Ant Colony Optimization Resource Allocation for Software Release Planning

Jasmine Sabeena, K.Sree Divya, K.Santhi
Department of CSE, S V College of Engineering, Tirupathi,India

***Abstract*- Release planning for incremental software development allocates features to releases such that technical, resource, risk, and budget constraints are met. A feature are offered to release only if all of its necessary tasks are done before the given release date. In the context of release planning, the question studied in this paper is how to allocate these resources to the tasks of implementing the features such that the value gained from the released features is maximized. We propose at two phase optimization approach called ANT$_{RASORP}$ that combines the strength of two existing solution methods. Phase1 applies integer linear programming to a relaxed version of the full problem. Phase2 uses Ant colony optimization a reduced search space to generate operational resource allocation plans.**

***Index Terms*—Release planning, resource allocation, software project management, incremental software development, Ant Colony Optimization.**

I. Introduction

Incremental software development provides products in releases where each release provides additional or modified functionality compared to the previous release. A major problem faced by companies developing or maintaining large and complex systems is determining which elements of a typically large set of candidate features should be assigned to which releases of the software. In addition, there is the question of how to assign resources accordingly. Without good release planning, critical features are not provided at the right time. This might result in dissatisfied customers; time an budget overruns, and decreased competitiveness in the market place. Release and resource decisions depend on each other. Defining releases without looking into the necessary resources means that the proposed plans are unlikely to be feasible. One of the key limitations of current release planning methods is the lack of a systematic process to balance the appropriate delivery of features with the resources available. Greer and Ruhe [7] and van den Akker et al. [8]. On the other hand, planning resources without conssidering the business impact of the product releases can result in missed opportunities for value creation. It is well known that productivity may vary significantly among developers [1][2]. Each developer might have different productivity when performing different types of tasks. Good resource allocation in this context takes on even more importance. The resulting problem of optimal assignment of resources to realize the features of a sequence of releases is called Ant Resource Allocation for Software Release Planning, abbreviated by ANT$_{RASORP}$. Surprisingly even with the large applicability and the significant results obtained by the Ant colony optimization. Meta heuristic very little has been done of this strategy to tackle software Engineering problems module as optimization problem. (ACO) Swam intelligence framework inspired by the behavior of ants during food in search.

A. *Features and Their Assignment to Releases*

We consider a feature to be "a logical unit of behavior that is specified by a set of functional and quality requirements"[3].nature. ACO minimizes the indirect communication strategy employed by real ants mediated pheromone trails. Most important and riskier requirements are anticipated.

Definition 1: A release plan is an assignment of feature store releases. It is formally described by a matrix of decision variables x(n,k) with X(n,k) =1 if and only if feature f(n) is offered at release k (k=1..k) (and x(n,k) =0 otherwise. Assignment of features to releases cannot be done without considering different types of feature dependencies. There are different types of dependencies, as stated by Carlshamreetal.[6], which might impact the planning process. We consider the two most important ones: coupling and precedence relationship among features. Coupling of features means that a pair of features only makes combination with each other. We assume that coupling will be handled in advance by integrating the respective features into a more comprehensive whole. Precedence means that a feature is not useful in a release without having another feature already implemented. The precedence relationship is formally defined as follows:

Definition 2. Feature f(i) precedes feature f(j) means that if feature f(i) is made available at release k, then f(j) cannot be made available at any release 1..k earlier than k. We will say that features f(i) and f(j) are in a precedence relationship.

II. Planning Objectives

What actually constitutes the planning objective needs careful consideration and cannot be answered in general terms. Often, the objective is to optimize a combination of criteria such as the product's business value, time to market, and risk. Furthermore, there is a dependency of actual business value created by a feature is larger when it enters the market sooner; perhaps because a product feature might be available before those of a competitor. For our purposes , we assume a utility function value F(x) expressing the cumulative business value of all features assigned to releases according to plan x. F(x) is composed of individual values v(n, k), where v(n, k) describes the expected value contribution of feature f(n) in case it is assigned to release k. Each value v(n, k) itself is composed from the priorities assigned to features by the (weighted) stakeholders. The prioritization can be done with respect to not only one but a portfolio of (weighted) criteria. Sample prioritization criteria are the following: the estimated market value of feature f(n), urgency of feature f(n), customer satisfaction, and estimated market opportunity. For this paper, we assume that value estimates v(n, k) are given by the project manager. For further details on the composition of the coefficients v(n, k), we refer to [4].

Definition 3. The quality of a release plan x is described by the degree of optimality that x achieves with respect to utility function F(x).

III. Feature-Related Tasks

The realization of each feature requires a sequence of tasks. We assume Q different types of tasks. These tasks correspond to the fundamental technical, managerial, and support contributions necessary to develop software. Minimally, these tasks cover design, implementation and testing. The definition of what constitutes a task is flexible. There might be further differentiation within an activity. Could be refined into unit testing, integration testing, acceptance testing, and regression testing. While, in principle, the granularity of the definition of a task is flexible, we have to keep our model reasonable in size. Data related to features f(n).

A. *Resources*

Human resources (e.g., different types of developers, analysts, external collaborators, etc.) are intended to perform the tasks needed to create the features. For each individual task task (n, q) of type q necessary to provide feature f(n), a workload w(n, q) is defined as the effort needed to fulfill the task. The time needed to perform a task depends not only on the workload but also on the productivity of the developer assigned to this task nonhuman resources (e.g., capital) are considered for each release as well. We assume M different types of nonhuman resources. Feature f(n) consumes an amount r(n, m) of nonhuman resources of type m. We introduce cap (k, m) with m =1..M and k=1..k as the quantity of the (nonhuman) resource of type m that is available in release k. This results in the following capacity constraints:

$$\sum_{h=1..k} \sum_{n=1..N} r(n,m).\, x(n,h) \leq \sum_{h=1..k} cap(h,m)$$

for all m =1 and k =1..K.

A. *Assignment of Tasks to Developers*

It is well known that there are major differences in the skills and productivity of software developers [1]. In order to accommodate this, we introduce an average skill level with a normalized productivity factor of 1.0 for each type of task. This allows us to consider more or less skilled developers We consider a pool of D developers, denoted by dev(1)...dev (D), performing one or more types of development activities. The productivity factor prod(d, q) of a developer dev(d) for performing a task of type q indicates whether the developer is able to perform the task at all (prod(d,q)≠0)and, if "yes," how productively (s) he performs that task. In order to summarize information related to the assignment of developers to tasks, we will later use vector u defined as u(d,t,n,q)=1 if and only if at moment t (t=1..t|K|) developer d(d=1..D) is working on task(n,q) (and u(d,t,n,q)=0 otherwise).The assignment of human resources to feature-related tasks is illustrated in Fig. 2. For illustrative purposes, we consider an example of three features with three distinct tasks. A pool of three developers is available to perform the tasks. Each developer has different levels of productivity for performing the individual tasks. The three-dimensional productivity vector indicates the relative productivity to perform three tasks (in this order): design, implementation, and testing. For example, developer dev (2) having productivity vector (1, 0, 2) means that this developer is twice as productive as an average tester (degree of productivity with regards to testing is 2) but cannot perform an implementation task (the degree of productivity with regards to implementation is 0). In Fig. 2, we can also see a possible assignment of developers to the nine tasks under consideration. For each developer, the number at the arc between a developer and a task indicates the order in which the assigned tasks will be performed by the developers. To look at dev (2) again, the first task is the design for feature f(3), followed by testing for f(3).Finally, testing for f(2) is done. To illustrate the impact of varying productivity values, we consider the estimated workload for task (1, 3) to be W (1,3)=10 person days. Then, developer dev (3) would need 10 days to perform the tasks. As the productivity of developer dev (2) for the task of testing is assumed to be 2, the same task when performed by this developer would take only five days. The best possible assignment takes into account the productivity of the developers, their availability, and the possible dependencies between tasks.

IV. OVERALL PROBLEM STATEMENT

We describe a release plan x and the associated resource allocation u by the combined vector (x, u). We also use the terms "assignment" for x and "detailed schedule" for u. The composite set of all feasible assignments and detailed schedules1 (x, u) is denoted by the Cartesian product X, Uor just (X, U). We note that the part of the problem related to detailed schedules does not directly influence the stated objective of planning but is part of the constraint set that needs to be fulfilled. We can say that the schedule "enables" the features. The objective is to maximize the stated utility function F, which is based on the value parameters v (n, k) introduced in Section 2.1.2. The problem ANTRASORP is formally stated as Maximize

$$\{F(x)= \sum_{n=1..N} \sum_{k=1..K} v(n,k).x(n,k) \text{subject to } (x,u) \in (X,U)\}$$

V. SOLUTION APPROACH

Two-Phase Problem Solution Approach

An Overview

Given the analysis of the problem's complexity, we have concluded that we need a specialized solution approach for $\text{ANT}_{\text{RASORP}}$. Our two-phase approach, called $\text{ANT}_{\text{RASORP}}$ OPTIMIZE, combines the strength of special structure integer linear programming (Phase 1) with the power of GAs (Phase 2). The advantage is twofold. First, from Phase 1, we can generate an upper bound for the maximum value achievable. This allows an evaluation of the solution generated in Phase 2. Second, the solution obtained from Phase 1 is used to restrict the set (permutations of N features) to the set used in the GA of Phase 2. This can significantly reduce the computational effort and allows solution of problems of small and medium size. Phase 2 can be applied without application of Phase 1, but the search space would be substantially larger in this case. The restriction provided by Phase 1 is heuristic in its nature. That means that it is likely that (reduced search space) contains a good solution. However, there is no guarantee that it necessarily contains the optimal solution. We will call the direct application of GA to $\text{ANT}_{\text{RASORP}}$ unfocused search (UFS), while the two-phase method OPTIMIZE $\text{ANT}_{\text{RASORP}}$ is also called focused search (FS) (when focusing on the search strategy).

Phase 1

To facilitate the process of solving (4), we initially consider a simplified version of $\text{ANT}_{\text{RASORP}}$ Instead of looking at all t (K) possible points of time t (t =1..t(K)) for scheduling tasks, in $\text{ANT}_{\text{RASORP}}$, we only look at release dates t(k) (k=1..K). This leads to a significant reduction of variables and constraints. To formally describe this simplified problem, we use the variables y(d, k, n, q) instead of variables u(d, t, n, q) Vector u is larger in size than y. The reason for that is that u is defined for each point in time instead of just the release dates t(k) (k=1..K). More specifically, y is defined as y(d, k, n, q)=1 if and only if task(n,q) is assigned to developer d and is finished in release k (and y(d, k, n, q)=0 otherwise).

Phase 2 The main idea of Phase 2 is that the optimal or at least a near-optimal solution of the complete ANTRASORP problem can be obtained from the relaxed version ANTRASORP . That means that we take solution x1 and try to adjust it to fulfill the additional constraints. In Phase 2, we use a GA [5] to search for the best solution for assigning human resources to the tasks of the features chosen for release. Cartesian Genetic Programming was originally developed by Miller and Thomson [11][12] for the purpose of evolving digital circuits and represents a program as a directed graph. For a solution x1 obtained from Phase 1, we define sets of indices of features belonging to the same release. An ACOG is differ from that algorithm given in reference [6]. (ACO) Swam intelligence framework inspired by the behavior of ants during food search in nature. ACO minimizes the indirect communication strategy employed by real ants mediated pheromone trails.

1.Most important and riskier requirements are anticipated .

Maximize $=\sum_{i=1}^{N}[score\ (p-x_i+1)$-risk$_i$.x$_i$].y$_i$

Score = $\sum_{i=1}^{M}[wj.importance(cj.xi)]$.

2. Cost and precedence constraints

$X_0 \leq X_a$ □ $(r_a \rightarrow r_b)$.where r_a ,$r_b \in$ R. $\sum_{i=1}^{N}(COSTi.\ F_i)$,k $\leq$ budget release$_k$, □ k $\in$ {1,…..p}.

Daemon actions can be used to implement centralized actions which cannot be performed by single ants, such as the invocation of a local optimization procedure, or the update of global information to be used to decide whether to bias the search process from a non-local perspective [9][10].

Problem encoding:

The problem will be encoded as a directed graph .G= (V,E) ,where E= E_m+E_0 , with E_m representing mandatory moves and E_0 representing optional ones.

i. Each vertex in V represents a requirement.
ii. A directed mandatory edge $(T_i ,T_j) \in E_m$, if $(r_i \rightarrow r_j)$;
iii. A directed optional edge $(T_i , T_j) \in E_0$ if (T_i ,T_j) ¢ E_m , and i$\neq$j.

Overall - cost$_i$= cost$_i$ if requirement r_i has no precedent requirements.
Overall - cost$_i$= cost$_i$ +∑ overall – cost $_j$ and $(r_i \rightarrow r_j)$ for all unvisited requirements where m and_ vis$_k$(i)={r_j/(r_i, r_j) ∈ E_m and visited$_j$ = false}
opt-vis$_k$(i) ={r_j/(r_i, r_j) ∈ E_0,effort(k)+overall_cost$_j$ $\leq$ budgetrelease and visited$_j$=false} overall initialization

```
count ← 1
main loop
Repeat
    Aw Algorithm
Count++
Until count > Max.count
Return best _planning
ACO Algorithm
main loop initialization
Single Release planning loop
main loop finalization
for all vertices $r_i \in$ v, visited$_i$ ← false
for all vertices $r_i \in$ v, current_planning; ←0
//finds a new release planning {current planning}//
Ifcurrent_planningeval ()>best.planning.eval () then
best_planning ← current planning
single release planning loop
      for each release, k
Randomly place ant k in a
vertex $r_i \in$ v, where

Visited ← false and overall _cost; budjet release k
ADDS (0,k)
While opt_vis$_k$(i) $\neq$ 0 DO
Move ant k to a vertex $r_j \in$ opt_ vis$_k$ (i) with probability $P_{ij}^k$
ADD$_s$($r_j$,k)
i ← j
```

The probability distribution is specified as follows. For ant k, the probability of moving from state t to state n depends on the combination of two values [11] the attractiveness of the move, as computed by some heuristic indicating the priori desirability of that move. the trail level of the move, indicating how proficient it has been in the past to make that particular move: it represents therefore an a posteriori indication of the desirability of that move For example, the emerging structures in the case of foraging in ants include spatiotemporally organized networks of pheromone trails [14][15][16].

ACOG Algorithm

An ACOG is differ from that algorithm given in reference [6],it use genetic programming to enhance performance .It consists of two main sections : initialization and a main loop, where G_p is used in the second sections .The main loop runs for a user defined number of iterations .these are described below:
Initialization :

a. Set initial parameters that are system: variable, states, function, input, output, Input trajectory, output trajectory
b. Set initial pheromone trails value .
c. Each ant is individually placed on initial state with empty memory.
While termination conditions not meet do
a. Construct Ant solution:

Each ant constructs a path by successively applying the transition function the probability of moving from state to state depend on: as the attractiveness of the move , and the trail level of the move.
b. Apply Local search
c. Best Tour check
If there is an improvement, update it.
d. Update Trails :

Evaporate a fixed proportion of the pheromone on each road . For each ant perform the "ant -cycle" pheromone update. Reinforce the best tour with a set number of "elitist ants" performing the "ant –cycle",Create a new population by applying the following operation, based on pheromone trails. The operations are applied to the individual(s) selected from the population with a probability based on fitness.

- Darwinian Reproduction
- Structure –Preserving crossover
- Structure –Preserving Mutation

End While

VI. CONCLUSION

Since Ant colony algorithm may produce redundant states in the graph, it's better to minimize such graphs to enhance the behavior of the inducted system. A colony of ants moves through system states **X**, by applying Genetic Operations. By moving, each ant incrementally constructs a solution to the problem when an ant completes solution, or during the construction phase, the ant evaluates the solution and modifies the trail value on the components used in its solution. Ants deposit a certain amount of pheromone on the components; that is, either on the vertices or on the edges that they traverse. The amount of pheromone deposited may depend on the quality of the solution found. Subsequent ants use the pheromone information as a guide toward promising regions of the search space. Ants adaptively modify the way the problem is represented and perceived by other ants, but they are not adaptive themselves. The genetic programming paradigm permits the evolution of computer programs which can perform alternative computations conditioned on the outcome of intermediate calculations, which can perform computations on variables of many different types, which can perform iterations and recursions to achieve the desired result, which can define and subsequently use computed values and sub-programs, and whose size, shape, and complexity.

REFERENCE

[1] Acuria, N.Juristo, and A.M.Moreno, 'Emphasizing Human capabilities in Software Development, "IEEE software, vol. 23, no. 2,pp.94-101,Mar./Apr.2006.

[2]G.Meyers,"A controlled Experiments in program Testing and code Walkthroughs /Inspections, " comm., ACM, vol.21, pp.760-768, 1978.

[3]j.van Gurp,J.Bosh and M.Svahnberg," Managing Variability in Software Product Lines",Pro.Landelijk Architecture Congers ,2000

[4] G. Ruhe and A. Ngo-The, "Hybrid Intelligence in Software Release Planning," Hybrid Intelligent System, vol.1,pp.99-110,2004.

[5] GALib, http://lancet.mit.edu/ga/, 2008.

Technique, IEEE computational intelligence magazine, November, 2006

[6] Nada M.A. AL-Salami, "System Evolving using Ant Colony Optimization Algorithm ", Journal of Computer Science 5 (5): 380-387, 2009, ISSN 1549-3636

[7] D.Greer and G.Ruhe, "Software Release Planning : An iterative and Evolutionary Approach," Information and Software Tehnology,vol.46,pp.243-253,2004

[8] J.M J van den Akker, s. Brinkkemper, G.Diepen,and J. Versendaal," Software Product Release Planning through Optimization and what-if Analysis," information and Software Ethnologies, vol.50.2008,pp. 101-111,2005

[9] M. Dorigo, M. Birattari, and T. Stitzle, "Ant Colony Optimization: Arificial Ants as a Computational Intelligence Technique, IEEE.

[10] M. Dorigo and G. Di Caro, "The Ant Colony Optimization meta-heuristic", in New Ideas in

Optimization, D. Corne et al., Eds., McGraw Hill, London, UK, pp. 11-32, 1999

[11] Nada M. A. AL-salami, Saad Ghaleb Yaseen, "Ant Colony Optimization", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.6, pp 351-357, June, 2008

[12] J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, Genetic Programming, Proceedings of EuroGP'2000, volume 1802 of LNCS, pages 121–132, Edinburgh, 2000. Springer-Verlag

[13] Nada M. A. AL-salami, Saad Ghaleb Yaseen, "Ant Colony Optimization", IJCSNS International Journal of Computer

Science and Network Security, VOL.8 No.6, pp 351-357, June, 2008

[14] M. Dorigo and G. Di Caro, "The Ant Colony Optimization meta-heuristic", in New Ideas in Optimization, D. Corne et al., Eds., McGraw Hill, London, UK, pp. 11-32, 1999

[15] Simon Harding, Julian F. Miller, Wolfgang Banzhaf, "Self-Modifying Cartesian Genetic Programming", GECCO'07, July 7–11, 2007, ACM 978-1-59593-697-4/07/0007, pp: 1021-1028

[16] J. Holland, "Adaptation in Natural and Artificial Systems", Ann Arbor: University of Michigan Press, 1975.