivastava, P. R., Varshney, A., Nama, P., & Yang, X.-S. (2012). Software test effort estimation: a model based on cuckoo search. International Journal of Bio-Inspired Computation, 4(5), 278-285.
32. Stützle, T., & López-Ibáñez, M. (2019). Automated Design of Metaheuristic Algorithms Handbook of Metaheuristics (pp. 541-579): Springer.
33. van den Akker, M., Brinkkemper, S., Diepen, G., & Versendaal, J. (2005). Determination of the Next Release of a Software Product: an Approach using Integer Linear Programming. Paper presented at the CAiSE Short Paper Proceedings.
34. Veerapen, N., Ochoa, G., Harman, M., & Burke, E. K. (2015). An integer linear programming approach to the single and bi-objective next release problem. Information and Software Technology, 65, 1-13.
35. Wu, J., Wu, G., & Wang, J. (2017). Flexible Job-shop Scheduling Problem Based on Hybrid ACO Algorithm. International Journal of Simulation Modelling, 16(3), 497-505.
36. Zhang, Y., Harman, M., Ochoa, G., Ruhe, G., & Brinkkemper, S. (2018). An Empirical Study of Meta-and Hyper-Heuristic Search for Multi-Objective Release Planning. ACM Transactions on Software Engineering and Methodology (TOSEM), 27(1), 3.