SBSE for Early Lifecyle Software Engineering

23rd February 2011

London, UK

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

Jerffeson Teixeira de Souza, Ph.D

Optimization in Software Engineering Group (GOES.UECE)

State University of Ceará, Brazil

Nice to meet you,

Jerffeson Teixeira de Souza, Ph.D.

State University of Ceará, Brazil

Professor

http://goes.comp.uece.br/

prof.jerff@gmail.com

Our little time will be divided as follows

- Motivation
- Problem Definition
- Research Questions
- Problem Encoding
- The ACO Algorithm
- Experimental Evaluation
- Conclusion

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 <u>cost and precedence constraints</u> are met.*** ”

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

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

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 and $overall\_cost_i = cost_i + \sum overall\_cost_j$ for all unvisited requirements r_j where $(r_i \rightarrow r_j)$

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

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

<u>OVERALL INITIALIZATION</u>

COUNT ← 1

<u>MAIN LOOP</u>

REPEAT

The Proposed ACO Algorithm for the Software Release Planning Problem

COUNT ++

UNTIL *COUNT* > *MAX_COUNT*

RETURN *best_planning*

<u>OVERALL INITIALIZATION</u>

COUNT ← 1

<u>MAIN LOOP</u>

REPEAT

COUNT ++

UNTIL *COUNT* > *MAX_COUNT*

RETURN *best_planning*

<u>OVERALL INITIALIZATION</u>

COUNT ← 1

<u>MAIN LOOP</u>

REPEAT

<u>MAIN LOOP</u> <u>INITIALIZATION</u>

<u>SINGLE RELEASE PLANNING LOOP</u>

<u>MAIN</u> <u>LOOP FINALIZATION</u>

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

MAIN LOOP FINALIZATION

COUNT ++

UNTIL *COUNT* > *MAX_COUNT*

RETURN *best_planning*

<u>OVERALL INITIALIZATION</u>

COUNT $\leftarrow$ 1

<u>MAIN LOOP</u>

REPEAT

<u>MAIN LOOP</u> <u>INITIALIZATION</u>

FOR ALL vertices $r_i \in V$, $visited_i \leftarrow False$

FOR ALL vertices $r_i \in V$, $current\_planning_i \leftarrow 0$

<u>SINGLE RELEASE PLANNING LOOP</u>

<u>MAIN</u> <u>LOOP FINALIZATION</u>

COUNT ++

UNTIL *COUNT* > *MAX_COUNT*

RETURN *best_planning*

<u>OVERALL INITIALIZATION</u>

$COUNT \leftarrow 1$

<u>MAIN LOOP</u>

REPEAT

<u>MAIN LOOP</u> **<u>INITIALIZATION</u>**

FOR ALL vertices $r_i \in V$, $visited_i \leftarrow False$

FOR ALL vertices $r_i \in V$, $current\_planning_i \leftarrow 0$

<u>SINGLE RELEASE PLANNING LOOP</u>

// FINDS A NEW RELEASE PLANNING (*current_planning*)

<u>MAIN</u> **<u>LOOP FINALIZATION</u>**

$COUNT$ ++

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

// FINDS A NEW RELEASE PLANNING (*current_planning*)

MAIN LOOP FINALIZATION

IF *current_planning.eval*( ) > *best_planning.eval*( ) **THEN**

best_planning $\leftarrow$ *current_planning*

COUNT ++

UNTIL *COUNT* > *MAX_COUNT*

RETURN *best_planning*

<u>OVERALL INITIALIZATION</u>

COUNT ← 1

<u>MAIN LOOP</u>

REPEAT

<u>MAIN LOOP</u> <u>INITIALIZATION</u>

FOR ALL vertices $r_i \in V$, $visited_i \leftarrow False$

FOR ALL vertices $r_i \in V$, $current\_planning_i \leftarrow 0$

<u>SINGLE RELEASE PLANNING LOOP</u>

// FINDS A NEW RELEASE PLANNING (*current_planning*)

<u>MAIN</u> <u>LOOP FINALIZATION</u>

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)

$i \leftarrow j$

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)

$i \leftarrow j$

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 ≠ ∅ DO
    r_d ← DEQUEUE (Q)
    FOR EACH r_s ← ∈ opt_vis_k(i) DO
        ENQUEUE (Q, r_s)
    visited_d ← True
    current_planning_d ← k
```

// 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 $Q \neq \varnothing$ **DO**

$r_d \leftarrow$ DEQUEUE (Q)

FOR EACH $r_s \leftarrow \in opt\_vis_k(i)$ **DO**

ENQUEUE (Q, r_s)

$visited_d \leftarrow True$

$current\_planning_d \leftarrow k$

// 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 $Q \neq \varnothing$ **DO**

$r_d \leftarrow$ DEQUEUE (Q)

FOR EACH $r_s \leftarrow \in opt\_vis_k(i)$ **DO**

ENQUEUE (Q, r_s)

$visited_d \leftarrow True$

$current\_planning_d \leftarrow k$

// 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 $Q \neq \varnothing$ **DO**

$r_d \leftarrow$ DEQUEUE (Q)

FOR EACH $r_s \leftarrow \in opt\_vis_k(i)$ **DO**

ENQUEUE (Q, r_s)

$visited_d \leftarrow True$

$current\_planning_d \leftarrow k$

// 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**

$r_d \leftarrow$ DEQUEUE (Q)

FOR EACH $r_s \leftarrow \in opt\_vis_k(i)$ **DO**

ENQUEUE (Q, r_s)

$visited_d \leftarrow True$

$current\_planning_d \leftarrow k$

// 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 ≠ ∅ DO
    r_d ← DEQUEUE (Q)
    FOR EACH r_s ← ∈ opt_vis_k(i) DO
        ENQUEUE (Q, r_s)
    visited_d ← True
    current_planning_d ← k
```

// 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**

$r_d \leftarrow$ DEQUEUE (Q)

FOR EACH $r_s \leftarrow \in opt\_vis_k(i)$ **DO**

ENQUEUE (Q, r_s)

$visited_d \leftarrow True$

$current\_planning_d \leftarrow k$

// 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**

$r_d \leftarrow$ DEQUEUE (Q)

FOR EACH $r_s \leftarrow \in opt\_vis_k(i)$ **DO**

ENQUEUE (Q, r_s)

$visited_d \leftarrow True$

$current\_planning_d \leftarrow k$

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)

$i \leftarrow j$

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)

$i \leftarrow j$

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)

$i \leftarrow j$

<u>SINGLE RELEASE PLANNING LOOP</u>

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)

$i \leftarrow j$

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 the three synthetically generated instances used in the experiments.

Instance Name	Instance Features		
	# Releases	**# Requirements**	**# Clients**
INST.A	5	30	3
INST.B	10	50	5
INST.C	20	80	8

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.

RESULTS

Instance	GA	SA	ACO
INST.A	**8,508.50** ± 337.08	**8,143.95** ± 679.84	**10,753.75** ± 174.15
INST.B	**29,815.60** ± 822.03	**27,683.75** ± 1,360.96	**37,031.40** ± 318.88
INST.C	**211,196.15** ± 3,562.85	**198,431.30** ± 8,549.32	**255,149.05** ± 2,547.04

Quality of Results for Instaces A, B anc C

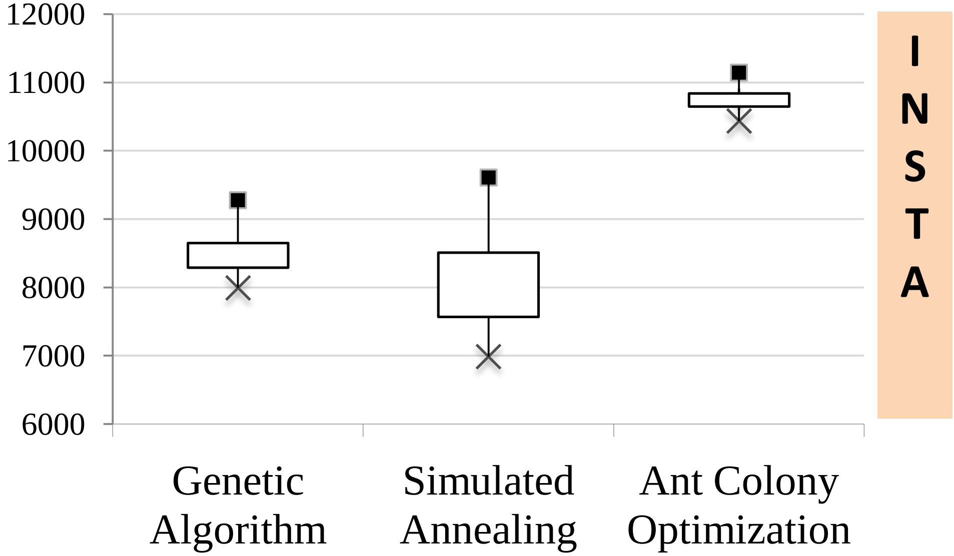
averages and standard deviations, over 100 executions

RESULTS

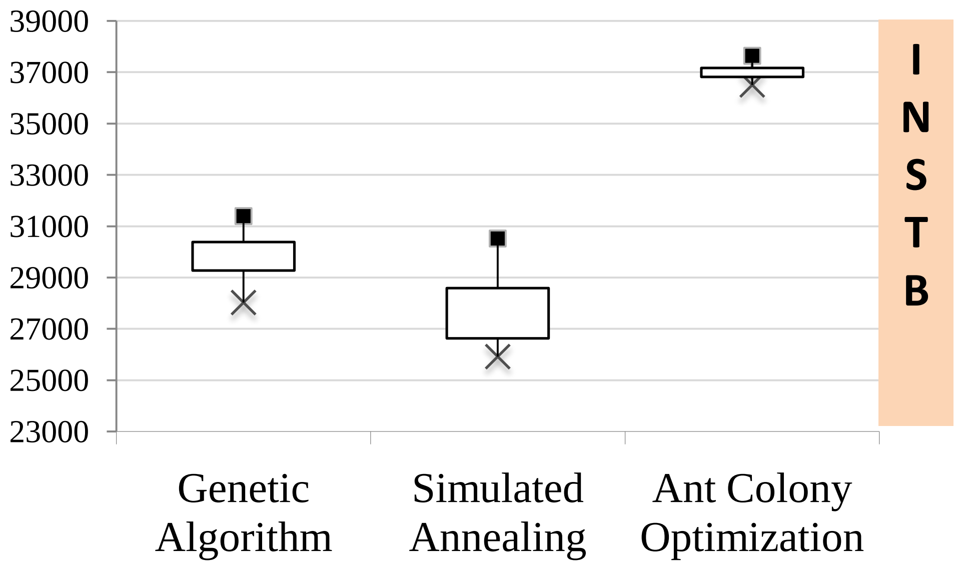
Instance	GA	SA	ACO
INST.A	**693.75** ± 26.69	**150.75** ± 7.79	**128.25** ± 17.85
INST.B	**2,597.10** ± 69.22	**329.60** ± 33.64	**284.25** ± 21.99
INST.C	**125,721.85** ± 13.037.98	**2,879.80** ± 1.038.67	**1,294.05** ± 35.39

Execution time (in milliseconds) for Instaces A, B anc C

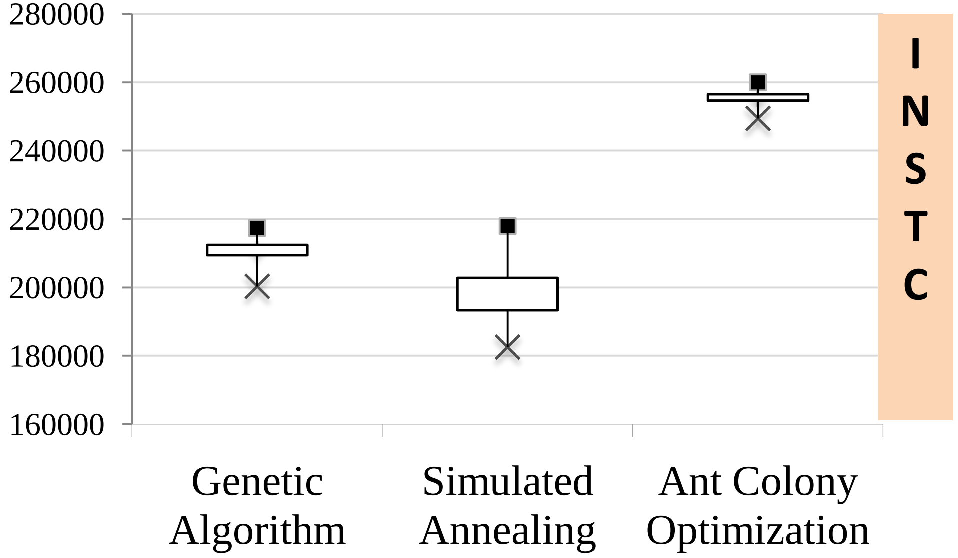
averages and standard deviations, over 100 executions



Boxplots showing maximum (■), minimum (×) and 25% - 75% quartile ranges of quality for all instances, for GA , SA and ACO.



Boxplots showing maximum (■), minimum (×) and 25% - 75% quartile ranges of quality for all instances, for GA , SA and ACO.



Boxplots showing maximum (■), minimum (×) and 25% - 75% quartile ranges of quality for all instances, for GA , SA and ACO.

Threats to Validity

- *Small number, size and diversity of instances*
- *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 talk described 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

INVITATION

II Brazilian Workshop on Optimization in 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 26-30, 2011
http://www.each.usp.br/cbsoft2011

That is it!

Thanks for your time and attention.