

# Development of an automatically configurable ant colony optimization framework. State of art.

Aldar Saranov

March 25, 2017

# Contents

<b>1</b>	<b>Introduction (1 page)</b>	<b>3</b>
<b>2</b>	<b>Combinatorial Optimization Problems and Constructive Heuristics (17 pages)</b>	<b>3</b>
<b>3</b>	<b>The Concepts of Ant Colony Optimization</b>	<b>6</b>
3.1	Choice of pheromone trails and heuristic information . . . . .	7
3.2	Solution component choice . . . . .	7
3.3	Construction extensions . . . . .	8
3.4	Global pheromone update . . . . .	9
3.5	Pheromone update schedule . . . . .	10
3.6	Initialization of pheromones . . . . .	10
3.7	Pheromone reinitialization . . . . .	10
3.8	Local pheromone update . . . . .	10
3.9	Pheromone limits . . . . .	10
3.10	Local search . . . . .	10
3.11	ACO algorithms as instantiations of the ACO Metaheuristic . .	11
<b>4</b>	<b>Applications of ACO to other problem types</b>	<b>11</b>
4.1	Continuous Optimization Problems . . . . .	11
4.2	Multi-objective problems . . . . .	11
4.3	Dynamic problems . . . . .	11
4.4	Stochastic problems . . . . .	11
<b>5</b>	<b>ACO in combination with other methods</b>	<b>11</b>
5.1	ACO and tree search methods . . . . .	11
5.2	ACO and exact methods . . . . .	11
5.3	ACO and surrogate models . . . . .	11
5.4	Parameter adaptation . . . . .	11
<b>6</b>	<b>Existing ACO framework (5 pages)</b>	<b>11</b>
6.1	Finding a better ACO configuration for the TSP . . . . .	11
6.2	Finding a better ACO configuration for the QAP . . . . .	11
<b>7</b>	<b>IRACE automatic configuration (3 pages)</b>	<b>11</b>
<b>8</b>	<b>Conclusions</b>	<b>11</b>

## Abstract

Some animal species show an extreme degree of social organization. Such species (e.g. ants) have pheromone production and detection body parts and therefore seize an ability to communicate between each other in indirect way. This concept has inspired the development of algorithms which are based on social behavior of population called ant colony optimization algorithms (ACO). These algorithms allow to solve NP-hard problems in a very efficient manner. Since these algorithms are considered metaheuristic the development of a ACO framework is the next step of formalizing of this area is to provide tools for resolving general optimization problems. This article gives the brief overview of the current ACO research area state, existing framework description and some tools which can be used for the framework automatic configuration.

## 1 Introduction (1 page)

Section descriptions. Pheromones. Constructive heuristics. Solution components. Problem models.

## 2 Combinatorial Optimization Problems and Constructive Heuristics (17 pages)

Combinatorial optimization problems (COP) are a whole class of mathematical optimization problems. These problems can be described by grouping, ordering, assigning or any other operations over the set of discrete objects. In practice one may need to resolve COP which have a large number of extra constraints for the solutions which are considered admissible. Many of these problems which are being thoroughly researched at the moment belong to NP-complete discrete optimization problems. NP-complete problem informally means that we cannot decompose a large instance of such problems into a smaller one. Algorithms solving such problems take time larger than polynomial (e.g. exponential).

### Definition

Optimization Problem is a tuple  $(\Phi, \omega, f)$ , where

- $\Phi$  is a search space consisting of all possible assignments of discrete variables  $x_i$ , with  $i = 1, \dots, n$
- $\omega$  is a set of constraints for the decision variables
- $f : \Phi \rightarrow R$  is an objective function which has to be optimized

The problem describes the abstract subclass of tasks (e.g. find the minimum spanning tree of some graph) while the instance of a problem describes a certain practical problem (e.g. find the minimum spanning tree of a given graph  $G$ ). The objective function in this case is the sum of the selected edges. One of the most frequently encountered problems is traveling salesman problem (TSP). Given a graph  $G = (N, E)$  with  $n = |N|$  nodes, where  $E$  - is a set of edges fully connecting the nodes and distances  $d_{ij}, \forall (i, j) \in E$  one should find a Hamiltonian path of minimal length (in terms of sum of the weighted edges). The solution path can be represented as  $\pi = (\pi_1, \dots, \pi_n)^t$  of all  $n$  nodes, where  $\pi_i$  is the node index at position  $i$ . The optimal value of the objective function is

$$\min_{\pi \in \Phi} d_{\pi_i \pi_{i+1}} + \sum_{i=1}^{n-1} d_{\pi_i \pi_{i+1}} \quad (1)$$

Thus  $\pi$  forms a permutation space and every permutation of  $\pi$  gives a admissible (but not necessarily optimal) solution. Plus it is obvious that the absolute position in the permutation sequence does not affect the value of the objective function but the relative one.

In addition to TSP the quadratic assignment problem (QAP) was deeply researched. In QAP there is a set of  $n$  locations and a set of  $n$  facilities which are connected with  $n \times n$  flows. The objective function is represented as the sum of paired production of distances between  $i$  and  $j$  locations and specified flows between  $\pi_i$  and  $\pi_j$  assigned flows. An instance of the problem is given  $n \times n$  matrices  $d_{ij}$  and  $f_{ij}$  with  $i, j = 1, \dots, n$ . A solution of QAP is assignment of the facilities to the locations represented by permutation  $\pi$  where  $\pi_i$  depicts assignment of the corresponding facility to the location  $i$ .

$$\min_{\pi \in \Phi} \sum_{i=1}^n \sum_{j=1}^n d_{ij} \times f_{\pi_i \pi_j} \quad (2)$$

Solution components are normally defined in the terms of COP. Solution

components  $C = c_1, c_2, \dots$  is a set, subset of which corresponds to one solution of the given problem (if also fulfills the constraints). Solutions that fulfill all the constraints are also called *feasible solutions*. In case of TSP solution components are the edges  $(i, j)$  of the given graph. In case of QAP solution components are all the possible assignments of every location  $i$  to every facility  $j$ . In order to provide the feasible solutions the algorithm must either to operate completely in the feasible candidate solution space or to bias towards the feasible ones with final constraint checking.

Since solving of such problems by using provably optimal solutions is unreasonable one can apply heuristic algorithms which more or less provide solutions with relatively good fitness consuming reasonable quantity of resources (time/power, memory etc.).

An essential way in such cases is using of constructive heuristics. Constructive heuristics starts with an empty or partially built solution and is being completed by iterated extension until finished. Each of the iterations adds one or several solution components to the solution. For example greedy constructive heuristics algorithm adds best-ranked component and therefore provides high level of exploiting.

In application to the TSP problem *nearest neighbor* heuristic is used. The algorithm starts from random node  $i$  with initial  $p_i = \langle \rangle$ , at each step it selects the solution component with the minimal distance  $d_{ij}$  and adds the corresponding  $\pi_1 = i$  and  $\pi_2 = j$  components to the solution.

In application to QAP one tends to place the facilities at the locations that are more "central". The algorithm computes  $f = (f_1, \dots, f_n)^t$  where  $f_i = \sum_{j=1}^n f_{ij}$  and  $d = (d_1, \dots, d_n)^t$  where  $d_k = \sum_{l=1}^n d_{kl}$ . Later on the algorithm assigns the facilities with the largest  $f_i$  to the locations with smallest  $d_k$ .

Generally heuristic values are assigned constants, however in extensions one can use heuristics which is a function of the generated partial solution as input. This is called *adaptive* heuristics and normally it consumes larger computer resources although leads to better quality of the solutions.

### 3 The Concepts of Ant Colony Optimization

ACO algorithms are a subclass of construction heuristics algorithms. The meta-heuristic features of these algorithms.

- ACO algorithm is a population-based algorithm. Solutions are being generated at each iteration.
- Solutions are being generated according to probability-based mechanism which is biased by pheromone assignment to the solution components.
- The quality of the generated solutions affect the pheromones are updated during the run of the algorithms.

#### Algorithm

```
procedure ACO-Metaheuristic
  repeat
    for each ant do
      repeat
        ExtendPartialSolutionProbabilistically()
      until solution is complete
      for each ant  $\in$  SelectAntsForLocalSearch() do
        ApplyLocalSearch(ant)
      EvaporatePheromones()
      DepositPheromones()
    until termination criteria met
end
```

Several ants generate the solutions by iterative approach. After this an optional local solution search is applied. After those pheromone evaporation and deposition is done. Evaporation helps to reduce the convergence-prone behavior of the algorithm. Deposition is the part where the solutions affect the pheromone values in order to bias the future solutions.

### 3.1 Choice of pheromone trails and heuristic information

Generally there are two mechanisms of biasing the solution production - pheromones and heuristic values.

Hereby we introduce the following components:

$C$  - Solution components.

$\tau_c \in T$  - pheromones of choosing.

$\tau'_c \in T'$  - pheromones of considering order.

$\pi$  - candidate solution.

$\eta_c \in H$  - heuristic information (constant in time).

Higher values of  $\tau_c$  stand for higher probability of that the component  $c$  will be added to the solution. Additional problem-specific pheromones as  $\tau'_c$  are used for auxiliary purposes (e.g. desirability of considering of one facility after another in QAP). Heuristic information  $H$  is similar to the pheromone trails however it is not updated during the algorithm execution ( $\forall c \in C, \exists \eta_c \in H$ ). Those are either constant values or values which depend on the current partially constructed solution.

### 3.2 Solution component choice

Solution construction phase as says the name yields a new solution set. Each ant starts with an empty solution  $s$ . Each ant produces may produce one solution at one run. At each step one solution component is added. The probability of  $c_j$  to be added at certain step can be calculated by different techniques (i.e.  $Pr(c_j|T, H, s)$ ). Frequently used rule is defined as follows:

$$Pr(c_j) = \frac{t_j^\alpha \times \eta_j^\beta}{\sum_{c_k \in N_i} t_k^\alpha \times \eta_k^\beta} \forall c_j \in N_i \quad (3)$$

$\alpha$  and  $\beta$  are the parameters which determine the impact of the pheromone trails and heuristic information on the final probability. Another alternative has been proposed by Maniezzo [82,83] which combines the pheromone trails and heuristic information in a linear way.

$$Pr(c_j) = \frac{\alpha \times \tau_j + (1 - \alpha) \times \eta_j}{\sum_{c_k \in N_i} \alpha \times \tau_k + (1 - \alpha) \times \eta_k} \forall c_j \in N_i \quad (4)$$

Since it does not use exponentiation operations this algorithm is preferable for performance-targeted frameworks. However this algorithm may cause undesired biases if the range of the values are not taken into account. The third alternative is invented by Dorigo and Gambardella [34] with Ant Colony System (ACS) algorithm. This algorithm is also called pseudo-random proportional rule. A random uniform value  $q$  is generated at range  $[0; 1)$  and if  $q > q_0$  where  $q_0$  is a predefined parameter then probability is being calculated according to the formula (3). Otherwise the solution component is picked as:

$$c_j = \operatorname{argmax}_{c_k \in N_i} t_k^\alpha \times \eta_k^\beta \quad (5)$$

Apparently larger  $q_0$  gives more greedy choice.

### 3.3 Construction extensions

**Lookahead** conception was introduced. It says that at each decision step several solution components should be considered at once in order to get the next solution component. Generally it is worth to be implemented when the cost of making a local prediction based on the current partial solution state is much lower than the cost of the real execution of the move sequence.

**Candidate list** restricts the solution component set to a smaller set to be considered. The solution components in this list have to be the most promising at the current step. Usually this approach yields a significant gain depending on the initial set-up (i.e. if this list is precalculated once before the run). Nonetheless it can also depend on the current partial solution. For TSP it is represented as nearest neighbor list for each of the cities.

**Hash table** of pheromone trails. It allows to efficiently save memory when the updated pheromone trails are a sparse set in comparison to the set of all solution components. Search and updating of the elements of the hash-table is expected to be done within linear time.

**Heuristic precomputation** of the values  $t_j^\alpha \times \eta_j^\beta$  for each of the solution components which are used in (5).

Extensions:

- Lookahead - pick several components at once[94]
- Candidate list - restriction of component choice at each step[33,34]



- Iterated greedy (partial deconstruction)[110]
- With external memory[1]
- Iterated ants[129]
- Cunning ants[128]
- Enhanced ACO[47]

### 3.4 Global pheromone update

Evaporation:

$$\tau_{new} = evaporation(\tau_{old}, \rho, S^{eva})$$

$\rho$  - evaporation rate

$S^{eva}$  - chosen solutions for evaporation

Deposition:

$w_k$  - weight of solution  $s_k$ .

$F(S_k)$  - non-decreasing solution quality scaling function.

Update selection:

1. Ant system (update all)
2. Single update selections:
  - (a) iteration-based update
  - (b) global-based update
  - (c) restart-based update

Update extensions:

1. Max-Min Ant System [122]
2. Rank-based Ant System [19]
3. Best-Worst Ant System [21]
4. Elitist Ant System [30, 36, 38]

### **3.5 Pheromone update schedule**

Exploration vs exploitation.

### **3.6 Initialization of pheromones**

### **3.7 Pheromone reinitialization**

### **3.8 Local pheromone update**

Parallel vs sequential. ACS [34]

### **3.9 Pheromone limits**

MMAS and ACS examples.

### **3.10 Local search**

Neighborhood operator. Best-improving and first-improving.

### 3.11 ACO algorithms as instantiations of the ACO Meta-heuristic

## 4 Applications of ACO to other problem types

### 4.1 Continuous Optimization Problems

### 4.2 Multi-objective problems

### 4.3 Dynamic problems

### 4.4 Stochastic problems

## 5 ACO in combination with other methods

### 5.1 ACO and tree search methods

### 5.2 ACO and exact methods

### 5.3 ACO and surrogate models

### 5.4 Parameter adaptation

## 6 Existing ACO framework (5 pages)

### 6.1 Finding a better ACO configuration for the TSP

### 6.2 Finding a better ACO configuration for the QAP

## 7 IRACE automatic configuration (3 pages)

## 8 Conclusions

## References