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

Aldar Saranov

March 29, 2017

# Contents

**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 (11 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 <u>search space</u> consisting of all possible assignments of discrete variables $x_i$, with $i = 1, ..., n$

- $\omega$ is a <u>set of constraints</u> for the decision variables

- $f : \Phi \to R$ is an <u>objective function</u> 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 $\pi = (\pi_1, ..., \pi_n)^t$ of all n nodes, where $pi_i$ is the node index at position i. The optimal value of the objective function is

$$\min_{\pi \in \Phi} d_{\pi_i \pi_{i+1}} + \sum_{i+1}^{n-1} d_{\pi_i \pi_{i+1}} \tag{1}$$

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 $i, j = 1, ..., n$. 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 $i$.

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

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

components $C = c_1, c_2, ...$ is a set, subset of which corresponds to one solution of the given problem (if also fulfills the constrains). 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 $pi = <>$, 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 $f = (f_1, ..., f_n)^t$ where $f_i = \sum_{j=1}^{n} f_{ij}$ and $d = (d_1, ..., d_n)^t$ 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:

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

$\alpha$ 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.

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

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:

$$c_j = \underset{c_k \in N_i}{\mathrm{argmax}}\, t_k^{\alpha} \times \eta_k^{\beta} \qquad (5)$$

Apparently larger $q_0$ gives more greedy choice.

## 3.3   Construction extensions

**Lookahead** conception was introduced. Is 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 is 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). The gain is based on the fact that all these exponentially-computed values will be shared by the ants at each iteration.

The following extensions are based on the starting from a partially constructed solution with partial destroying of a certain good solution and reconstructing it and thus anticipating to obtain even a better solution.

TO-ELABORATE!!! **External memory**. **Iterated ants**. **Cunning ants**.

## 3.4 Global pheromone update

As it was told the key moment of the algorithm is the pheromone trail biasing. It is composed of two parts - pheromone evaporation and pheromone deposition. Pheromone evaporation decreases the values in order to reduce the impact of the previously deposited solutions. The general form formula is as follows"

$$\tau_{new} = evaporation(\tau_{old}, \rho, S^{eva}) \tag{6}$$

where:

- $\tau_{new}, \tau_{old}$ - new and old pheromone trail values

- $\rho \in (0, 1)$ - evaporation rate

- $S^{eva}$ - selected solution set for evaporation

Classic linear reduction:

$$\tau_j = (1 - \rho) \times \tau_j \ \ \forall \tau_j \in T | c_j \in S^{eva} \tag{7}$$

Hence $\rho = 1$ stands for the pheromone trails are being reset completely. $\rho = 0$ stands for complete missing of evaporation. Other values cause geometrically reducing sequence of the pheromone trail. Often $S^{eva}$ is implemented as the whole set of trails instead of distinctive selection. The generic intention of the evaporation is to slow down the convergence of the run as it opposes the selection of previously generated solution components.

In contrast pheromone deposition increases trail values of the selected solution components. This solution components may belong to several solutions at once. General deposition formula is described as:

$$\tau_j = \tau_j + \sum_{s_k \in S^{upd}} w_k \times F(s_k) \tag{8}$$

- $S^{upd} \subseteq S^{eva}$ - the set of solutions selected for the deposition

- $F$ - non-decreasing function with respect to the solution quality

Following update selection techniques can be used:

1. **Ant system** - selects all solution from the last iteration

2. Single update selections:

    (a) **iteration-based** update - selects the best solution from the last iteration

    (b) **global-based** update - selects the best solution recorded since the start of the run. Provides fast convergence but may lead to stagnation.

    (c) **restart-based update - selects the best solution since last pheromone reset. Prevents stagnation.**

In minimization case typically one assigns adds to a trail a value inversely proportional to output of the objective function.

$$w_k \times F(s_k) = 1/f(s_k) \tag{9}$$

For the mentioned update techniques several variants are possible:

1. **Ant System** - i.e. without extensions.

2. **Max-Min Ant System** - deposits a constant value of pheromone to the components instead of a value defined by the function of quality. The amount of pheromones per component is bounded $t_i \in [t_{max}; t_{min}]$. The pheromones are deposited either by iteration-best or global-best solution. Also update schedule switches between ib, gb and rb depending on the branching factor.

$$\tau'_i = \rho\tau_i + \sum_{\forall ants} \delta t_i^k \tag{10}$$

where

$$\delta t_i^k = \begin{cases} \frac{1}{L^k(t)} & \text{if } i\text{-th component is used by ant } k \text{ at the iteration} \\ 0 & \text{otherwise} \end{cases}$$

$L^k(t)$ - *is the length of k-th ant tour*

3. **Rank-based Ant System** uses the notion of rank (based on the length) for the trail value. $w_k \times F(s_k) = \frac{max(0, w-r)}{f(s_k)}$ where: $w = |S^{upd}|$, r-solution rank in the current iteration

4. **Elitist Ant System** - all solutions from the current iteration deposit pheromones as well as the global-best solution deposits an amount $w_{gb} \times F(s_{gb}) = Q \times \frac{m_{elite}}{f(s_{gb})}$

5. **Best-Worst Ant System** denotes that the global-best solution deposits the pheromones but also evaporation is applied to the components from the worst solution of the current iteration (which also do not present in global-best one)

Pheromone update schedule mechanism allows to dynamically switch between different solution selections. For example algorithm typically starts from ib and then coverts to gb or rb.

## 3.5 Initialization and reinitialization of pheromones

The solution selection algorithm plays critical role in determining of the ACO algorithm behavior. However it is also important how one initializes and reinitializes the pheromones. Typically for ACS and BWAS very small initial values are assigned in the beginning and large ones for MMAS. Small values stand for exploitative bias whereas large stand for exploration one. Pheromone reinitialization is often applied in TSP and QAP since otherwise the run may converge rapidly. MMAS uses a notion of branching factor in a way such that when it exceeds certain value, all pheromone trails are reset to initial state. BWAS resets whenever the distance between global-best and global-worst solution for a certain iteration plummets down lower than a predefined value.

## 3.6 Local pheromone update

For sake of making the behavior more exploratory one can apply local pheromone update according to the formula:

$$\tau_j = (1 - \epsilon) \times \tau_j + \epsilon \times \tau_0 \quad , \forall \tau_j | c_j \tag{11}$$

$\epsilon$ - update strength. $\tau_0$ - initial pheromone trail.

Therefore already explored components become less attractive. Important part is whether the algorithm will work sequentially or in parallel.

## 3.7   Pheromone limits

As it was said MMAS is based on restricting the pheromone values in certain range. It is to prevent stagnation (i.e. situation when certain components cannot be selected at all). Parameter $p_{dec}$ is the probability that an ant has chosen exactly the sequence of solution components that correspond to the best solution.

$$\tau_{min} = \frac{\tau_{max} \times (1 - \sqrt[n]{p_{dec}}}{n' \times \sqrt[n]{p_{dec}}} \qquad (12)$$

where n' - is an estimation of the number of solution components available to each ant at each construction step (often corresponds to $\frac{n}{2}$).

## 3.8   Local search

Local search allows to significantly increase the obtained solutions quality for specific problems. It is based on small iterative solution changes obtained by applying a neighborhood operator (which is problem-specific).

- **best-improvement** scans all the neighborhood and chooses the best solution.

- **first-improvement** takes the first found improving solution in the neighborhood.

# 4   Applications of ACO to other problem types (1 page)

ACO algorithm can be applied to other problem types either directly or by approximated way.

- **Continuous Optimization Problems**. Simplest way is to approximate the problem to its discrete analogue, however it cannot be applied to certain problems. Some adaptations are carried out by Socha and Dorigo [113]. Another model was presented by Liao[72, 74].

12

- **Multi-objective problems** are the problems that have several defined objective functions either have a deterministic preference model or demand the Pareto set as the solution. Several algorithms for resolving such problems were reviewed in the paper of Lpez-Ibez and Sttzle [77]. There are two common ways: maximizing of each of objectives (COMPETants) or finding non-dominated solutions (BicreterionAnt, MACS, mACO-3). TO-ELABORATE???

- **Dynamic problems** are the problems that have some information revealed during the execution. These problems are resolved in a strongly different manner - ants act asynchronously and no global pheromones are updated, instead specific update mechanisms are held.

- **Stochastic problems** deal with information that is not deterministic. Such problems as probabilistic TSP where for each city there is a given probability that it is required to visit.

# 5 Existing ACO framework (5 pages)

ACOTSP unified framework was developed by IRIDIA group. It was developed with tendency towards the purpose and component generality. It separates the general structures of ACO metaheuristics from the problem-specific domain. All standard parameters can be specified ($\alpha, \beta, \rho, m, \epsilon, etc.$) plus one can set the specific parameters ($t_{max}, t_{min}, res_{it}$ - number of iterations since last found rb-solution, $res_{bf}$ - branching factor, $res_{dist}$ - distance between global-best and iteration-worst ants, $q_0$).