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:

1. Motivation
2. Problem Definition
3. Research Questions
4. Problem Encoding
5. The ACO Algorithm
6. Experimental Evaluation
7. 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.
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.

\[ x_b \leq x_a, \forall (r_a \rightarrow r_b), \text{where } r_a, r_b \in R \]

\[ \sum_{i=1}^{N} \text{cost}_i \cdot f_{i,k} \leq \text{budgetRelease}_k, \text{ for all } k \in \{1, ..., P\} \]
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 framework be 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 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 \( r_i \to r_j \);

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 encoding

\[ \text{overall}_i = \text{cost}_i \quad \text{if requirement } r_i \quad \text{has no precedent} \]

\[ \text{overall}_i = \text{cost}_i + \sum \text{overall}_j \quad \text{for all unvisited requirements } r_j \quad \text{where } (r_i \rightarrow r_j) \]

\[ \text{mand}_k(i) = \{ r_j | (r_i, r_j) \in E_m \text{ and } \text{visited}_j = \text{False} \} \]

\[ \text{opt}_k(i) = \{ r_j | (r_i, r_j) \in E_o, \text{effor}(k) + \text{overall}_j \leq \text{budgetRelease}_k \text{ and } \text{visited}_j = \text{False} \} \]
OVERALL INITIALIZATION
\[ \text{COUNT} \leftarrow 1 \]

MAIN LOOP
REPEAT

\[ \text{THE PROPOSED ACO ALGORITHM FOR THE SOFTWARE RELEASE PLANNING PROBLEM} \]

\[ \text{COUNT} ++ \]
\[ \text{UNTIL} \, \text{COUNT} > \text{MAX\_COUNT} \]

\[ \text{RETURN} \, \text{best\_planning} \]
OVERALL_INITIALIZATION

COUNT ← 1

MAIN_LOOP

REPEAT

COUN T ++

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 $\leftarrow COUNT + 1$

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
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

\[ COUNT ++ \]
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 \( current_planning.evaluation() > best_planning.evaluation() \) THEN
    \( best_planning \leftarrow current_planning \)

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 $best\_planning$
**SINGLE RELEASE PLANNING LOOP**

**FOR EACH** Release, $k$

Randomly place ant $k$ in a vertex $r_i \in V$, where $\text{visited}_i \leftarrow False$ and $\text{overall \_ cost}_i \leq \text{budgetRelease}_k$

ADDS $(r_i, k)$

**WHILE** $\text{opt \_ vis}_k(i) \neq 0$ **DO**

Move ant $k$ to a vertex $r_j \in \text{opt \_ vis}_k(i)$ 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 $overall\_cost_i \leq budget\_Release_k$

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 $overall\_cost_i \leq budget\_Release_k$

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

**ADD$$S (r_i, k)$$**

```
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

$\text{ADDS} (r_i, k)$

$\text{ENQUEUE} (Q, r_i)$

$\text{WHILE } Q \neq \emptyset \text{ DO}$

\[
\begin{align*}
  r_d & \leftarrow \text{DEQUEUE} (Q) \\
  \text{FOR EACH } r_s & \leftarrow \in opt_{vis_k}(i) \text{ DO} \\
  & \text{ENQUEUE} (Q, r_s)
\end{align*}
\]

\[
\begin{align*}
  visited_d & \leftarrow \text{True} \\
  current_{\text{planning}}_d & \leftarrow k
\end{align*}
\]
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

\[
\text{ADDS} \ (r_i, k) \\
\text{ENQUEUE} \ (Q, r_i) \\
\text{WHILE} \ Q \neq \emptyset \text{ DO} \\
\quad r_d \leftarrow \text{DEQUEUE} \ (Q) \\
\quad \text{FOR EACH} \ r_s \leftarrow \in \text{opt}_vis_k(i) \text{ DO} \\
\quad \quad \text{ENQUEUE} \ (Q, r_s) \\
\quad \text{visited}_d \leftarrow \text{True} \\
\quad \text{current_planning}_d \leftarrow k
\]
// 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 $r_s \in \text{opt\_vis}_k(i)$ 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

\[
\text{ADD}\ (r_i, k)
\]
\[
\begin{align*}
\text{ENQUEUE}\ (Q, r_i) \\
\text{WHILE } Q \neq \emptyset \text{ DO} \\
\quad r_d \leftarrow \text{DEQUEUE}\ (Q) \\
\quad \text{FOR EACH } r_s \leftarrow \in \text{opt}_{\text{vis}}_k(i) \text{ DO} \\
\quad \quad \text{ENQUEUE}\ (Q, r_s) \\
\quad \text{visited}_d \leftarrow \text{True} \\
\quad \text{current}_{\text{planning}}_d \leftarrow k
\end{align*}
\]
// Besides \( r_i \), adds to release \( k \) all of its dependent requirements, and, repeatedly, their dependent requirements

ADD \( (r_i, k) \)

ENQUEUE \( (Q, r_i) \)

WHILE \( Q \neq \emptyset \) DO

\( r_d \leftarrow \text{DEQUEUE} \( Q \) \)

FOR EACH \( r_s \leftarrow \in \text{opt_vis}_k(i) \) 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 $\text{visited}_i \leftarrow False$ and $\text{overall\_cost}_i \leq \text{budgetRelease}_k$

ADDS $(r_i, k)$

WHILE $\text{opt\_vis}_k(i) \neq 0$ DO

Move ant $k$ to a vertex $r_j \in \text{opt\_vis}_k(i)$ 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 \(overall\_cost_i \leq budget\_Release_k\)

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 $\text{visited}_i \leftarrow False$ and $\text{overall\_cost}_i \leq \text{budget\_Release}_k$

ADDS $(r_i, k)$

WHILE $\text{opt\_vis}_k(i) \neq 0$ DO

Move ant $k$ to a vertex $r_j \in \text{opt\_vis}_k(i)$ 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 $(r_j, k)$

$i \leftarrow j$
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.

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Instance Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Releases</td>
</tr>
<tr>
<td>INST.A</td>
<td>5</td>
</tr>
<tr>
<td>INST.B</td>
<td>10</td>
</tr>
<tr>
<td>INST.C</td>
<td>20</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>Instance</th>
<th>GA</th>
<th>SA</th>
<th>ACO</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST.A</td>
<td>8,508.50 ± 337.08</td>
<td>8,143.95 ± 679.84</td>
<td>10,753.75 ± 174.15</td>
</tr>
<tr>
<td>INST.B</td>
<td>29,815.60 ± 822.03</td>
<td>27,683.75 ± 1,360.96</td>
<td>37,031.40 ± 318.88</td>
</tr>
<tr>
<td>INST.C</td>
<td>211,196.15 ± 3,562.85</td>
<td>198,431.30 ± 8,549.32</td>
<td>255,149.05 ± 2,547.04</td>
</tr>
</tbody>
</table>

**Quality of Results for Instances A, B and C**

averages and standard deviations, over **100 executions**
## RESULTS

<table>
<thead>
<tr>
<th>Instance</th>
<th>GA</th>
<th>SA</th>
<th>ACO</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST.A</td>
<td>693.75 ± 26.69</td>
<td>150.75 ± 7.79</td>
<td>128.25 ± 17.85</td>
</tr>
<tr>
<td>INST.B</td>
<td>2,597.10 ± 69.22</td>
<td>329.60 ± 33.64</td>
<td>284.25 ± 21.99</td>
</tr>
<tr>
<td>INST.C</td>
<td>125,721.85 ± 13.037.98</td>
<td>2,879.80 ± 1.038.67</td>
<td>1,294.05 ± 35.39</td>
</tr>
</tbody>
</table>
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 requirements.

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