Ant Colony Optimization for Constraint Satisfaction

Christine Solnon

LIRIS, UMR 5205 CNRS / University of Lyon

Tutorial at CP’2007
Table of contents

1. Basic principles of Ant Colony Optimization
2. Application to the car sequencing problem
3. Application to binary CSPs
4. Conclusion
A metaheuristic inspired by real ants

Foraging behavior of real ants

- Simple and autonomous agents
- Indirect communication via the environment (stigmergy)

\[
\begin{align*}
\text{Ants lay pheromone trails while walking} \\
\text{Ants randomly choose paths w.r.t. phero. trails} \\
Pheromone trails evaporate
\end{align*}
\]

⇒ Shortest path between the nest and a food source

⇝ Collective problem solving / swarm intelligence
Brief history of ACO

**Ant System**
[Dorigo 92]: application to the Travelling Salesman Problem

**Extensions of Ant System**
Ant Colony System [Dorigo & Gambardella 97],
$\text{MAX} - \text{MIN}$ Ant System [Stützle & Hoos 00],
Hyper-cube Ant System [Blum, Roli & Dorigo 01], ...

**Many applications**
Vehicle routing, Sequential ordering, Quadratic assignment,
Graph coloring, Open shop, Maximum clique, ...

**Generalization**
Ant Colony Optimization (ACO) metaheuristic
# How to solve a problem with ACO?

## 1) Define a construction graph
- Solution components $\leadsto$ vertices or edges
- Solution $\leadsto$ best path in the graph

## 2) Use artificial ants to search for good paths
- Behavior inspired by real ants...
  - Ants lay pheromone on vertices and/or edges
  - Greedy randomized construction of paths w.r.t. pheromone
  - Pheromone evaporates
- ...with extra capabilities
  - Pheromone laying is delayed and proportional to solution quality,
  - Pheromone is combined with problem-dependent heuristics,
  - Hybridation with local search, ...
The MAX – MIN Ant System

- initialize pheromone trails to $\tau_{max}$
- repeat
  1. each ant builds a path
  2. update pheromone trails
- until optimal solution found or stagnation
The \textit{MAX} − \textit{MIN} Ant System

- initialize pheromone trails to $\tau_{\text{max}}$
- repeat
  1. each ant builds a path
  2. update pheromone trails
- until optimal solution found or stagnation

Greedy randomized construction of a path

- Let $C$ = visited vertices and $cand$ = candidate vertices
- Choose $v_j \in cand$ with probability

$$P(v_j) = \frac{[\tau_C(v_j)]^\alpha \cdot [\eta_C(v_j)]^\beta}{\sum_{v_k \in cand} [\tau_C(v_k)]^\alpha \cdot [\eta_C(v_k)]^\beta}$$
The MAX – MIN Ant System

- initialize pheromone trails to $\tau_{max}$
- repeat
  1. each ant builds a path
  2. update pheromone trails
- until optimal solution found or stagnation

Greedy randomized construction of a path
- Let $C = \text{visited vertices}$ and $cand = \text{candidate vertices}$
- Choose $v_j \in cand$ with probability

$$p(v_j) = \frac{[\tau_C(v_j)]^\alpha \cdot [\eta_C(v_j)]^\beta}{\sum_{v_k \in cand} [\tau_C(v_k)]^\alpha \cdot [\eta_C(v_k)]^\beta}$$

$\tau_C(v_j)$ pheromone factor (past experience of the colony)
The *MAX − MIN* Ant System

- initialize pheromone trails to $\tau_{max}$
- repeat
  1. each ant builds a path
  2. update pheromone trails
- until optimal solution found or stagnation

Greedy randomized construction of a path

- Let $C = \text{visited vertices}$ and $cand = \text{candidate vertices}$
- Choose $v_j \in cand$ with probability

$$p(v_j) = \frac{[\tau_C(v_j)]^\alpha \cdot [\eta_C(v_j)]^\beta}{\sum_{v_k \in cand}[\tau_C(v_k)]^\alpha \cdot [\eta_C(v_k)]^\beta}$$

$\eta_C(v_j)$ $\rightsquigarrow$ heuristic factor (problem-dependent)
The MAX – MIN Ant System

- initialize pheromone trails to $\tau_{max}$
- repeat
  1. each ant builds a path
  2. update pheromone trails
- until optimal solution found or stagnation

Greedy randomized construction of a path
- Let $C =$ visited vertices and $cand =$ candidate vertices
- Choose $v_j \in cand$ with probability
  $$p(v_j) = \frac{[\tau_C(v_j)]^\alpha \cdot [\eta_C(v_j)]^\beta}{\sum_{v_k \in cand} [\tau_C(v_k)]^\alpha \cdot [\eta_C(v_k)]^\beta}$$

$\alpha, \beta \sim$ factor weights (parameters)
The \textit{MAX} – \textit{MIN} Ant System

- initialize pheromone trails to $\tau_{max}$
- repeat
  1. each ant builds a path
  2. update pheromone trails
- until optimal solution found or stagnation

Pheromone updating step

- Evaporation: multiply pheromone trails by $(1 - \rho)$
  $\Rightarrow \rho =$ evaporation rate $(0 \leq \rho \leq 1)$
- Reward: add pheromone on the best path components
- Bound pheromone trails between $\tau_{min}$ and $\tau_{max}$
  $\Rightarrow$ prevent from premature stagnation
Example: Travelling Salesman Problem

Construction graph

- Complete graph that associates a vertex with each city
- Pheromone is laid on edges:
  \[ \tau(i, j) \Rightarrow \text{desirability of visiting } j \text{ just after } i \]

At each cycle, each ant builds an hamiltonian cycle

- Random choice of the first vertex
- Probability to go to \( j \) for an ant that is on vertex \( i \):
  \[ p(j) = \frac{[\tau(i, j)]^\alpha \cdot [1/d(i, j)]^\beta}{\sum_{k \in \text{cand}} [\tau(i, k)]^\alpha \cdot [1/d(i, k)]^\beta} \]
  where \( \text{cand} = \text{set of non visited vertices} \)

Pheromone updating step

- Evaporation
- Add pheromone on the edges of the best cycle
  Quantity proportionally inverse to the length of the cycle
Example: Travelling Salesman Problem

**Construction graph**
- Complete graph that associates a vertex with each city
- Pheromone is laid on edges: 
  \[ \tau(i, j) \sim \text{desirability of visiting } j \text{ just after } i \]

**At each cycle, each ant builds an hamiltonian cycle**
- Random choice of the first vertex
- Probability to go to \( j \) for an ant that is on vertex \( i \):
  \[
p(j) = \frac{[\tau(i, j)]^\alpha \cdot [1/d(i, j)]^\beta}{\sum_{k \in \text{cand}} [\tau(i, k)]^\alpha \cdot [1/d(i, k)]^\beta}
\]
  where \( \text{cand} = \text{set of non visited vertices} \)

**Pheromone updating step**
- Evaporation
- Add pheromone on the edges of the best cycle
  Quantity proportionally inverse to the length of the cycle
Example: Travelling Salesman Problem

**Construction graph**
- Complete graph that associates a vertex with each city
- Pheromone is laid on edges: $\tau(i, j) \sim \text{desirability of visiting } j \text{ just after } i$

**At each cycle, each ant builds an hamiltonian cycle**
- Random choice of the first vertex
- Probability to go to $j$ for an ant that is on vertex $i$:

$$p(j) = \frac{[\tau(i, j)]^\alpha \cdot [1/d(i, j)]^\beta}{\sum_{k \in \text{cand}} [\tau(i, k)]^\alpha \cdot [1/d(i, k)]^\beta}$$

where $\text{cand} = \text{set of non visited vertices}$

**Pheromone updating step**
- Evaporation
- Add pheromone on the edges of the best cycle Quantity proportionally inverse to the length of the cycle
### ACO w.r.t. other (meta)heuristic approaches

#### Instance-based approaches
- Construction of new solutions by modifying existing ones
  - Genetic algorithms: cross-over + mutation
  - Local search: local moves

#### Model-based approaches
- Construction of new solutions using a probabilistic model
  - Greedy randomized construction:
    - static model
  - Greedy Randomized Adaptive Search Procedure (GRASP):
    - adaptive model
  - Ant Colony Optimization (ACO):
    - model biased by previous experience
  - Estimation of Distribution Algorithms (EDA):
    - model built from a population which evolves
# ACO w.r.t. other (meta)heuristic approaches

## Instance-based approaches

Construction of new solutions by modifying existing ones
- Genetic algorithms: cross-over + mutation
- Local search: local moves

## Model-based approaches

Construction of new solutions using a probabilistic model
- Greedy randomized construction:
  - static model
- Greedy Randomized Adaptive Search Procedure (GRASP):
  - adaptive model
- Ant Colony Optimization (ACO):
  - model biased by previous experience
- Estimation of Distribution Algorithms (EDA):
  - model built from a population which evolves
## Hybridization ACO / Local Search

### Two views of an ACO/LS hybrid approach
- Ants build initial solutions...
  - ... that are improved by local search;
- or LS finds locally optimal solutions...
  - ... that are used by ACO to build new starting points.

### Choice of a local search strategy
- Compromise between CPU time and solution quality
- Usually: simple greedy LS
Intensifying/diversifying search with ACO

**Intensification**
- Goal: Increase search around promising areas
- Means:
  - Add pheromone on components of best solutions
  - Favor the choice of components with high pheromone trails
- Risk: Premature convergence (stagnation)

**Diversification**
- Goal: Explore new areas
- Means:
  - Probabilistic choice of components
  - Bound pheromone trails within $[\tau_{min}, \tau_{max}]$
  - Initialize pheromone trails to $\tau_{max}$
- Risk: convergence to optimality may be too long

⇒ Theoretical proof of convergence to optimality
Influence of ACO parameters on intensification/diversification

\( \tau_{min}, \tau_{max} \) : pheromone lower and upper bounds
\( \Rightarrow \) Intensification increases when \( \tau_{max} - \tau_{min} \) increases

\textit{nbAnts} : number of ants
\( \Rightarrow \) Diversification increases when \textit{nbAnts} increases

\( \alpha \) : weight of the pheromone factor
\( \Rightarrow \) Intensification increases when \( \alpha \) increases

\( \rho \) : pheromone evaporation rate
\( \Rightarrow \) Intensification increases when \( \rho \) increases

The best parameter setting depends on available time!
Illustration on a maximum clique problem

Without pheromone: $\alpha = 0$
Illustration on a maximum clique problem

With a small influence of pheromone: $\alpha = 1$, $\rho = 0.5\%$
Illustration on a maximum clique problem

When increasing the evaporation rate $\rho$ from 0.5% to 1%
Illustration on a maximum clique problem

When increasing the pheromone factor weight $\alpha$ from 1 to 2
Illustration on a maximum clique problem

When increasing the evaporation rate $\rho$ from 1% to 2%
Measuring Intensification/Diversification

**Resampling ratio (RR) ⇝ quantifies diversification**

- \( RR = \frac{\#\{\text{computed solutions}\} - \#\{\text{different computed solutions}\}}{\#\{\text{computed solutions}\}} \)
- Maximal diversification \( \iff 0 \leq RR \leq 1 \Rightarrow \text{Stagnation} \)

**Similarity ratio (SR) ⇝ quantifies intensification**

- \( SR = \text{average similarity of the set } S \text{ of computed solutions} \)
  \( \iff \text{average similarity of pairs of solutions of } S \)
  \( \iff \text{similarity of 2 solutions} = \text{percentage of shared components} \)
- SR increases when search is intensified

These 2 ratio may be computed (nearly) for free with appropriate data structures!
Measuring Intensification/Diversification

**Resampling ratio (RR) \( \leadsto \) quantifies diversification**

\[
RR = \frac{{\#\{\text{computed solutions}\} - \#\{\text{different computed solutions}\}}}{{\#\{\text{computed solutions}\}}}
\]

Maximal diversification \( \iff 0 \leq RR \leq 1 \implies \text{Stagnation} \)

**Similarity ratio (SR) \( \leadsto \) quantifies intensification**

\[
SR = \text{average similarity of the set } S \text{ of computed solutions}
\]
\( \leadsto \text{average similarity of pairs of solutions of } S \)
\( \leadsto \text{similarity of 2 solutions} = \text{percentage of shared components} \)

SR increases when search is intensified

These 2 ratios may be computed (nearly) for free with appropriate data structures!
### Ant Colony Optimization

#### Application to car sequencing

#### Application to CSPs

#### Conclusion

---

**Measuring Intensification/Diversification**

#### Resampling ratio (RR) quantifies diversification

\[ RR = \frac{\#\{\text{computed solutions}\} - \#\{\text{different computed solutions}\}}{\#\{\text{computed solutions}\}} \]

- Maximal diversification: \( 0 \leq RR \leq 1 \) \( \Rightarrow \) Stagnation

#### Similarity ratio (SR) quantifies intensification

- SR = average similarity of the set \( S \) of computed solutions
- \( \sim \) average similarity of pairs of solutions of \( S \)
- \( \sim \) similarity of 2 solutions = percentage of shared components

- SR increases when search is intensified

---

These 2 ratio may be computed (nearly) for free with appropriate data structures!
Measuring intensification/Diversification: Example

### Resampling ratio

<table>
<thead>
<tr>
<th>Number of cycles:</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 1$, $\rho = 0.01$</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>$\alpha = 2$, $\rho = 0.01$</td>
<td>0.00</td>
<td>0.04</td>
<td>0.06</td>
<td>0.07</td>
<td>0.07</td>
</tr>
<tr>
<td>$\alpha = 2$, $\rho = 0.02$</td>
<td>0.06</td>
<td>0.10</td>
<td>0.12</td>
<td>0.13</td>
<td>0.13</td>
</tr>
</tbody>
</table>

### Similarity ratio

![Graph showing similarity ratio with curves for different $\alpha$ and $\rho$ values.](image)
The car sequencing problem

Goal: Sequence cars along an assembly line
- Each car requires a set of options
- Space cars requiring a same option

Example
Set of cars to be sequenced:
The car sequencing problem

Goal: Sequence cars along an assembly line
- Each car requires a set of options
- Space cars requiring the same option

Example
Set of cars to be sequenced:

Sequencing constraints:

\[
\begin{align*}
&\leq \frac{1}{2} ; \\
&\leq \frac{2}{5} ; \\
&\leq \frac{1}{5} ; \\
&\leq \frac{1}{3}
\end{align*}
\]
The car sequencing problem

Goal: Sequence cars along an assembly line
- Each car requires a set of options
- Space cars requiring a same option

Example
Set of cars to be sequenced:

Sequencing constraints:

\[ \leq \frac{1}{2} ; \quad \leq \frac{2}{5} ; \quad \leq \frac{1}{5} ; \quad \leq \frac{1}{3} \]

Solution:
Greedy randomized algorithm
  \implies problem-dependent heuristic for identifying critical cars

ACO 1
  \implies pheromone structure for identifying good car sequences

ACO 2
  \implies pheromone structure for identifying critical cars

ACO 1+2
  \implies combining the two pheromone structures
Greedy randomized algorithm

- Start from an empty sequence $\pi$
- While not all cars have been sequenced in $\pi$:
  - Let $cand$ be the set of cars not sequenced in $\pi$
  - Narrow $cand$ to cars that
    - introduce the fewest new constraint violations
    - require different sets of options
  - Choose $c_i \in cand$ w.r.t. probability $p(c_i, cand, \pi)$
  - Add $c_i$ at the end of $\pi$
Greedy randomized algorithm

- Start from an empty sequence $\pi$
- While not all cars have been sequenced in $\pi$:
  - Let $cand$ be the set of cars not sequenced in $\pi$
  - Narrow $cand$ to cars that
    - introduce the fewest new constraint violations
    - require different sets of options
  - Choose $c_i \in cand$ w.r.t. probability $p(c_i, cand, \pi)$
  - Add $c_i$ at the end of $\pi$

$$p(c_i, cand, \pi) = \frac{[\eta(c_i, \pi)]^\beta}{\sum_{c_k \in cand} [\eta(c_k, \pi)]^\beta}$$

Where
- $\eta(c_i, \pi)$ = problem-dependent heuristic function
  \sim sum of utilisation rates of options required by $c_i$
- $\beta$ = parameter that controls heuristic weight
ACO algorithm for learning good sequences

Construction graph

- Complete directed graph s.t. vertices = cars
  - Hamiltonian path = sequence of cars
- Pheromone is laid on edges: $\tau_1(c_i, c_j) =$ pheromone on $(c_i, c_j)$
  - experience of the colony / sequencing $c_j$ just after $c_i$

At each cycle, each ant builds a sequence

Probability of adding car $c_j$ at the end of a sequence $\pi$

$$p(c_j) = \frac{[\tau_\pi(c_j)]^{\alpha_1} \cdot [\eta_\pi(c_j)]^{\beta}}{\sum_{c_k \in \text{cand}} [\tau_\pi(c_k)]^{\alpha_1} \cdot [\eta_\pi(c_k)]^{\beta}}$$
ACO algorithm for learning good sequences

**Construction graph**

- Complete directed graph s.t. vertices = cars
  - Hamiltonian path = sequence of cars
- Pheromone is laid on edges: $\tau_1(c_i, c_j) = \text{pheromone on } (c_i, c_j)$
  - experience of the colony / sequencing $c_j$ just after $c_i$

**At each cycle, each ant builds a sequence**

Probability of adding car $c_j$ at the end of a sequence $\pi$

\[
p(c_j) = \frac{[\tau_\pi(c_j)]^{\alpha_1} \cdot [\eta_\pi(c_j)]^\beta}{\sum_{c_k \in \text{cand}} [\tau_\pi(c_k)]^{\alpha_1} \cdot [\eta_\pi(c_k)]^\beta}
\]
ACO algorithm for learning good sequences

### Construction graph
- Complete directed graph s.t. vertices = cars
  - Hamiltonian path = sequence of cars
- Pheromone is laid on edges: $\tau_1(c_i, c_j) = \text{pheromone on } (c_i, c_j)$
  - Experience of the colony / sequencing $c_j$ just after $c_i$

### At each cycle, each ant builds a sequence
- Probability of adding car $c_j$ at the end of a sequence $\pi$

\[
p(c_j) = \frac{[\tau_\pi(c_j)]^{\alpha_1} \cdot [\eta_\pi(c_j)]^{\beta}}{ \sum_{c_k \in \text{cand}}[\tau_\pi(c_k)]^{\alpha_1} \cdot [\eta_\pi(c_k)]^{\beta} }
\]

- $\tau_\pi(c_j) = \text{pheromone factor}$
  - if the last car of $\pi$ is $c_i$ then $\tau_\pi(c_j) = \tau_1(c_i, c_j)$
  - when choosing the first car of a sequence, $\tau_\pi(c_j) = 1$
## ACO algorithm for learning good sequences

### Construction graph
- Complete directed graph s.t. vertices = cars
  - Hamiltonian path = sequence of cars
- Pheromone is laid on edges: \( \tau_1(c_i, c_j) = \text{pheromone on } (c_i, c_j) \)
  - experience of the colony / sequencing \( c_j \) just after \( c_i \)

### At each cycle, each ant builds a sequence

Probability of adding car \( c_j \) at the end of a sequence \( \pi \)

\[
p(c_j) = \frac{[\tau_\pi(c_j)]^{\alpha_1} \cdot [\eta_\pi(c_j)]^{\beta}}{\sum_{c_k \in \text{cand}} [\tau_\pi(c_k)]^{\alpha_1} \cdot [\eta_\pi(c_k)]^{\beta}}
\]

\( \eta_\pi(c_j) = \text{local heuristic that evaluates the hardness of } c_j \)
- \( \sim \) sum of utilization rates of options required by \( c_j \)
# ACO algorithm for learning good good sequences

## Construction graph
- Complete directed graph s.t. vertices = cars
  - Hamiltonian path = sequence of cars
- Pheromone is laid on edges: $\tau_1(c_i, c_j)$ = pheromone on $(c_i, c_j)$
  - experience of the colony / sequencing $c_j$ just after $c_i$

## At each cycle, each ant builds a sequence
- Probability of adding car $c_j$ at the end of a sequence $\pi$
  \[ p(c_j) = \frac{[\tau_\pi(c_j)]^{\alpha_1} \cdot [\eta_\pi(c_j)]^{\beta}}{\sum_{c_k \in \text{cand}} [\tau_\pi(c_k)]^{\alpha_1} \cdot [\eta_\pi(c_k)]^{\beta}} \]

## At the end of each cycle, reward the best sequences
- Add pheromone on consecutive cars in the cycle best sequences
- Quantity of pheromone added = $1$/number of violated constraints
ACO algorithm for learning for critical cars

Pheromone structure

Pheromone trails associated with cars
(grouped in car classes w.r.t. required options):

\[ \tau_2(cc) = \text{quantity of pheromone associated with car class } cc \]
\[ \tau_2(cc) = \text{experience of the colony / difficulty of sequencing cars of } cc \]

At each cycle, each ant builds a sequence

Probability of adding a car of class \( cc_j \) at the end of a sequence

\[ p(cc_j) = \frac{[\tau_2(cc_j)]^{\alpha_2}}{\sum_{cc_k \in \text{cand}}[\tau(cc_k)]^{\alpha_2}} \]

Pheromone updating step

- While constructing sequences:
  \[ \tau_2 \text{ add pheromone on classes violating constraints} \]
- At the end of every sequence construction:
  \[ \tau_2 \text{ evaporation} \]
ACO algorithm for learning for critical cars

**Pheromone structure**

Pheromone trails associated with cars (grouped in car classes w.r.t. required options):

\[ \tau_2(\text{cc}) = \text{quantity of pheromone associated with car class } \text{cc} \]

\[ \tau_2(\text{cc}) = \text{experience of the colony / difficulty of sequencing cars of } \text{cc} \]

**At each cycle, each ant builds a sequence**

Probability of adding a car of class \( \text{cc}_j \) at the end of a sequence

\[ p(\text{cc}_j) = \frac{[\tau_2(\text{cc}_j)]^\alpha}{\sum_{\text{cc}_k \in \text{cand}} [\tau(\text{cc}_k)]^\alpha} \]

**Pheromone updating step**

- While constructing sequences:
  - add pheromone on classes violating constraints
- At the end of every sequence construction:
  - evaporation
ACO algorithm for learning for critical cars

Pheromone structure

Pheromone trails associated with cars (grouped in car classes w.r.t. required options):
- $\tau_2(cc) =$ quantity of pheromone associated with car class $cc$
- $\tau_2$ experience of the colony / difficulty of sequencing cars of $cc$

At each cycle, each ant builds a sequence

Probability of adding a car of class $cc_j$ at the end of a sequence:

$$p(cc_j) = \frac{[\tau_2(cc_j)]^{\alpha_2}}{\sum_{cc_k \in cand} [\tau(cc_k)]^{\alpha_2}}$$

Pheromone updating step

- While constructing sequences:
  - $\tau$ add pheromone on classes violating constraints
- At the end of every sequence construction:
  - $\tau$ evaporation
Double ACO algorithm

Combine the two pheromone structures

- To learn for sequences: ∀ cars $c_i$ and $c_j$
  \[ \tau_1(c_i, c_j) = \text{experience of the colony / sequence } c_j \text{ after } c_i \]
- To learn for critical cars: ∀ car class $cc$
  \[ \tau_2(cc) = \text{experience of the colony / sequence cars of } cc \]

At each cycle, each ant builds a sequence

Probability of adding car $c_j$ at the end of a sequence

\[
p(c_j) = \frac{[\tau_1(c_i,c_j)]^{\alpha_1} [\tau_2(\text{classOf}(c_j))]^{\alpha_2}}{\sum_{c_k \in \text{cand}} [\tau_1(c_i,c_k)]^{\alpha_1} [\tau_2(\text{classOf}(c_k))]^{\alpha_2}}
\]

Pheromone updating steps

- While constructing sequences: add trails on $\tau_2(cc)$
- At the end of every sequence construction: evaporate $\tau_2(cc)$
- At the end of every cycle: evaporate + add trails on $\tau_1(c_i, c_j)$
Double ACO algorithm

Combine the two pheromone structures

- To learn for sequences: \( \forall \) cars \( c_i \) and \( c_j \)
  \[ \tau_1(c_i, c_j) = \text{experience of the colony / sequence } c_j \text{ after } c_i \]
- To learn for critical cars: \( \forall \) car class \( cc \)
  \[ \tau_2(cc) = \text{experience of the colony / sequence cars of } cc \]

At each cycle, each ant builds a sequence

Probability of adding car \( c_j \) at the end of a sequence

\[
p(c_j) = \frac{[\tau_1(c_i, c_j)]^{\alpha_1} \cdot [\tau_2(\text{classOf}(c_j))]^{\alpha_2}}{\sum_{c_k \in \text{cand}}[\tau_1(c_i, c_k)]^{\alpha_1} \cdot [\tau_2(\text{classOf}(c_k))]^{\alpha_2}}
\]

Pheromone updating steps

- \( \Rightarrow \) While constructing sequences: add trails on \( \tau_2(cc) \)
- \( \Rightarrow \) At the end of every sequence construction: evaporate \( \tau_2(cc) \)
- \( \Rightarrow \) At the end of every cycle: evaporate + add trails on \( \tau_1(c_i, c_j) \)
Double ACO algorithm

Combine the two pheromone structures

- To learn for sequences: \( \forall \) cars \( c_i \) and \( c_j \)
  \[ \tau_1(c_i, c_j) = \text{experience of the colony / sequence } c_j \text{ after } c_i \]

- To learn for critical cars: \( \forall \) car class \( cc \)
  \[ \tau_2(cc) = \text{experience of the colony / sequence cars of } cc \]

At each cycle, each ant builds a sequence

Probability of adding car \( c_j \) at the end of a sequence

\[
\rho(c_j) = \frac{[\tau_1(c_i, c_j)]^{\alpha_1} \cdot [\tau_2(\text{classOf}(c_j))]^{\alpha_2}}{\sum_{c_k \in \text{cand}} [\tau_1(c_i, c_k)]^{\alpha_1} \cdot [\tau_2(\text{classOf}(c_k))]^{\alpha_2}}
\]

Pheromone updating steps

- While constructing sequences: add trails on \( \tau_2(cc) \)
- At the end of every sequence construction: evaporate \( \tau_2(cc) \)
- At the end of every cycle: evaporate + add trails on \( \tau_1(c_i, c_j) \)
## Experimental results

### Experimental settings

<table>
<thead>
<tr>
<th>Algo</th>
<th>Heur. $\eta$</th>
<th>Pheromone 1</th>
<th>Pheromone 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\beta$</td>
<td>$\alpha_1$</td>
<td>$\tau_{min_1}$</td>
</tr>
<tr>
<td>Greedy($\eta$)</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$ACO(\tau_1, \eta)$</td>
<td>6</td>
<td>2</td>
<td>1%</td>
</tr>
<tr>
<td>$ACO(\tau_2)$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$ACO(\tau_1, \tau_2)$</td>
<td>-</td>
<td>2</td>
<td>1%</td>
</tr>
</tbody>
</table>

5000 cycles, 30 ants $\rightarrow$ 150000 sequence constructions
50 runs per instance on a 2GHz Pentium 4

### Benchmark

- Instances of [Lee et al. 98] trivially solved in less than 0.01s
- Instances of [Perron & Shaw 04]
  - Satisfiable instances
  - 32, 21 and 29 instances with 100, 300 and 500 cars resp.
  - 20 classes and 8 options
Experimental results

### Experimental settings

<table>
<thead>
<tr>
<th>Algo</th>
<th>Heur. $\eta$</th>
<th>Pheromone 1</th>
<th>Pheromone 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\beta$ $\alpha_1$ $\rho_1$ $\tau_{min_1}$ $\tau_{max_1}$ $\alpha_2$ $\rho_2$ $\tau_{min_2}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greedy($\eta$)</td>
<td>6 - - - - - -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACO($\tau_1$, $\eta$)</td>
<td>6 2 1% 0.01 4 - - -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACO($\tau_2$)</td>
<td>- - - - - 6 3% 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACO($\tau_1$, $\tau_2$)</td>
<td>- 2 1% 0.01 4 6 3% 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5000 cycles, 30 ants $\rightarrow$ 150000 sequence constructions
50 runs per instance on a 2GHz Pentium 4

### Benchmark

- Instances of [Lee et al. 98] trivially solved in less than 0.01s
- Instances of [Perron & Shaw 04]
  - Satisfiable instances
  - 32, 21 and 29 instances with 100, 300 and 500 cars resp.
  - 20 classes and 8 options
Comparison: success / CPU time

The graph compares the percentage of instances resolved (50 executions / 82 instances) against the CPU time (logarithmic scale) for a specific algorithm, labeled 'Glouton'. The x-axis represents the CPU time, while the y-axis shows the percentage of instances resolved.
Comparison: success / CPU time
Comparison: success / CPU time

Pourcentage d’instances résolues (50 exécutions / 82 instances)

Temps CPU (échelle logarithmique)

Glouton
ACO 1
ACO 2
Comparison: success / CPU time

- Glouton
- ACO 1
- ACO 2
- ACO 1+2
Considered approaches

- **SN** = Winner of the ROADEF’2005 challenge [Estellon et al. 05]
  - First solution built in a greedy way
  - Neighborhood = swap / mirror / insert / shuffle
  - First non decreasing neighbor

- **ID Walk** [Neveu et al. 04]
  - First solution built in a greedy way
  - Neighborhood = swap
  - at most $Max$ neighbors are considered at each move
    - ⇝ First non decreasing neighbor...
    - ... or best neighbor over $Max$ neighbors
    - ⇝ Adaptive tuning of $Max$
Comparison: 32 instances with 100 cars

![Comparison Graph]

- ACO 1+2
- SN
- IDWalk

- Temps CPU (échelle logarithmique)
- Pourcentage d'instances résolues (50 exécutions / 32 instances)
Comparison: 21 instances with 300 cars
Comparison: 29 instances with 500 cars
Table of contents

1. Basic principles of Ant Colony Optimization
2. Application to the car sequencing problem
3. Application to binary CSPs
4. Conclusion
Application of ACO to binary CSPs

Construction graph associated with a CSP \((X, D, C)\)

Complete non directed graph \(G = (V, E)\) s.t.

\[
V = \{\langle X_i, v_i \rangle : X_i \in X, v_i \in D(X_i)\}
\]

Two different pheromone strategies

- **Vertex strategy:** pheromone is laid on vertices
  \[\tau_{\langle X_i, v_i \rangle} \leadsto \text{desirability of assigning } X_i \text{ to } v_i\]

- **Edge strategy:** pheromone is laid on edges
  \[\tau_{\langle X_i, v_i \rangle, \langle X_j, v_j \rangle} \leadsto \text{desir. of assigning together } X_i \text{ to } v_i \text{ and } X_j \text{ to } v_j\]
Application of ACO to CSP

At each cycle, each ant constructs a complete assignment $A$

Given a partial assignment $A$:
- Choose a non assigned variable $X_i$ w.r.t. min domain heuristic
- Choose a value $v_i \in D(X_i)$ with probability

$$p(v_i) = \frac{\tau_{\text{factor}}(v_i)^\alpha \cdot \eta_{\text{factor}}(v_i)^\beta}{\sum_{v_k \in D(X_i)} \tau_{\text{factor}}(v_k)^\alpha \cdot \eta_{\text{factor}}(v_k)^\beta}$$
Application of ACO to CSP

At each cycle, each ant constructs a complete assignment $A$

Given a partial assignment $A$:
- Choose a non assigned variable $X_i$ w.r.t. min domain heuristic
- Choose a value $v_i \in D(X_i)$ with probability

\[
p(v_i) = \frac{\tau_{\text{factor}}(v_i)^{\alpha} \cdot \eta_{\text{factor}}(v_i)^{\beta}}{\sum_{v_k \in D(X_i)} \tau_{\text{factor}}(v_k)^{\alpha} \cdot \eta_{\text{factor}}(v_k)^{\beta}}
\]

- Pheromone factor depends on the strategy:
  - Vertex strategy $\leadsto \tau_{\text{factor}}(v_i) = \tau\langle X_i, v_i \rangle$
  - Edge strategy $\leadsto \tau_{\text{factor}}(v_i) = \sum \langle X_k, v_k \rangle \in A \tau\langle X_i, v_i \rangle, \langle X_k, v_k \rangle$

- Heuristic factor: $\eta_{\text{factor}}(v_i) = 1/(1 + \text{new violations})$
Application of ACO to CSP

At each cycle, each ant constructs a complete assignment $A$

Given a partial assignment $A$:
- Choose a non assigned variable $X_i$ w.r.t. min domain heuristic
- Choose a value $v_i \in D(X_i)$ with probability

$$p(v_i) = \frac{\tau_{factor}(v_i)^\alpha \cdot \eta_{factor}(v_i)^\beta}{\sum_{v_k \in D(X_i)} \tau_{factor}(v_k)^\alpha \cdot \eta_{factor}(v_k)^\beta}$$

At the end of each cycle, update pheromone

- Evaporate pheromone
  - Vertex strategy: on every vertex $\langle X_i, v_i \rangle$
  - Edge strategy: on every edge $(\langle X_i, v_i \rangle, \langle X_k, v_k \rangle)$
- Add pheromone on the best assignment $A_{best}$ of the cycle
  - Vertex strategy: add pheromone on every $\langle X_i, v_i \rangle \in A_{best}$
  - Edge strategy: add pheromone on every $(\langle X_i, v_i \rangle, \langle X_j, v_j \rangle) \in A_{best} \times A_{best}$
Experimental results: considered instances

CSP solver competition in 2006

- 1195 binary instances defined in extension...
- ...selection of satisfiable instances
- ...selection of instances that pass through my parser

→ 230 instances from 6 benchmarks
Experimental results: ACO experimental setting

Hybridization with local search
- At the end of each assignment construction, perform local search
  - Move = change the value of one variable
  - Accept the first non tabu and non deteriorating move
  - Tabu list of infinite length

Parameter setting
- Choice of a setting that favors a quick convergence:
  \[ \alpha = 2, \beta = 8, \rho = 0.02, 15 \text{ ants}, \tau_{min} = 0.01, \tau_{max} = \delta_{avg}/\rho \]
  (where \( \delta_{avg} \) = average qty of pheromone laid at each cycle)
- Exploitation of the resampling ratio to prevent from stagnation
  - When a solution is resampled, decrease pheromone trails on its components (multiplication by 0.1)
  - Restart if more than 1000 resamplings
## Experimental results: considered solvers

- 23 solvers of the competition, all based on complete approaches
- CPU time limit of 1800s on a 3GHz Intel Xeon
- ACO (Vertex and Edge)
- CPU time limit of 1800s on a 2.16GHz Intel Core Duo
- Tabu (tabu list length=50, restart every 1,000,000 moves)
- CPU time limit of 1800s on a 2.16GHz Intel Core Duo

<table>
<thead>
<tr>
<th>Solver</th>
<th>#solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>buggy 2.5.s</td>
<td>207</td>
</tr>
<tr>
<td>buggy 2.5</td>
<td>207</td>
</tr>
<tr>
<td>Abscon 109 AC</td>
<td>207</td>
</tr>
<tr>
<td>VALCSP 3.1</td>
<td>206</td>
</tr>
<tr>
<td>VALCSP 3.0</td>
<td>206</td>
</tr>
<tr>
<td>Abscon 109 ESAC</td>
<td>204</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

6 best complete solvers of the competition in 2006
Experimental results: considered solvers

- 23 solvers of the competition, all based on complete approaches
  - CPU time limit of 1800s on a 3GHZ Intel Xeon
- ACO (Vertex and Edge)
  - CPU time limit of 1800s on a 2.16GHZ Intel Core Duo
- Tabu (tabu list length=50, restart every 1,000,000 moves)
  - CPU time limit of 1800s on a 2.16GHZ Intel Core Duo

<table>
<thead>
<tr>
<th>Solver</th>
<th>#solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>buggy_2.5_s</td>
<td>207</td>
</tr>
<tr>
<td>buggy_2.5</td>
<td>207</td>
</tr>
<tr>
<td>Abscon 109 AC</td>
<td>207</td>
</tr>
<tr>
<td>VALCSP 3.1</td>
<td>206</td>
</tr>
<tr>
<td>VALCSP 3.0</td>
<td>206</td>
</tr>
<tr>
<td>Abscon 109 ESAC</td>
<td>204</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ACO(Vertex)</td>
<td>218</td>
</tr>
<tr>
<td>ACO(Edge)</td>
<td>214</td>
</tr>
</tbody>
</table>

6 best complete solvers of the competition in 2006
Experimental results: considered solvers

- 23 solvers of the competition, all based on complete approaches
  - CPU time limit of 1800s on a 3GHZ Intel Xeon
- ACO (Vertex and Edge)
  - CPU time limit of 1800s on a 2.16GHZ Intel Core Duo
- Tabu (tabu list length=50, restart every 1,000,000 moves)
  - CPU time limit of 1800s on a 2.16GHZ Intel Core Duo

<table>
<thead>
<tr>
<th>Solver</th>
<th>#solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>buggy_2.5_s</td>
<td>207</td>
</tr>
<tr>
<td>buggy_2.5</td>
<td>207</td>
</tr>
<tr>
<td>Abscon 109 AC</td>
<td>207</td>
</tr>
<tr>
<td>VALCSP 3.1</td>
<td>206</td>
</tr>
<tr>
<td>VALCSP 3.0</td>
<td>206</td>
</tr>
<tr>
<td>Abscon 109 ESAC</td>
<td>204</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ACO(Vertex)</td>
<td>218</td>
</tr>
<tr>
<td>ACO(Edge)</td>
<td>214</td>
</tr>
<tr>
<td>Tabu</td>
<td>226</td>
</tr>
</tbody>
</table>

6 best complete solvers of the competition in 2006
Experimental results

Easy benchmarks (all instances solved in less than 1800 s.)

- composed-25-10-20: 10 instances, $|X| = 105, |D| = 10$
- marc: 5 instances, $|X| \in \{80, 84, 88, 92, 96\}, |D| = |X|$
- rand-2-k: 23 instances, $|X| = k, |D| = k, k \in \{23, 24, 25, 26, 27\}$

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Best complete solver</th>
<th>Vertex</th>
<th>Edge</th>
<th>Tabu</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp-25-10-20</td>
<td>0.06 (VALCSP 3.1)</td>
<td>1.31</td>
<td>8.03</td>
<td>0.32</td>
</tr>
<tr>
<td>marc</td>
<td>19.95 (Abscon 109 AC)</td>
<td>26.96</td>
<td>27.05</td>
<td>28.21</td>
</tr>
<tr>
<td>rand-2-23</td>
<td>14.82 (VALCSP 3.0)</td>
<td>1.42</td>
<td>3.15</td>
<td>5.76</td>
</tr>
<tr>
<td>rand-2-24</td>
<td>9.94 (VALCSP 3.1)</td>
<td>3.22</td>
<td>7.43</td>
<td>8.25</td>
</tr>
<tr>
<td>rand-2-25</td>
<td>52.69 (buggy_2_5_s)</td>
<td>3.20</td>
<td>7.27</td>
<td>6.71</td>
</tr>
<tr>
<td>rand-2-26</td>
<td>180.86 (VALCSP 3.1)</td>
<td>3.26</td>
<td>87.71</td>
<td>21.13</td>
</tr>
<tr>
<td>rand-2-27</td>
<td>248.71 (VALCSP 3.1)</td>
<td>23.45</td>
<td>193.88</td>
<td>26.35</td>
</tr>
</tbody>
</table>

(average CPU time, in seconds, for each class of instances)
Experimental results

More difficult benchmarks

- rand-2-n-k-fcd : 60 instances, $|X| = n$, $|D| = k$
- geom : 92 instances, $|X| = 50$, $|D| = 20$

<table>
<thead>
<tr>
<th></th>
<th>Best complete solver</th>
<th>ACO(Verte‌x)</th>
<th>ACO(Edge)</th>
<th>Tabu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nb</td>
<td>time</td>
<td>solver</td>
<td>nb</td>
</tr>
<tr>
<td>2-30-15</td>
<td>20</td>
<td>0.2</td>
<td>(VALCSP 3.0)</td>
<td>20</td>
</tr>
<tr>
<td>2-40-19</td>
<td>20</td>
<td>25.3</td>
<td>(VALCSP 3.1)</td>
<td>20</td>
</tr>
<tr>
<td>2-50-23</td>
<td>14</td>
<td>829.7</td>
<td>(buggy_2.5)</td>
<td>20</td>
</tr>
<tr>
<td>geom</td>
<td>92</td>
<td>1.0</td>
<td>(VALCSP 3.0)</td>
<td>91</td>
</tr>
</tbody>
</table>

nb = number of instances solved in less than 1800 s.

Time = average CPU time for the solved instances
Experimental results

Really hard benchmark

- frbn-k [Xu et al. / IJCAI’05] : 40 instances, $|X| = n$, $|D| = k$

<table>
<thead>
<tr>
<th></th>
<th>nb</th>
<th>Best solver time solver</th>
<th>ACO(Vert) nb time</th>
<th>ACO(Edge) nb time</th>
<th>Tabu nb time</th>
</tr>
</thead>
<tbody>
<tr>
<td>30-15</td>
<td>5</td>
<td>0.2 (VALCSP 3.1)</td>
<td>5 0.4</td>
<td>5 1.3</td>
<td>5 0.5</td>
</tr>
<tr>
<td>35-17</td>
<td>5</td>
<td>2.1 (VALCSP 3.1)</td>
<td>5 3.0</td>
<td>5 5.0</td>
<td>5 0.9</td>
</tr>
<tr>
<td>40-19</td>
<td>5</td>
<td>9.4 (VALCSP 3.0)</td>
<td>5 7.0</td>
<td>5 103.5</td>
<td>5 9.1</td>
</tr>
<tr>
<td>45-21</td>
<td>5</td>
<td>123.4 (VALCSP 3.1)</td>
<td>5 467.7</td>
<td>5 354.1</td>
<td>5 43.4</td>
</tr>
<tr>
<td>50-23</td>
<td>3</td>
<td>327.2 (VALCSP 3.1)</td>
<td>3 430.4</td>
<td>3 680.5</td>
<td>4 9.9</td>
</tr>
<tr>
<td>53-24</td>
<td>1</td>
<td>74.0 (Abs. 109 AC)</td>
<td>3 105.7</td>
<td>3 530.8</td>
<td>4 291.6</td>
</tr>
<tr>
<td>56-25</td>
<td>0</td>
<td>-</td>
<td>2 535.8</td>
<td>2 170.2</td>
<td>4 329.3</td>
</tr>
<tr>
<td>59-26</td>
<td>1</td>
<td>413.9 (Abs. 109 AC)</td>
<td>1 63.6</td>
<td>0 -</td>
<td>4 523.7</td>
</tr>
</tbody>
</table>

nb = number of instances solved in less than 1800 s.

time = average CPU time for the solved instances
Table of contents

1. Basic principles of Ant Colony Optimization
2. Application to the car sequencing problem
3. Application to binary CSPs
4. Conclusion
From CSP solving to CP

Ants can solve CSPs !!!

- Competitive results...
- ...obtained with *ad hoc* ACO algorithms

Next step: CP with ACO

- Get rid of parameter tuning $\rightarrow$ reactive ACO
  - use diversification measures to adaptively tune parameters

- Integration within a CP language
  - Investigation of two directions:
    - Use ACO to guide a CP search within Ilog solver
    - Design a Comet-based modelling language for ACO
From CSP solving to CP

Ants can solve CSPs !!!
- Competitive results...
- ...obtained with ad hoc ACO algorithms

Next step: CP with ACO
- Get rid of parameter tuning $\rightsquigarrow$ reactive ACO
- Use diversification measures to adaptively tune parameters
- Integration within a CP language
  - Investigation of two directions:
    - Use ACO to guide a CP search within Ilog solver
    - Design a Comet-based modelling language for ACO
Next step: CP with ACO

Using ACO to guide a CP search

Joint project with Ilog (Patrick Albert and Madjid Khichane)

- Use a CP modeling language to describe the problem
- Use ACO to guide the search engine
  - Iterative construction of partial consistent assignments
    - ACO $\rightsquigarrow$ heuristic to choose values
    - CP solver $\rightsquigarrow$ constraint propagation and verification

First promising results on the car sequencing problem...
...see First Workshop on Autonomous Search at CP’2007
Next step: CP with ACO

A Comet-based modelling language for ACO

Joint work with Yves Deville and Pascal Van Hentenryck

- Enrich Comet with new abstractions to ease the design of ACO algorithms
- First investigations on 2 classes of problems:
  - Hamiltonian path finding problems (Traveling Salesman Problem, Car sequencing Problem, ...)
  - Subset selection problems (Multidimensional Knapsack Problem, Maximum Clique Problem, ...)
Some references

- **On ACO and the MAX-MIN Ant System**
  - [http://iridia.ulb.ac.be/~mdorigo/ACO](http://iridia.ulb.ac.be/~mdorigo/ACO)

- **On solving the car sequencing problem with ACO**
  - C. Solnon: Combining two ACO algorithms for solving the car sequencing problem. EJOR (to appear)

- **On solving binary CSPs with ACO**
Generalization to Subset Selection Problems

**Definition**

- An SS-problem is defined by \((S, S_C, f)\) where
  - \(S\) = set of candidate objects
  - \(S_C \subseteq \mathcal{P}(S)\) = set of consistant subsets
  - \(f : S_C \rightarrow \mathbb{R}\) = objective function

- Goal: find \(s^* \in S_C\) such that \(f(s^*)\) is maximal (or minimal)

**Examples**

- Maximum clique problem
- Multidimensionnal knapsack problem
- Set covering problem
- ...  
- Constraint satisfaction problems
### Solving SS-problems with ACO

#### Vertex strategy: identify good objects

Pheromone is laid on objects: $\tau(i) \sim \text{interest of selecting } i$

- Knapsack [Leguizamon & Michalewicz 99]
- Set covering [Hadji & al 00]
- Minimum weight K-trees [Blum 02]
- CSP [Solnon & Bridge 06]
- Maximum clique [Solnon & Fenet 06]

#### Edge strategy: identify good pairs of objects

Pheromone is laid on pairs: $\tau(i, j) \sim \text{interest of selecting } i \text{ and } j$

- CSP [Solnon 02]
- Minimum weight K-trees [Blum 02]
- Maximum clique [Fenet & Solnon 03]
- Knapsack [Alaya, Solnon & Ghédira 04]
- Graph matching [Sammoud, Solnon & Ghédira 05]
Generic ACO algorithm for SS-problems

Algorithm parameterized by:

\[ \Rightarrow \text{an SS-problem } (S, S_C, f) \]
\[ \Rightarrow \text{a pheromone strategy } \Phi \in \{ \text{Vertex, Edge} \} \]

At each cycle, each ant builds a subset

Probability of adding object \( o_i \in \text{Candidates} \) to subset \( S_k \)

\[
p(o_i) = \frac{[\tau_{\text{factor}}(o_i, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_i, S_k)]^\beta}{\sum_{o_j \in \text{Candidates}} [\tau_{\text{factor}}(o_j, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_j, S_k)]^\beta}
\]
Generic ACO algorithm for SS-problems

Algorithm parameterized by:
\[ \leadsto \text{an SS-problem } (S, S_C, f) \]
\[ \leadsto \text{a pheromone strategy } \Phi \in \{ \text{Vertex, Edge} \} \]

At each cycle, each ant builds a subset

Probability of adding object \( o_i \in Candidats \) to subset \( S_k \)

\[
p(o_i) = \frac{[\tau_{\text{factor}}(o_i, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_i, S_k)]^\beta}{\sum_{o_j \in Candidats} [\tau_{\text{factor}}(o_j, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_j, S_k)]^\beta}
\]
### Generic ACO algorithm for SS-problems

**Algorithm parameterized by:**

- \( \rightsquigarrow \) an SS-problem \((S, S_C, f)\)
- \( \rightsquigarrow \) a pheromone strategy \( \Phi \in \{\text{Vertex, Edge}\} \)

**At each cycle, each ant builds a subset**

Probability of adding object \( o_i \in \text{Candidates} \) to subset \( S_k \)

\[
p(o_i) = \frac{[\tau_{\text{factor}}(o_i, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_i, S_k)]^\beta}{\sum_{o_j \in \text{Candidates}} [\tau_{\text{factor}}(o_j, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_j, S_k)]^\beta}
\]

\( \tau_{\text{factor}}(o_i, S_k) \rightsquigarrow \) depends on \( \Phi \)

- If \( \Phi = \text{Vertex} \):
  \[
  \tau_{\text{factor}}(o_i, S_k) = \tau(o_i)
  \]

- If \( \Phi = \text{Edge} \):
  \[
  \tau_{\text{factor}}(o_i, S_k) = \sum_{o_j \in S_k} \tau(o_j, o_i)
  \]
Generic ACO algorithm for SS-problems

Algorithm parameterized by:

\[ \leadsto \text{an SS-problem } (S, S_C, f) \]

\[ \leadsto \text{a pheromone strategy } \Phi \in \{\text{Vertex, Edge}\} \]

At each cycle, each ant builds a subset

Probability of adding object \( o_i \in \text{Candidates} \) to subset \( S_k \)

\[
p(o_i) = \frac{[\tau_{\text{factor}}(o_i, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_i, S_k)]^\beta}{\sum_{o_j \in \text{Candidates}} [\tau_{\text{factor}}(o_j, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_j, S_k)]^\beta}
\]

\( \eta_{\text{factor}}(o_i, S_k) \leadsto \text{depends on the SS-problem} \)
Generic ACO algorithm for SS-problems

Algorithm parameterized by:

\[ \leadsto \text{an SS-problem } (S, S_C, f) \]
\[ \leadsto \text{a pheromone strategy } \Phi \in \{ \text{Vertex}, \text{Edge} \} \]

At each cycle, each ant builds a subset

Probability of adding object \( o_i \in \text{Candidates} \) to subset \( S_k \)

\[
p(o_i) = \frac{[\tau_{\text{factor}}(o_i, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_i, S_k)]^\beta}{\sum_{o_j \in \text{Candidates}} [\tau_{\text{factor}}(o_j, S_k)]^\alpha \cdot [\eta_{\text{factor}}(o_j, S_k)]^\beta}
\]

At the end of each cycle, reward best subsets.

The components of \( S_k \) to be rewarded depend on \( \Phi \)

If \( \Phi = \text{Vertex} \):
  pheromone laying on vertices

If \( \Phi = \text{Edge} \):
  pheromone laying on clique edges