3rd International Symposium on Search Based Software Engineering
September 10 – 12,
Szeged, Hungary

An Ant Colony Optimization Approach to the Software Release Planning Problem with Dependent Requirements

Jerffeson Teixeira de Souza, Camila Loiola Brito Maia, Thiago do Nascimento Ferreira, Rafael Augusto Ferreira do Carmo and Márcia Maria Albuquerque Brasil

Optimization in Software Engineering Group (GOES.UECE)

State University of Ceará, Brazil

Motivation

The Search Based Software Engineering (SBSE) field has been benefited from a number of general search methods.

Surprisingly, even with the large applicability and the significant results obtained by the Ant Colony Optimization (ACO) metaheuristic, very little has been done regarding the employment of this strategy to tackle software engineering problems modeled as optimization problems.

Ant Colony Optimization

"swarm intelligence framework, inspired by the behavior of ants during food search in nature."

"ACO mimics the indirect communication strategy employed by real ants mediated by pheromone trails, allowing individual ants to adapt their behavior to reflect the colony´s search experience."

“

The software release planning problem addresses the selection and assignment of requirements to a sequence of releases, such that the most important and riskier requirements are anticipated, and both cost and precedence constraints are met.

”

*“The **software release planning problem** addresses the selection and assignment of requirements to a sequence of releases, such that the **most important** and **riskier** requirements are anticipated, and both **cost** and **precedence constraints** are met.”*

$$Maximize \sum_{i=1}^{N} (score_i . (P - x_i + 1) - risk_i . x_i) . y_i$$

$$score_i = \sum_{j=1}^{M} w_j . importance(c_j, r_i)$$

*"The **software release planning problem** addresses the selection and assignment of requirements to a sequence of releases, such that the **most important and riskier requirements are anticipated**, and both cost and precedence constraints are met."*

“ *The **software release planning problem** addresses the selection and assignment of requirements to a sequence of releases, such that the **most important** and **riskier** requirements are anticipated, and both **cost and precedence constraints** are met.* ”

$$x_b \leq x_a , \forall (r_a \rightarrow r_b) , where\ r_a, r_b \in R$$

$$\sum_{i=1}^{N} cost_i . f_{i,k} \leq budgetRelease_k,\ for\ all\ k \in \{1, \dots, P\}$$

How can the ACO framework be adapted to solve the Software Release Planning problem in the presence of dependent requirements?

ACO for the Software Release Planning problem

How can the ACO frameworkbe adapted to solve the Software Release Planning problem in the presence of dependent requirements?

ACO for the Software Release Planning problem

How does the proposed ACO adaptation compare to other metaheuristics in solving the Software Release Planning problem in the presence of dependent requirements?

ACO versus Other Metaheuristics

How can the ACO algorithm be adapted to solve the Software Release Planning problem in the presence of dependent requirements?

ACO for the Software Release Planning problem

The ACO Algorithm

PROBLEM ENCONDING

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

i. each vertex in V represents a requirement r_i ;

ii. a directed mandatory edge $(r_i, r_j) \in E_m$, if $(r_i \rightarrow r_j)$;

iii. a directed optional edge $(r_i, r_j) \in E_o$, if $(r_i, r_j) \notin E_m$ and $i \neq j$.

MORE

PROBLEM ENCONDING

$overall\_cost_i = cost_i$ if requirement r_i has no precedent requirements

$overall\_cost_i = cost_i + \sum overall\_cost_j$

and for all r_j $(r_i \rightarrow r_j)$

unvisited requirements where

$$mand\_vis_k(i) = \{r_j | (r_i, r_j) \in E_m \text{ and } visited_j = False\}$$

$$opt\_vis_k(i) = \{r_j | (r_i, r_j) \in E_o, effor(k) + overall\_cost_j \leq budgetRelease_k \text{ and } visited_j = False\}$$

```
OVERALL INITIALIZATION
COUNT ← 1

MAIN LOOP
REPEAT
```

The Proposed ACO Algorithm for the Software Release Planning Problem

```
COUNT ++
UNTIL  COUNT > MAX_COUNT

RETURN best_planning
```

```
OVERALL INITIALIZATION
COUNT ← 1

MAIN LOOP
REPEAT

        MAIN LOOP INITIALIZATION

        SINGLE RELEASE PLANNING LOOP

        MAIN LOOP FINALIZATION

COUNT ++
UNTIL COUNT > MAX_COUNT

RETURN best_planning
```

```
OVERALL INITIALIZATION
$COUNT \leftarrow 1$

MAIN LOOP
REPEAT

        MAIN LOOP INITIALIZATION
                FOR ALL vertices $r_i \in V$, $visited_i \leftarrow False$
                FOR ALL vertices $r_i \in V$, $current\_planning_i \leftarrow 0$
        SINGLE RELEASE PLANNING LOOP


        MAIN LOOP FINALIZATION



$COUNT$ ++
UNTIL $COUNT > MAX\_COUNT$

RETURN $best\_planning$
```

```
OVERALL INITIALIZATION
COUNT ← 1

MAIN LOOP
REPEAT

        MAIN LOOP INITIALIZATION
                FOR ALL vertices $r_i \in V$, $visited_i \leftarrow False$
                FOR ALL vertices $r_i \in V$, $current\_planning_i \leftarrow 0$
        SINGLE RELEASE PLANNING LOOP
                // FINDS A NEW RELEASE PLANNING (current_planning)

        MAIN LOOP FINALIZATION



COUNT ++
UNTIL  COUNT  > MAX_COUNT

RETURN best_planning
```

```
OVERALL INITIALIZATION
COUNT ← 1

MAIN LOOP
REPEAT

          MAIN LOOP INITIALIZATION
                    FOR ALL vertices $r_i \in V$, $visited_i \leftarrow False$
                    FOR ALL vertices $r_i \in V$, $current\_planning_i \leftarrow 0$
          SINGLE RELEASE PLANNING LOOP
                    // FINDS A NEW RELEASE PLANNING (current_planning)

          MAIN LOOP FINALIZATION
                    IF current_planning.eval( ) > best_planning.eval( ) THEN
                              best_planning ← current_planning

COUNT ++
UNTIL COUNT > MAX_COUNT

RETURN best_planning
```

SINGLE RELEASE PLANNING LOOP

```
FOR EACH Release, $k$
        Randomly place ant $k$ in a vertex $r_i \in V$, where
        $visited_i \leftarrow False$ and $overall\_cost_i \leq budgetRelease_k$
ADDS ($r_i$, $k$)
WHILE $opt\_vis_k(i) \neq 0$ DO
        Move ant $k$ to a vertex $r_j \in opt\_vis_k(i)$ with
        probability $p_{ij}^{k}$
        ADDS ($r_j$, $k$)
        $i \leftarrow j$
```

// Besides r_i, adds to release k all of its dependent requirements, and, repeatedly, their dependent requirements

```
ADDS ($r_i$, $k$)
    ENQUEUE ($Q$, $r_i$)
    WHILE $Q \neq \emptyset$ DO
        $r_s \leftarrow$ DEQUEUE ($Q$)
        FOR EACH $r_t \leftarrow \in mand\_vis_k(s)$ DO
            ENQUEUE ($Q$, $r_t$)
        $visited_s \leftarrow True$
        $current\_planning_s \leftarrow k$
```

SINGLE RELEASE PLANNING LOOP

```
FOR EACH Release, $k$
        Randomly place ant $k$ in a vertex $r_i \in V$, where
         $visited_i \leftarrow False$ and $overall\_cost_i \leq budgetRelease_k$
    ADDS ($r_i$, $k$)
    WHILE $opt\_vis_k(i) \neq 0$ DO
        Move ant $k$ to a vertex $r_j \in opt\_vis_k(i)$ with
         probability $p_{ij}^{\ k}$
        ADDS ($r_j$, $k$)
        $i \leftarrow j$
```

Experimental Evaluation

Results and Analyses

How does the proposed ACO adaptation compare to other metaheuristics in solving the Software Release Planning problem in the presence of dependent requirements?

?

ACO versus Other Metaheuristics

The Experimental Data

Table below presents the number of releases, requirements and clients of a sample of the 72 synthetically generated instances used in the experiments.

Instance Name	Instance Features				
	Number of Requirements	Number of Releases	Number of Clients	Precedence Density	Overall Budget
I_50.5.5.80	50	5	5	0%	80%
I_50.5.5.120	50	5	5	0%	120%
…	…	…	…	…	…
I_200.50.20.20.80	200	50	20	20%	80%
…	…	…	…	…	…
I_500.50.20.20.120	500	50	20	20%	120%

The Algorithms

Genetic Algorithm (GA)

widely applied evolutionary algorithm, inspired by Darwin´s theory of natural selection, which simulates biological processes such as Inheritance, mutation, crossover, and selection

Simulated Annealing (SA)

it is a procedure for solving arbitrary optimization problems based on an analogy with the annealing process in solids.

Comparison Metrics

Quality

it relates to the quality of each generated solution, measured by the value of the objective function.

Execution Time

it measures the required execution time of each strategy.

PAPER SUPPORTING MATERIAL WEBPAGE

http://www.larces.uece.br/~goes/rp/aco/

Results

ACO performed better than GA and SA in all cases.

Percentagewise, ACO generated solutions, in average, 78.27% better than those produced by GA and 96.77% than SA.

In terms of execution time, ACO operated substantially slower than the other two metaheuristics. In average, ACO required almost 60 times more than GA and more than 90 times more than SA.

ACO (1k) x GA (1k) x SA (1k) (General Results)

Instances in which statistical confidence cannot be assured when comparing the quality of the solutions generated by ACO with GA and SA, with all algorithms executing 1000 evaluations, calculated with the Wilcoxon Ranked Sum Test

	90% confidence level	95% confidence level	99% confidence level
GA	I_50.5.5.20.80, I_50.5.20.0.80, I_50.20.20.0.120, I_50.20.20.20.80, I_200.5.20.0.80,I_200.5.20.0.120, I_500.5.5.0.80,I_500.5.20.0.80	I_50.5.5.20.80, I_50.5.20.0.80 I_50.20.20.0.120,I_50.20.20.20.80, I_200.5.20.0.80,I_200.5.20.0.120, I_500.5.5.0.80,I_500.5.20.0.80	I_50.5.5.20.80, I_50.5.20.0.80, I_50.20.20.0.120,I_50.20.20.20.80, I_200.5.20.0.80,I_200.5.20.0.120, I_200.50.20.0.80,I_500.5.5.0.80, I_500.5.20.0.80,I_500.5.20.0.120, I_500.20.20.0.80
SA	-	-	-

Only for 8 instances - out of 72 -, statistical significance could not be assured under the 95% confidence level when comparing ACO with GA.

For SA, even within the 99% level, ACO performed significantly better in all cases.

ACO (1k) x GA (1k) x SA (1k) (Statistical Analyses)

Results

The ACO algorithm did better than GA in 69 out of the 72 instances.

The exact same behavior occurred with SA, which outperformed ACO over the same 3 instances.

ACO was still substantially slower than GA and SA. This time, however, ACO performed around 9 times slower than both GA and SA.

ACO (1k) x GA (10k) x SA (10k) (General Results)

Instances in which statistical confidence cannot be assured when comparing the quality of the solutions generated by ACO with GA and SA, with ACO executing 1000 evaluations, and GA and SA executing 10000 evaluations, when ACO performed better, calculated with the Wilcoxon Ranked Sum Test

	90% confidence level	95% confidence level	99% confidence level
GA	I_50.20.5.0.120,I_50.20.20.0.80, I_50.20.20.20.80,I_50.50.20.0.120, I_50.50.20.20.120,I_200.5.20.0.120, I_200.5.20.20.80,I_200.50.20.0.80, I_500.5.5.0.120, I_500.5.20.0.80	I_50.5.5.20.120,I_50.20.5.0.120, I_50.20.20.0.80,I_50.20.20.20.80, I_50.50.20.0.120,I_50.50.20.20.120, I_200.5.20.0.120,I_200.5.20.20.80, I_200.50.20.0.80,I_500.5.5.0.120, I_500.5.20.0.80	I_50.5.5.20.80,I_50.5.5.20.120, I_50.5.20.20.80,I_50.20.5.0.120, I_50.20.5.20.120,I_50.20.20.0.80, I_50.20.20.20.80,I_50.50.5.20.120, I_50.50.20.0.120,I_50.50.20.20.120, I_200.5.20.0.120,I_200.5.20.20.80, I_200.50.20.0.80,I_500.5.5.0.80, I_500.5.5.0.120,I_500.5.20.0.80
SA	I_50.5.20.20.120, I_500.5.20.0.80	I_50.5.20.20.120, I_500.5.20.0.80	I_50.5.20.20.80,I_50.5.20.20.120

Considering a confidence level of 95%, GA and SA had, respectively, two and three cases where they were able to produce significantly better solutions.

ACO (1k) x GA (10k) x SA (10k) (Statistical Analyses)

Results

Even with the time restriction, ACO continues to outperform both GA and SA, respectively, in 70 and 72 out of 72 cases.

ACO (Restricted Time) x GA (1k) x SA (1k) (General Results)

Correlation metrics (Pearson, Kendall and Spearman) over the results generated by ACO before and after the time restriction.

	Pearson Correlation	Kendall Correlation	Spearman Correlation
ACO (1k) vs ACO - Time GA (1k)	0.9999907	0.9929577	0.9993890
ACO (1k) vs ACO - Time SA (1k)	0.9999879	0.9866980	0.9987137

ACO lost very little of its capacity when subjected to such time constraints.

When performing better, GA could not obtain solutions significantly better than ACO (95% confidence level).

Over all other cases, under the 95% level, ACO did significantly better 63 times.

Considering SA, ACO significantly outperformed this algorithm all but one case.

ACO (Restricted Time) x GA (1k) x SA (1k) (Statistical Analyses)

Threats to Validity

Artificial instances

Parameterization of algorithms

Very little has been done regarding the employment of the Ant Colony Optimization (ACO) framework to tackle software engineering problems modeled as optimization problems.

This paper describes a novel ACO-based approach for the Software Release Planning problem with the presence of dependent requirement.

All experimental results pointed out to the ability of the proposed ACO approach to generate precise solutions with very little computational effort.

CONCLUSIONS

ANNOUNCEMENT

II Brazilian Workshop on Search Based Software Engineering

along with
XXV Brazilian Symposium on Software Engineering (SBES 2011)
XV Brazilian Symposium on Programming Languages (SBLP 2011)
XIV Brazilian Symposium on Formal Methods (SBMF 2011)
V Brazilian Symposium on Software Components,
Architectures and Reuse (SBCARS 2011)

SÃO PAULO - SP, BRAZIL
SEPTEMBER 26, 2011
http://www.compose.ufpb.br/wesb2011/

That is it!

Thanks for your time and attention.