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

An Ant Colony based Algorithm for Test Case Prioritization with Precedence

Camila Loiola Brito Maia, Thiago do Nascimento Ferreira, Fabrício Gomes de Freitas and Jerffeson Teixeira de Souza

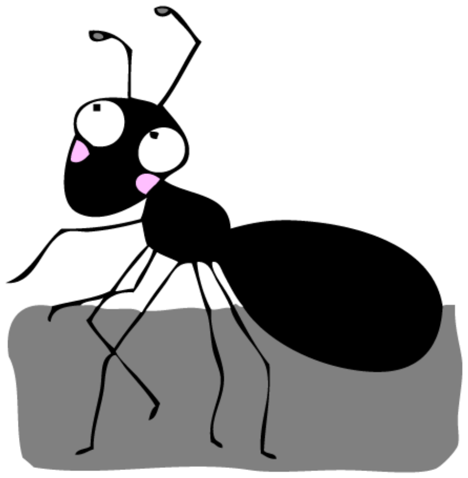
Optimization in Software Engineering Group (GOES.UECE)

State University of Ceará, Brazil

The following topics will be discussed

- Introduction
- Problem definition
- Problem encoding
- The proposed ACO-based algorithm
- Conclusions and future works

INTRODUCTION



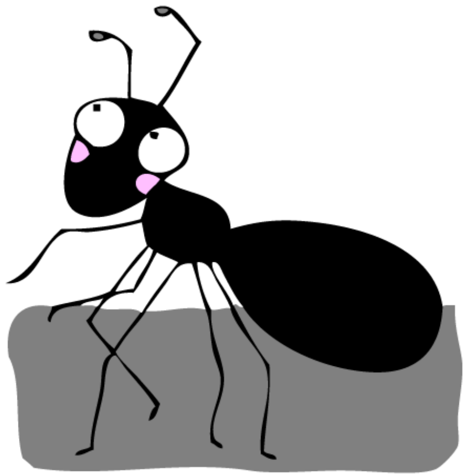
Test Case Prioritization

- The more important ones has to be tested before
  - What is more important for the client?
  - Are there any constraints for the prioritization?

ACO applied to TCP problem

- Few works on TCP
- Precedence of test cases not considered yet
- Application in other problems in our group
- A new approach considering the precedence of test cases

PROBLEM DEFINITION



Requirements

- Set ***R*** of requirements
  - ***N*** elements
- Attributes
  - $importance_i$
  - $volatility_i$

Test Cases

- Set ***C*** of test cases
  - ***M*** elements
- Attributes
  - *precedence*$_j$
  - *coverage*$_j$
  - *executionTime*$_j$
  - $$score_j = \sum \left((importance_i * weight1) + (volatility_i * weight2) \right), \quad \forall\, requirement_i \in coverage_j$$

Mathematical Formulation

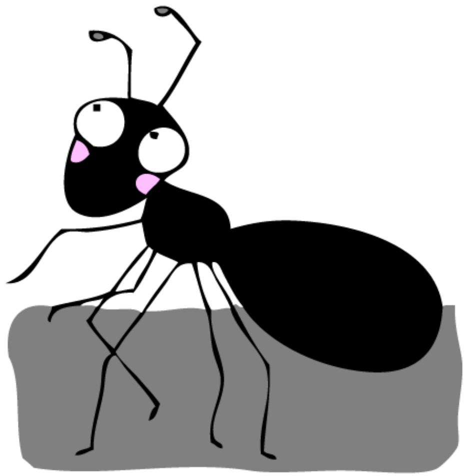
$$Maximize \sum_{j=1}^{M} \left(\frac{score_j}{executionTime_j} * P_j\right)$$

Subject to:

$$\forall\, t_{j1}, t_{j2} \in C, \qquad \left(precedence_{t_{j2}} = t_{j1}\right) \rightarrow \left(q_{t_{j1}} < q_{t_{j2}}\right)$$

- Where $P_j = M - q_j + 1$
- q_j is the position of test case j in the ordered suite

PROBLEM ENCODING



Overview

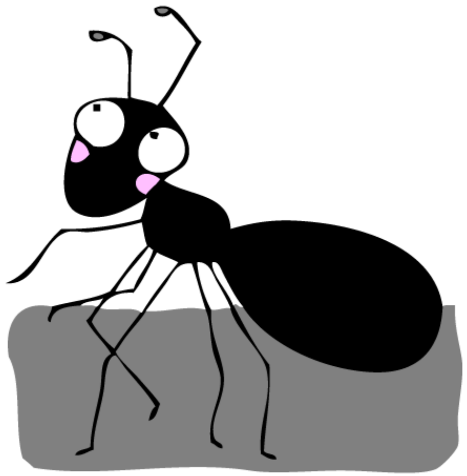
- Directed graph G = (V,E)
- Each test case represents a vertice on the graph
  - *V* has *M* elements
- All vertices are connected with all others
- Pheromone equally distributed

The Proposed Approach

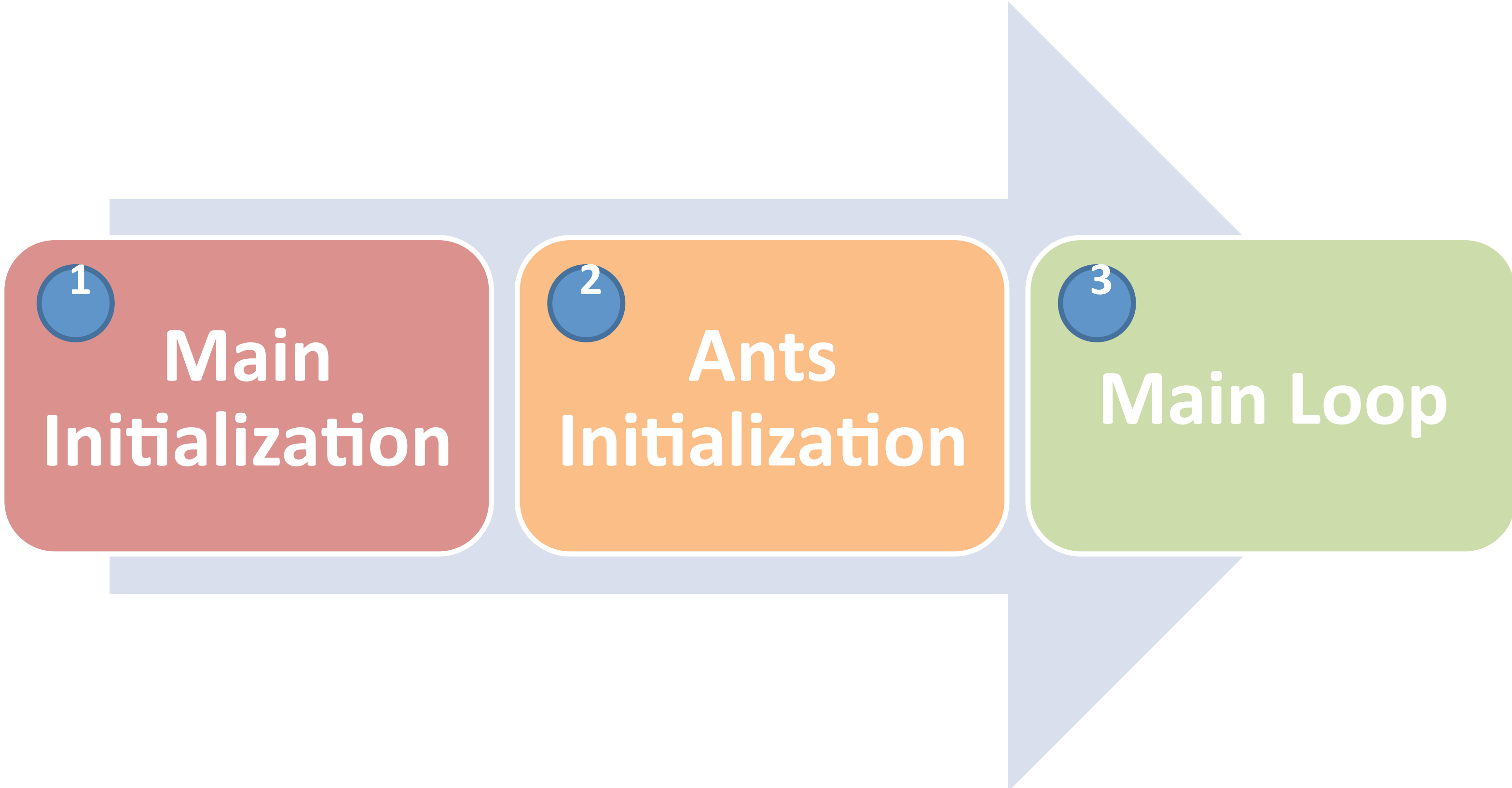
- Each **ant** has the following information
  - nextNode
  - visitedNodes
  - allowedNodes
- Heuristic Function

$$\frac{score_j}{executionTime_j}$$

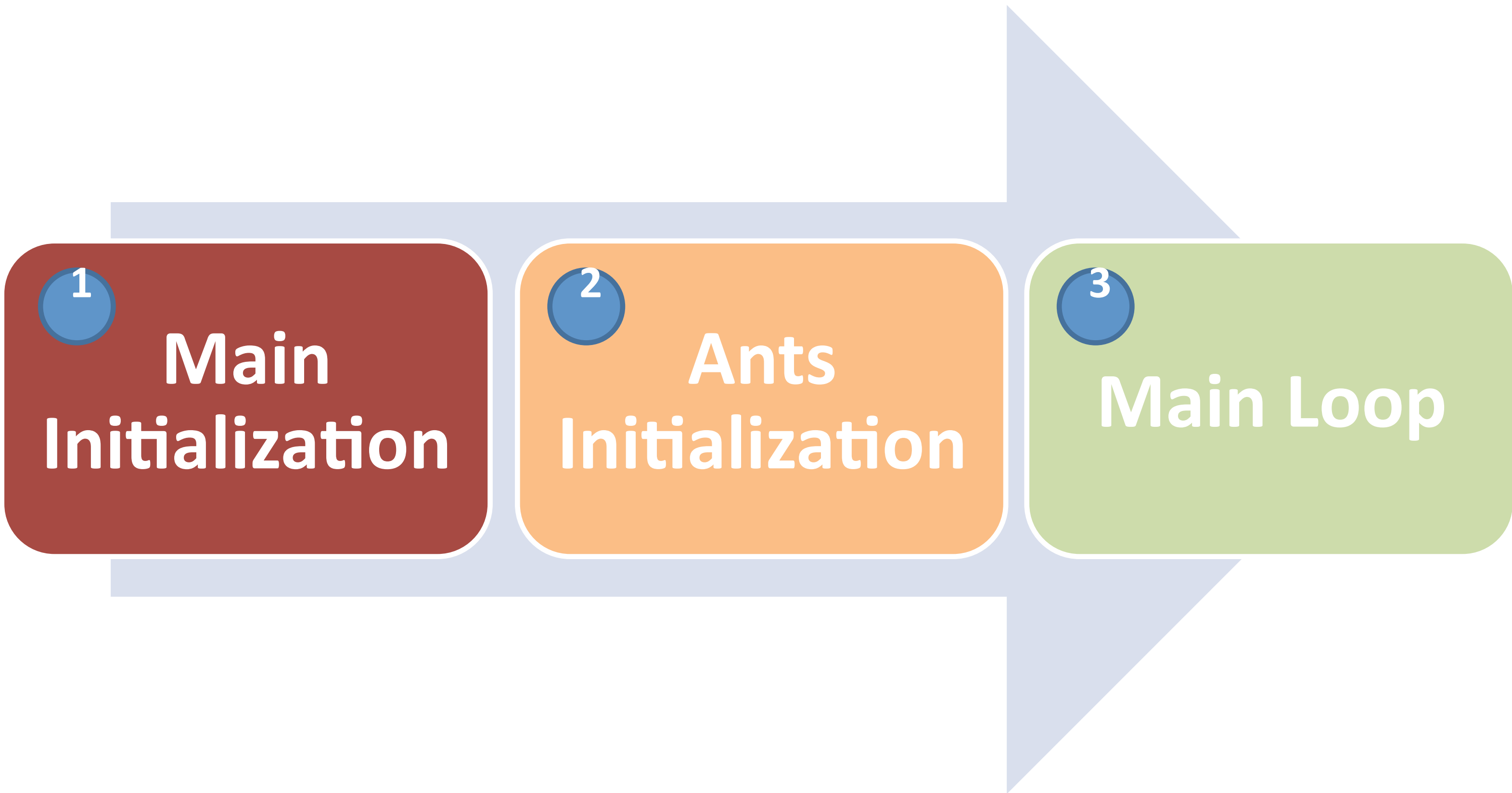
THE PROPOSED ACO-BASED ALGORITHM



Steps of the Algorithm



Steps of the Algorithm



MAIN_INITIALIZATION

ITERATION = 0

Read test case precedence information

Generate directed graph

Initialize pheromone

MAIN_INITIALIZATION

ITERATION = 0

Read test case precedence information

Generate directed graph

Initialize pheromone

MAIN_INITIALIZATION

ITERATION = 0

Read test case precedence information

Generate directed graph

Initialize pheromone

MAIN_INITIALIZATION

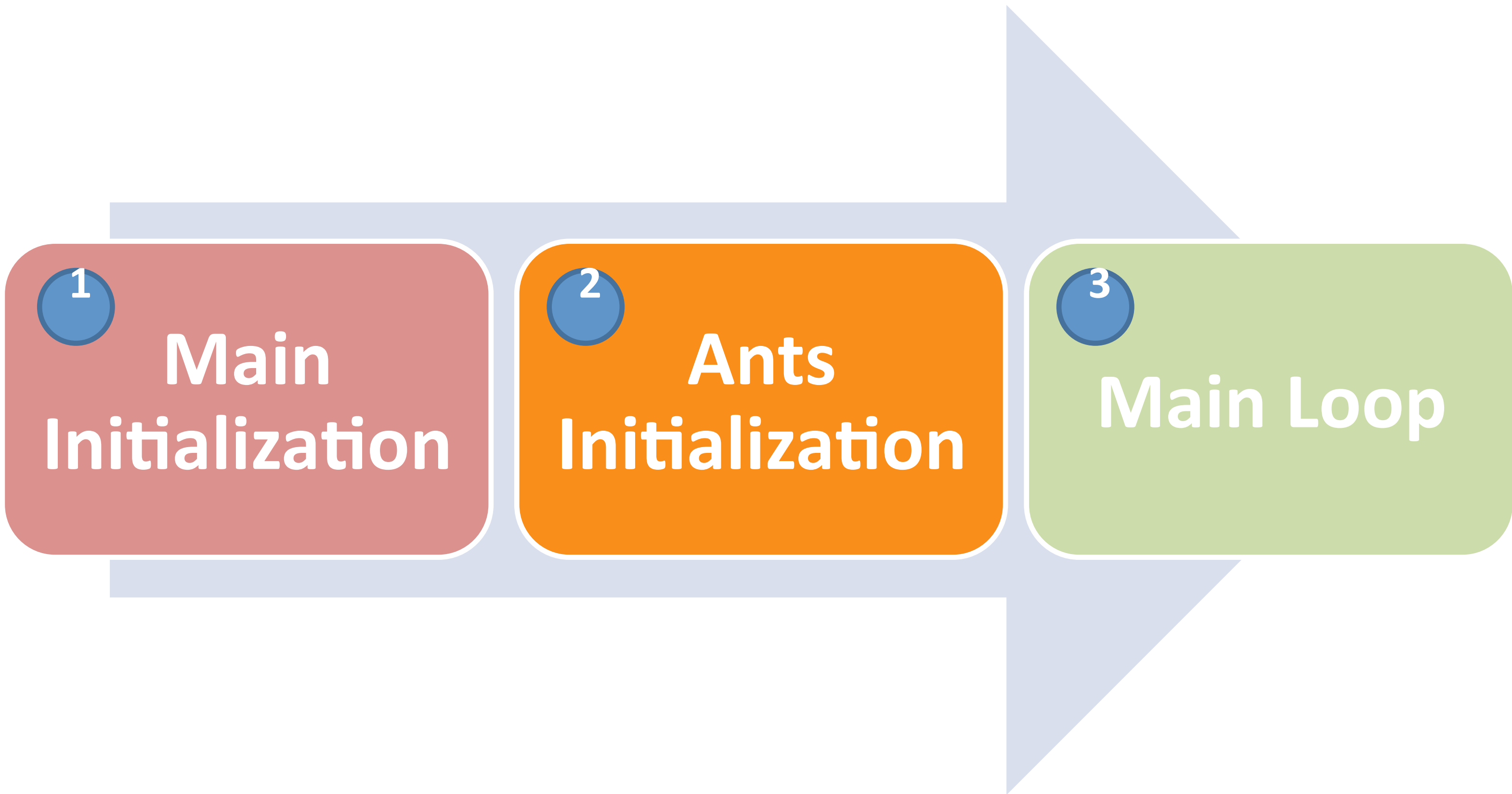
ITERATION = 0

Read test case precedence information

Generate directed graph

Initialize pheromone

Steps of the Algorithm



ANTS_INITIALIZATION

FOR ALL ants

FOR ALL vertices t_j of V, $visited_j \leftarrow$ False

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

ant_k.nextNode = FIND_INITIAL_NODE()

ant_k.visitedNode.add(nextNode)

ANTS_INITIALIZATION

FOR ALL ants

FOR ALL vertices t_j of V, $visited_j \leftarrow$ False

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

ant_k.nextNode = FIND_INITIAL_NODE()

ant_k.visitedNode.add(nextNode)

ANTS_INITIALIZATION

FOR ALL ants

FOR ALL vertices t_j of V, $visited_j$ ← False

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

ant_k.nextNode = FIND_INITIAL_NODE()

ant_k.visitedNode.add(nextNode)

ANTS_INITIALIZATION

FOR ALL ants

FOR ALL vertices t_j of V, $visited_j \leftarrow$ False

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

ant_k.nextNode = FIND_INITIAL_NODE()

ant_k.visitedNode.add(nextNode)

ANTS_INITIALIZATION

FOR ALL ants

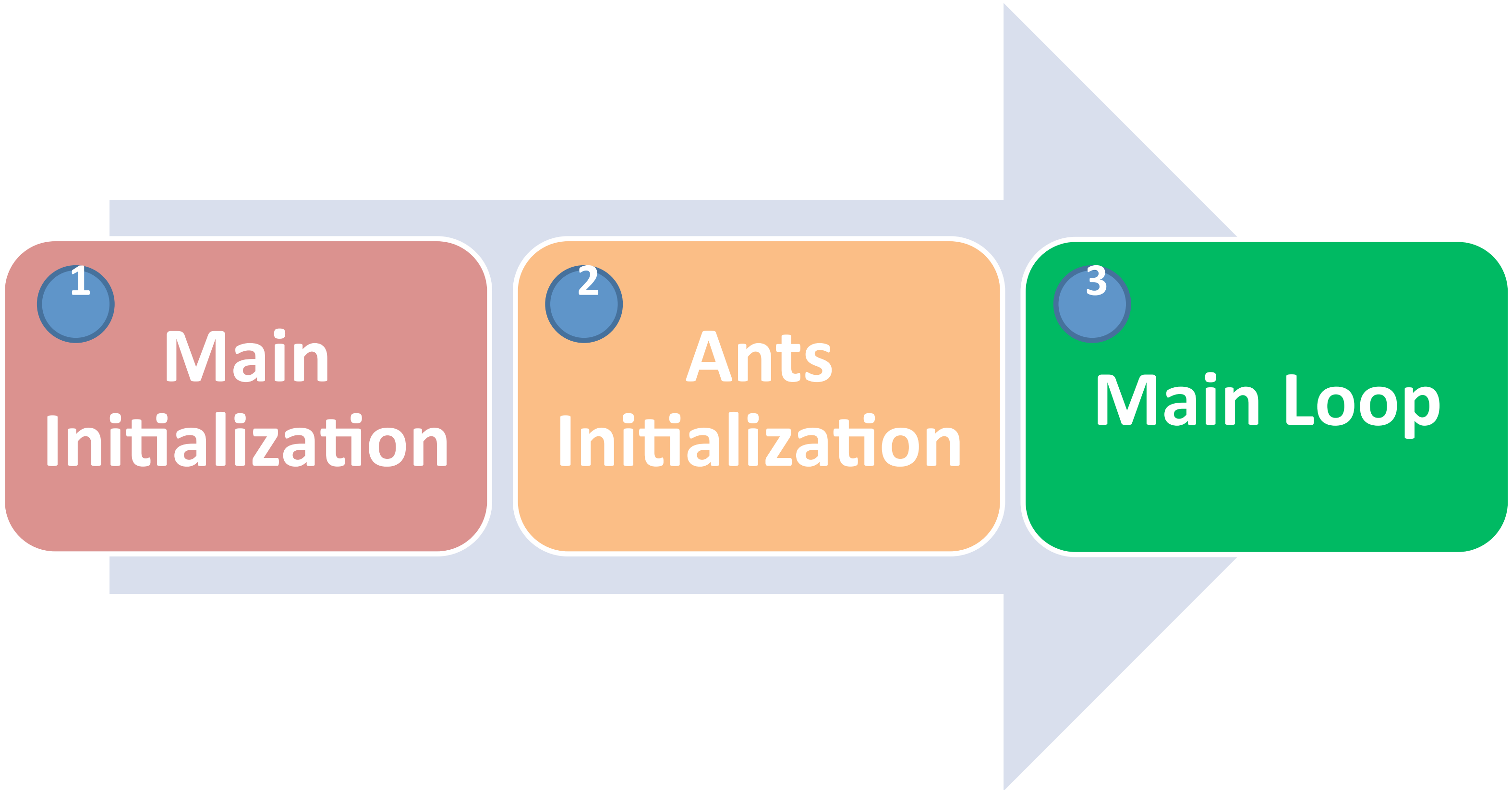
FOR ALL vertices t_j of V, $visited_j \leftarrow$ False

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

ant_k.nextNode = FIND_INITIAL_NODE()

ant_k.visitedNode.add(nextNode)

Steps of the Algorithm



MAIN_LOOP

FOR EACH ant_k

WHILE ant_k.allowedNodes.size() > 0

k = ant_k.actualNode

ant_k.nextNode = ant_k.FIND_NEXT_NODE()

j = ant_k.actualNode

ant_k.visitedNodes.add(ant_k.nextNode)

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

Update pheromone in edge (k,j),with

$$\tau_{kj} = (1 - \varphi) \cdot \tau_{kj} + \varphi \cdot \tau_0$$

MAIN_LOOP

FOR EACH ant_k

WHILE ant_k.allowedNodes.size() > 0

k = ant_k.actualNode

ant_k.nextNode = ant_k.FIND_NEXT_NODE()

j = ant_kactualNode

ant_k.visitedNodes.add(ant_k.nextNode)

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

Update pheromone in edge (k,j),with

$$\tau_{kj} = (1 - \varphi) \cdot \tau_{kj} + \varphi \cdot \tau_0$$

MAIN_LOOP

FOR EACH ant_k

WHILE ant_k.allowedNodes.size() > 0

k = ant_k.actualNode

ant_k.nextNode = ant_k.FIND_NEXT_NODE()

j = ant_kactualNode

ant_k.visitedNodes.add(ant_k.nextNode)

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

Update pheromone in edge (k,j),with

$$\tau_{kj} = (1 - \varphi) \cdot \tau_{kj} + \varphi \cdot \tau_0$$

MAIN_LOOP

FOR EACH ant_k

WHILE ant_k.allowedNodes.size() > 0

k = ant_k.actualNode

ant_k.nextNode = ant_k.FIND_NEXT_NODE()

j = ant_kactualNode

ant_k.visitedNodes.add(ant_k.nextNode)

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

Update pheromone in edge (k,j),with

$$\tau_{kj} = (1 - \varphi) \cdot \tau_{kj} + \varphi \cdot \tau_0$$

MAIN_LOOP

FOR EACH ant_k

WHILE ant_k.allowedNodes.size() > 0

k = ant_k.actualNode

ant_k.nextNode = ant_k.FIND_NEXT_NODE()

j = ant_kactualNode

ant_k.visitedNodes.add(ant_k.nextNode)

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

Update pheromone in edge (k,j),with

$$\tau_{kj} = (1 - \varphi) \cdot \tau_{kj} + \varphi \cdot \tau_0$$

MAIN_LOOP

FOR EACH ant_k

WHILE ant_k.allowedNodes.size() > 0

k = ant_k.actualNode

ant_k.nextNode = ant_k.FIND_NEXT_NODE()

j = ant_kactualNode

ant_k.visitedNodes.add(ant_k.nextNode)

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

Update pheromone in edge (k,j),with

$$\tau_{kj} = (1 - \varphi) \cdot \tau_{kj} + \varphi \cdot \tau_0$$

MAIN_LOOP

FOR EACH ant_k

WHILE ant_k.allowedNodes.size() > 0

k = ant_k.actualNode

ant_k.nextNode = ant_k.FIND_NEXT_NODE()

j = ant_kactualNode

ant_k.visitedNodes.add(ant_k.nextNode)

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

Update pheromone in edge (k,j),with

$$\tau_{kj} = (1 - \varphi) \cdot \tau_{kj} + \varphi \cdot \tau_0$$

MAIN_LOOP

FOR EACH ant_k

WHILE ant_k.allowedNodes.size() > 0

k = ant_k.actualNode

ant_k.nextNode = ant_k.FIND_NEXT_NODE()

j = ant_kactualNode

ant_k.visitedNodes.add(ant_k.nextNode)

ant_k.allowedNodes = UPDATE_ALLOWED_NODES()

Update pheromone in edge (k,j),with

$$\tau_{kj} = (1 - \varphi) \cdot \tau_{kj} + \varphi \cdot \tau_0$$

MAIN_LOOP_FINALIZATION

```
currentSolution = EVALUATE_BEST_SOLUTION()
IF ((bestSolution.value() is null)
        or (bestSolution.value() < currentSolution.value())) THEN
            bestSolution = currentSolution
ITERATION ++
```

MAIN_LOOP_FINALIZATION

```
currentSolution = EVALUATE_BEST_SOLUTION()
IF ((bestSolution.value() is null)
        or (bestSolution.value() < currentSolution.value())) THEN
            bestSolution = currentSolution
ITERATION ++
```

MAIN_LOOP_FINALIZATION

```
currentSolution = EVALUATE_BEST_SOLUTION()
IF ((bestSolution.value() is null)
        or (bestSolution.value() < currentSolution.value())) THEN
            bestSolution = currentSolution
ITERATION ++
```

MAIN_LOOP_FINALIZATION

```
currentSolution = EVALUATE_BEST_SOLUTION()
IF ((bestSolution.value() is null)
        or (bestSolution.value() < currentSolution.value())) THEN
            bestSolution = currentSolution
ITERATION ++
```

The bestSolution variable is the result of the algorithm

ant_k.UPDATE_ALLOWED_NODES()

```
ant_k.allowedNodes.clear()
FOR ALL vertices t_i ∈ V
    IF (t_i respects the precedence constraint and is not yet in solution)
        ant_k.allowedNodes.add(t_i)
```

ant$_k$.UPDATE_ALLOWED_NODES()

```
ant_k.allowedNodes.clear()
FOR ALL vertices t_i ∈ V
    IF (t_i respects the precedence constraint and is not yet in solution)
        ant_k.allowedNodes.add(t_i)
```

ant_k.UPDATE_ALLOWED_NODES()

ant_k.allowedNodes.clear()

FOR ALL vertices $t_i \in V$

IF (t_i respects the precedence constraint and is not yet in solution)

ant_k.allowedNodes.add(t_i)

ant_k.UPDATE_ALLOWED_NODES()

ant_k.allowedNodes.clear()

FOR ALL vertices $t_i \in V$

IF (t_i respects the precedence constraint and is not yet in solution)

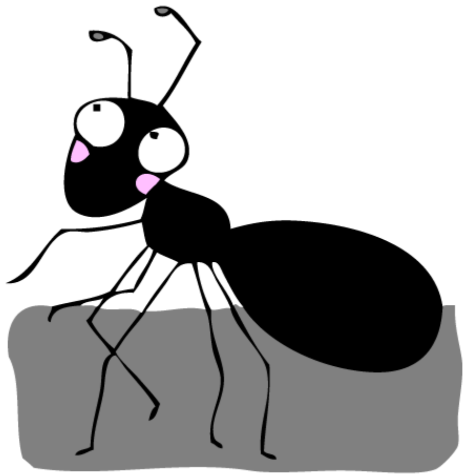
ant_k.allowedNodes.add(t_i)

ant$_k$.FIND_NEXT_NODE()

Move ant *k* to a vertex t_i with probability p_{ij}^{k} or considering $max(\tau_{ij} \cdot w_j^{\beta})$, considering only nodes in allowedNodes set

Return t_i

CONCLUSIONS AND FUTURE WORKS



Conclusions

- Currently
  - Implementation phase
  - Preliminary results not already to present
- Future
  - Evaluate and compare to other search-based algorithms

Camila Maia
camila.maia@gmail.com