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

Aldar Saranov

May 29, 2017

Contents

1 Introduction **3**

2 Combinatorial Optimization Problems and Constructive Heuristics **4**

3 Ant Colony Optimization **6**
3.1 Choice of pheromone trails and heuristic information 8
3.2 Solution component choice 8
3.3 Construction extensions 9
3.4 Global pheromone update 12
3.5 Initialization and reinitialization of pheromones 15
3.6 Local pheromone update 15
3.7 Pheromone trail limits 16
3.8 Local search 16

4 Applications of ACO to other problem types **17**

5 Existing optimization frameworks **18**
5.1 ACOTSPQAP 18
5.2 Other frameworks 19

6 Automatic configuration in IRACE **20**

7 Research state **23**
7.1 Finding a better ACO configuration for the TSP 24
7.2 Finding a better ACO configuration for the QAP 24

8 Conclusions **24**

Abstract

Some 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 an indirect way. This concept has inspired the development of algorithms, which are based on the social behavior of the ant colonies called ant colony optimization algorithms. These algorithms allow to solve NP-hard problems in a very efficient manner. These algorithms are considered to be metaheuristics. The development of an ACO framework is the next step of formalizing this area. Such a framework can then be used as a tool to help resolving various optimization problems. This report gives a brief overview of the current state of the ACO research area, existing framework description and some tools which can be used for the automatic configuration of the framework.

1 Introduction

Metaheuristic algorithms are widely used for solving such difficult problems as NP-hard problems, since the exact solution of such problems often takes unreasonable computational time in practice. One family of such algorithms is the Ant Colony Optimization algorithms. These algorithms comprise a various set of options, that affect the final performance of the algorithm. Thus, it is a logical decision to develop a unified framework which summarizes ACO algorithmic components for later assembly of ACO algorithms. After developing such a framework, the next step is to create a tuning tool that will observe the configurations space of ACO algorithms and select the best ones, in order to improve the quality of obtained solutions. In this report, we display the current research state of the Ant Colony Optimization frameworks that solve optimization problems and their configuration. To do that we split the report into several sections which will represent different practical points of the ACO application.
In the section 2, we introduce the basic notions of Combinatorial Optimization Problems and will introduce the Traveling Salesman Problem and the Quadratic Assignment Problem as examples of NP-hard problems to solve. In the section 3, we provide a description of Ant Colony Optimization in terms of constructive heuristics with introduction into the main components and choices of this metaheuristic. In the section 4, we describe the application

of ACO to different problem classes such as continuous optimization problems, multi-objective problems, dynamic problems and stochastic problems. The existing optimization frameworks are described in section 5, where we describe the ACOTSPQAP framework and some additional frameworks with other purposes. Automatic configuration process, which is applied "above" the developed ACO algorithms, is explained in the section 6. In section 7, we list the configurations that were rendered by the configuration process on a test set for both TSP and QAP problems. Section 8 concludes the report and proposes further possible contributions into the area.

2 Combinatorial Optimization Problems and Constructive Heuristics

Combinatorial optimization problems (COPs) are a class of mathematical optimization problems. These problems can be described as a grouping, ordering, assignment or any other operations over a set of discrete objects. In practice, one may need to resolve COPs, which have a large number of extra constraints for the solutions to consider them as admissible. Many of these problems which are still being thoroughly researched at the moment, belong to NP-hard discrete optimization problems. The best performing algorithms known today to solve such problems have a worst-case run-time larger than polynomial (e.g. exponential).

Definition

An Optimization Problem is a tuple [2] (Φ, ω, f), where

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

The problem describes the abstract task (e.g. find the minimum spanning tree of some graph), while the instance of a problem describes a specific practical problem (e.g. find the minimum spanning tree of a given graph G). The objective function is the sum of the weights of the selected edges.
One of the most studied problems is the traveling salesman problem (TSP)

[1]. Given a graph $G = (N, E)$ with $n = |N|$ nodes and E being the set of edges that fully connects the nodes and distances $d_{ij}, \forall (i, j) \in E$, one should find a Hamiltonian path of minimal length (in terms of the sum of the weighted edges). The solution path can be represented as permutation $\pi = (\pi_1, ..., \pi_n)^t$ of all n nodes, where π_i is the node index at position i. The objective 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 π forms a permutation space and every permutation of π gives a admissible (but not necessarily optimal) solution. It is obvious that the absolute position in the permutation sequence does not affect the value of the objective function but only the relative position.

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

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

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

Since solving of such NP-hard problems by trying to find provably optimal solutions is unreasonable, one can apply heuristic algorithms, which

more or less provide solutions with relatively good quality consuming reasonable quantity of resources (time/power, memory etc.).

An important class of such heuristic algorithms are constructive heuristics. Constructive heuristics start with an empty or partially built solution, which is then being completed by iterative extension until a full solution is completed. Each of the iterations adds one or several solution components to the solution. For example, greedy constructive heuristics add the best-ranked components by their heuristic values.

For the TSP, the *nearest neighbor* heuristic can be used. The nearest neighbor heuristic starts from a random node π_i with initial random $\pi =< \pi_1 >$. At each step it selects the node with the minimal distance d_{ij} to the current node and adds the corresponding next node $\pi_2 = j$ components to the solution. The index of the next node at each step can be computed as following.

$$nextNode = \underset{j \in N, j \notin \pi}{\operatorname{argmin}} (\pi_{ij}) \tag{3}$$

For the QAP, one tends to place the facilities which exchange a lot of flow at 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 $d_k = \sum_{l=1}^{n} d_{kl}$. Then the algorithm assigns the facility with the largest f_i to the location with smallest d_k and iterates through these steps.

Generally, heuristic values are assigned as constants at the start of the solution construction. However, in extensions one can use heuristic values, which are a function of the generated partial solution. These are called *adaptive* heuristics and normally they consume larger computer resources although they often lead to a better quality of the generated solutions.

3 Ant Colony Optimization

ACO algorithms are a class of constructive heuristics. Meta-heuristic is a top-level heuristic, which is used to improve the performance of an underlying,

basic heuristic. ACO is such a metaheuristic that can be used to improve the performance of a construction heuristic. Features of ACO algorithms are the following.

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

Listing 1: General ACO pseudo-code

```
procedure ACO–Metaheuristic
repeat
        foreach ant do
                repeat
                        Extend partial solution
                          ↪ probabilistically
                until solution is complete

        foreach ant in SelectAntsForLocalSearch() do
                ApplyLocalSearch(ant)

        EvaporatePheromones
        DepositPheromones
until termination criteria met
end
```

Pheromones are numeric values associated to the solution components that are meant to bias the solution construction in order to improve the quality of the generated solutions. Several ants generate the solutions by an iterative approach (see listing 1). After this, an optional local search is applied. Next, pheromone evaporation and deposit is done. Evaporation helps to reduce the convergence-prone behavior of the algorithm. Deposit 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 construction - pheromones and heuristic values.
Hereby we introduce the following components:
C - set of solution components (a combination of which can constitute a solution).
$\tau_c \in T$ - pheromones trail values for solution component bias.
$\tau'_c \in T'$ - pheromones trail values for particular purposes (as choosing the next facility to consider in QAP).
π - candidate solution.
$\eta_c \in H$ - heuristic information (constant in time).

Higher values of τ_c stand for a higher probability that the component c will be added to a solution. Additionally, problem-specific pheromones such as τ'_c can be used for auxiliary purposes (e.g. desirability of considering of one facility after another in the QAP). Heuristic information H is similar to the pheromone trails in terms of that they both bias the solution component choice. However, it is defined in problem-specific way and is not updated during the algorithm execution ($\forall c \in C, \exists \eta_c \in H$). Heuristic information is either static values or values which are defined by a function of the current partially constructed solution.

3.2 Solution component choice

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

$$Pr(c_j) = \frac{t_j^\alpha \times \eta_j^\beta}{\sum\limits_{c_k \in N_i} t_k^\alpha \times \eta_k^\beta} \forall c_j \in N_i \tag{4}$$

α and β are parameters that determine the impact of the pheromone trails and heuristic information on the final probability. Another alternative

has been proposed by Maniezzo [3], which combines the pheromone trails and heuristic information in a linear way.

$$Pr(c_j) = \frac{\alpha \times \tau_j + (1-\alpha) \times \eta_j}{\sum\limits_{c_k \in N_i} \alpha \times \tau_k + (1-\alpha) \times \tau_k} \forall c_j \in N_i \tag{5}$$

Since it does not use exponentiation operations such choice rule 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 [4] in Ant Colony System (ACS) algorithm. The rule of solution component choice in ACS is also called pseudo-random proportional rule. A random uniform value q is generated in range $[0; 1)$ and if $q > q_0$, where q_0 is a predefined parameter, then the probability of choosing c_j is calculated according to formula (4). Otherwise, the solution component is picked as:

$$c_j = \underset{c_k \in N_i}{\operatorname{argmax}} t_k^\alpha \times \eta_k^\beta \tag{6}$$

Apparently, larger q_0 gives more greedy choice.

3.3 Construction extensions

Lookahead concept 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 [5]. This says that in constructing a candidate solution, ants use a transition rule that incorporates complete information on past decisions (*the trail*) and local information (*visibility*) on the immediate decision that is to be made [6]. The look-ahead mechanism allows the incorporation of information of the anticipated decisions that are beyond the immediate choice horizon. 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.

A 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 ones at the current step [7]. Usually, this approach yields a significant gain of computation time, 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 the TSP it is represented as the 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 in 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 [8].

Heuristic precomputation of the values $t_j^\alpha \times \eta_j^\beta$ for each of the solution components which are used in formula (6). The reduction of computation time is based on the fact that all these values will be shared by the ants at each iteration.

External memory extension is based on starting the solution construction 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 [9]. This iterated greedy extension was inspired by genetic algorithms and described in [26]. It uses reinforcement procedures of the elite solutions with deferred reintroducing of solutions segments in following iterations (see listing 2). The ACO iteration is composed of two stages. First is meant to initialize the external memory. The second is the solution construction itself based on a partial solution. **Iterated ants** also uses the partially constructed solutions. Based on the following additional notions. Destruct() removes some solution components from a complete solution [10]. Constuct() constructs a complete solution from initially partial solution. An acceptance-criterion chooses one of two complete solutions in order to continue the construction with it. Concrete implementations of these strategies are defined in problem-specific way. The algorithm of the extension is showed in listing 3. **Cunning ants** algorithm tackles to the solution generation by iterated producing of new ant population. The algorithm has a pheromone database and an ant population of fixed size. For every existing ant, a new one is produced which borrows some solution parts from its parent [11]. Then in each such ant pair a winner is selected and all winners continue their work in the next iteration. After each iteration all winners jointly update the pheromone database and stop if the termination criteria is met. Similarly the solution component inheritance process is problem-specific.

Listing 2: External memory iteration pseudo-code

```
procedure ACO-external-memory
initialize the external memory
repeat
        set m ants in the graph
        ants construct a solution using neighborhood
           ↪ graph and the pheromone matrix
        select k-best solutions and cut randomly
           ↪ positioned and sized segments
        store the segments into the external memory
until the external memory is full
done = false
while not(done)
        ants select their segments according to
           ↪ tournament selection
        ants finish the solution construction
        update the pheromone matrix
        update the memory
end
end
```

Listing 3: General iterated ants pseudo-code

```
procedure iterated-ants
        s0 = initial-solution()
        s = local-search(s0)
        repeat
                sp = destruct(s)
                s' = construct(sp)
                s' = local-search(s') // optional
                s = acceptances-criterion(s, s')
        until termination criterion met
end
```

3.4 Global pheromone update

As it was said before, the key idea of the ACO algorithm is the pheromone trail biasing. It is composed of two parts - pheromone evaporation and pheromone deposit. Pheromone evaporation decreases the values in order to reduce the impact of the previously deposited solutions. The general form formula is as follows.

$$\tau_{new} = evaporation(\tau_{old}, \rho, S^{eva}) \tag{7}$$

where:

- τ_{new}, τ_{old} - new and old pheromone trail values
- $\rho \in (0, 1)$ - evaporation rate
- S^{eva} - selected solution set for evaporation

The classic linear reduction is as follows:

$$\tau_j = (1 - \rho) \times \tau_j \quad \forall \tau_j \in T | c_j \in S^{eva} \tag{8}$$

Hence $\rho = 1$ stands for the pheromone trails are being reset completely to zero whereas $\rho = 0$ means that pheromone trail remains exactly the same. Other values cause a geometrically decreasing sequence of the pheromone trails with a number of iterations. Usually, all the solution components are being selected for the evaporation, however, some modifications perform distinctive selection of the components. The generic intention of the evaporation is to slow down the convergence of solution components, which can be selected with some reasonable probability, to a limited subset of all solution components.

In contrast, the pheromone deposit increases the pheromone trail values of selected solution components. The solution components may belong to several solutions at once. The general deposit formula is described as:

$$\tau_j = \tau_j + \sum_{s_k \in S^{upd}} w_k \times F(s_k) \tag{9}$$

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

- F - non-decreasing function with respect to the solution quality
- w_k - multiplication weight of the k-th solution.

The 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 a state called stagnation. Stagnation is a state, where the pheromone trails are defined in such way, that some solution components are selected regularly from one iteration to another, so that virtually no other solution components can be selected.
   (c) **restart-based update - selects the best solution since last pheromone reset.**

In the minimization case, typically one adds a value inversely proportional to a value of the solution quality function.

$$w_k \times F(s_k) = 1/f(s_k) \tag{10}$$

For the mentioned update techniques several variants are possible:

1. **Ant System** - i.e. without extensions. Every pheromone trail is evaporated.
2. **Ant Colony System** - uses formulas 4 or 6 for solution construction. It also uses local pheromone update according to formula 14. Thus, it makes the components, that have been already chosen, less attractive for the rest of the ants. After the solution construction is finished, the ants deposit the pheromone trails according to formula 11

   $$\tau_j = (1-\rho) \times \tau_j + \rho \times \Delta\tau_j^*, \forall j \in S^* \tag{11}$$

   where S^* is the best solution so far, and $\Delta t_j^* = 1/z^*$, z^* is the cost of S^*.

3. **Max-Min Ant System**. The pheromone values are updated by evaporating all pheromone trails according to 13 with consequent deposit of $\Delta\tau = 1/z$ to the solution components in global-best or iteration-best or reset-based solution, where z is the cost of this solution. The amount of pheromones per component is bounded $\tau_i \in [\tau_{max}; \tau_{min}]$. The update schedule switches between ib, gb and rb depending on the branching factor [12].

$$\tau_i' = \rho\tau_i + \sum_{\forall ants} \delta t_i^k \tag{12}$$

where

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

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

$$\tau_j = (1-\rho) \times \tau_j, \forall j \in N \tag{13}$$

4. **Rank-based Ant System** uses the notion of rank by depositing [13] to the pheromone trail the following 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

5. **Elitist Ant System** - all solutions from the current iteration deposit pheromones as well as the global-best solution s_{gb} deposits an amount $w_{gb} \times F(s_{gb}) = Q \times \frac{m_{elite}}{f(s_{gb})}$ [14], where m_{elite} is the multiplicative factor that increases the weight of s_{gb} solution, Q is the same multiplication factor from the standard AS.

6. **Best-Worst Ant System** denotes that pheromone trail values are deposited to the solution components that constitute the global-best solution but also evaporation is applied to the components from the worst solution of the current iteration (which are in the global-best solution) [15]

The pheromone update schedule allows to dynamically switch between different solution selections. For example, an ACO algorithm may start from ib and then converge to gb or rb. If one wants to start the problem solution process with exploratory behavior and end up with exploiting one, then it

is possible to start solution process with ib-update and then switch to gb or ib updates at the later solution iterations. The update schedule has a major influence on the ACO algorithm performance, since it determines the balance between convergence and exploration traits of the algorithm in a global scale.

3.5 Initialization and reinitialization of pheromones

The solution selection algorithm plays a critical role in determining 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, because in the first case better solution components will rapidly gain pheromone values advantage over the rest ones, and in the second case the high values will oppose such fast convergence. Pheromone reinitialization in often applied in ACO algorithms since otherwise the run may converge rapidly. MMAS uses a notion of branching factor in such a way that when it falls below a certain value, all pheromone trails are reset to the initial pheromone trail value. 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 [16] local pheromone update in ACS according to the formula:

$$\tau_j = (1 - \epsilon) \times \tau_j + \epsilon \times \tau_0 \quad , \forall \tau_j | c_j \tag{14}$$

ϵ - update strength. τ_0 - initial pheromone trail.

This local evaporation is applied to solution components that are used by some ant and future ants will choose these components with smaller probability. Therefore, already explored components become less attractive. An important algorithmic detail is whether the algorithm will work sequentially or in parallel, since in parallel case ants will have to modify the pheromone trails simultaneously, whereas in sequential they have no such a problem. However it does not behave very efficiently with other ACO algorithms but ACS, because it relies on a strong gb-update and a lack of pheromone evaporation.

3.7 Pheromone trail limits

As it was said before, MMAS is based on restricting the pheromone values in a certain range. Parameter p_{dec} is the probability that an ant chooses exactly the solution components that reproduce best solution found so far.

$$\tau_{min} = \frac{\tau_{max} \times (1 - \sqrt[n]{p_{dec}})}{n' \times \sqrt[n]{p_{dec}}} \tag{15}$$

where n' - is an estimation of the number of solution components available to each ant at each construction step (often corresponds to $\frac{n}{2}$). The p_{dec} allows to estimate the reasonable values for τ_{min} according to formula 15. This formula relies on two facts. First is that the best solutions are found shortly before search stagnation occurs. The second fact is that the relative difference between upper and lower pheromone trail limits has more influence on the solution construction than the relative differences of the heuristic information

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 so-called neighborhood operator.

- **best-improvement** scans all the neighborhood and chooses the best solution.
- **first-improvement** takes the first found improving solution in the neighborhood.

In the general ACO algorithm this stage is optional. If it is fast and effective it may be used to all solutions at each iteration. In other cases applying it in iteration-best ants may be the optimal strategy. A neighborhood operator is always defined in a problem-specific way. For example, in the TSP problem a frequent way to find a solution in solution neighborhood is 2-opt. This algorithm considers the solution route partition into three sequential sub-routes with reversing of the middle one, in case if it will reduce the length of the tour. 2-opt for the QAP problem swaps two facilities from two certain locations if the total distance-flow cost will be lower.

4 Applications of ACO to other problem types

Initially ACO algorithms were meant to solve single-objective combinatorial optimization problems. However, ACO algorithms can be applied to other problem types.

COPs. The simplest way to solve COPs by means of ACO algorithms 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 [18]. Another model was presented by Liao[19].

Multi-objective problems are problems that have several objective functions, which sometimes have a deterministic preference model or demand the Pareto set as the solution. In practice, such problems normally do not have a preference of one Pareto-optimal solutions over others. For example, one such problem, which has been solved, is bi-objective Traveling Salesman Problem (bTSP). This problem is identical to the original TSP, except that it has two length markups for each edge, and each of two objective functions corresponds to tour length is estimated by one of these length markups.

So far, various multi-objective ACO algorithms have been developed. Several ACO algorithms for resolving such problems were reviewed in the paper of López-Ibáñez and Thomas Stützle [20]. Different MOACO algorithms implement different design choices and, therefore, have different algorithmic components. This was the reason to implement a unified flexible framework. The developed MOACO framework encompasses many MOACO algorithms. This framework allows either to instantiate MOACO algorithms based on the existing ones, or to combine algorithmic components from different MOACO algorithms to produce new ones.

For solution component choice the possible options are *single/multiple pheromone trails and single/multiple heuristic information*. Since multiple pheromone matrices need to be aggregated into a single one, it is necessary to choose the aggregation rule. That can be *weighted sum/weighted product/random/next-weight aggregation*. As for selecting a solution for pheromone information update one can choose *non-dominated solutions/ best-of-objective/best-of-objective-per-weight*.

Many MOACO algorithms use the idea of multiple colonies with different application of those. A colony is a group within the total ant population. Each such group is associated to its own pheromone information. Thus, in bi-objective case each colony has two pheromone matrices. The following algorithmic components define cooperation behavior of colonies. *MultiColony-*

Weights defines how the colonies share the weights that are defined at the aggregation step. *MultiColonyUpdate* defines the notion of solution archive where the solutions from colonies are submitted. After this they are redistributed back to the colonies for pheromone information update. Thus, the colonies exchange the solutions between each other.

As it was said, these algorithmic components are combined in order to obtain new MOACO algorithms, and therefore, it is possible to perform an automatic configuration of these algorithms.

Dynamic problems are problems that have some information (such as data or objectives) revealed in time. Such problems are usually encountered in network routing. In problems like dynamic TSP and dynamic vehicle routing problem cities appear and disappear during the solution, as well as distances between them. These problems are resolved in a strongly different manner - ants act asynchronously and no global pheromones are updated, instead specific update mechanisms are held. An overview of ACO algorithms in application to dynamic optimization problems was done by Alba Leguizamón in [21].

Stochastic problems deal with information that is not deterministic. Such problems mean that instead of deterministic values of the problem data and objectives, we have their probability distribution. In practice such problems are caused by uncertainty or noise affecting the problem information. The first stochastic problem to deal with was probabilistic TSP (pTSP), where for each city there is a given probability that it is required to visit. The first algorithm was proposed by Bianchi in [22] with further development by Guntsch and Branke in [23].

5 Existing optimization frameworks

5.1 ACOTSPQAP

ACOTSPQAP is a unified framework that was developed by researchers of the IRIDIA group, it is publicly available at http://iridia.ulb.ac.be/aco-tsp-qap/. It was developed with the aim to provide generality of algorithmic components of ACO algorithms. 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). All these parameters are to be tuned by an external configuration software.

Algorithmic frameworks can be classified as one of these two (although their distinction is ambiguous in particular cases):

- Top-down - develop a fixed template-based algorithm. It is based on strictly structured algorithm which allows some parametric configuration of some of its details with minor behavior modifications.
- Bottom-up - algorithm is build of flexible components with some rule restrictions. Often involves application of genetic programming and evolution ideas.

ACOTSPQAP can be classified as the first one, since its stages (solution generation, local search, pheromone trail update) are concretely defined and its parameters are strongly determined.

5.2 Other frameworks

The **MOACO** framework was briefly described in the section 4. Besides there are some other optimization frameworks which have different features.

Another remarkable framework is **SATenstein**. It is based on solving satisfiability problem (SAT) of a boolean formula. SAT problem is considered important because other NP-complete problems can be encoded as instances of SAT problem. Various Stochastic Local Search (SLS) algorithms have been developed for solving SAT problems. The SATenstein can be configured to instantiate a broad range of existing high-performance SLS-based solvers [25]. The general form of SLS algorithm is made of five blocks. First block performs search diversification. Blocks from the second to the fourth perform a variable flip with instantiating G^2WSAT-derived, WalkSAT-based or dynamic local search algorithm respectively. The fifth block performs updates to the promising variable list, tabu attributes, clause penalties or dynamically adapted algorithm parameters. Thus, a high-performing SLS algorithm can be assembled of such algorithmic blocks in SATenstein. A generic algorithmic configuration tool was applied to SATenstein for finding best algorithm instantiations for predefined sets, which is described in the mentioned paper.

Hybrid Stochastic Local Search (SLS) algorithms, as in the previously mentioned frameworks, are composed of the separate algorithmic components. These algorithms rely on the manipulating a single solution called *current solution*. The neighborhood of the current solution will be explored during the algorithm run [24]. The first phase of the algorithm is the local search for the initial solution. Then the algorithm iterations start. One iteration consists from *perturbation, local search* and *acceptanceCriterion*. Perturbation is a transformation of the input solution. Perturbation may be implemented as one simple move in neighborhood space, but it may also be composed of k applications of such moves. *Local search* can range from a simple iterative improvement over short runs of an SA algorithm to a full-fledged iterated local search. *Acceptance criterion* returns either the initial solution or the obtained one, according to some condition. The simplest criterion is *improveAccept* accepts the solution with better quality. Other acceptance criteria allow worse solutions to be accepted in order to increase the exploration of the search space. For instance *probAccept* accepts a worsening solution with probability $p \in [0; 1]$. These iterations are repeated until a defined termination criterion is fulfilled. As a result, such metaheuristic algorithms are among the most effective non-exact algorithms for tackling hard combinatorial optimization problems.

6 Automatic configuration in IRACE

Automatic configuration is a process that optimizes the performance of a certain algorithm as a goal function based on input parameters of the algorithm. The general parameter types are:

- **categorical** parameters - represent discrete values without any implicit order or distance measure between its possible values. Define the choice of constructive procedure, choice of branching strategies (i. e. algorithmic blocs) and so on.
- **ordinal** parameters - are also assigned to finite discrete values, but they have implicit value order (such as *low, middle, high*). Define lower bounds, neighborhoods.
- **numerical** parameters - define integer or real values/constants such as weighting factors, population sizes.

Some of the parameters maybe subordinate to other ones. This means that their values only make sense if the parameters, that they subordinate to, are assigned to certain values. This can be expressed as a scalar value constraint (e.g $a < b$) or as a dependency on concrete values of some categorical or ordinal parameter.

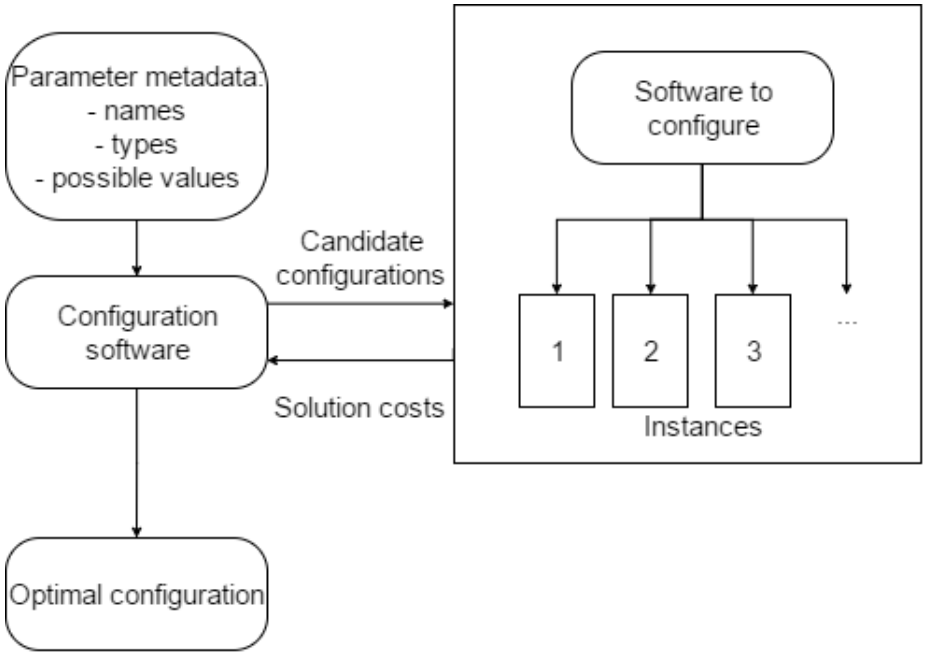

Figure 1: Automatic configuration scheme

Figure 1 shows the configuration software and software to configure. Configuration script has parameter metadata at its disposal. Based on them, the configuration software runs the software to configure with candidate configurations according to some algorithm. After the configuration process finishes, the configuration software renders the best configurations obtained. In the most general software case there are two measures of the performance - solution quality (to maximize) and computation time (to minimize). This general approach is called an offline tuning. It introduces a learning stage on training instances before learning on the real-world instance set.

A widely used configuration algorithmic family is racing algorithms. The simplified algorithm is shown in 4 and an illustration is in 2.

Listing 4: General racing pseudo-code

```
procedure racing
start with an initial candidate set Theta
repeat iterations I
        process an instance stream
        evaluate the candidates sequentially
        remove inferior candidates
until winner is selected or exit condition fulfilled
end
```

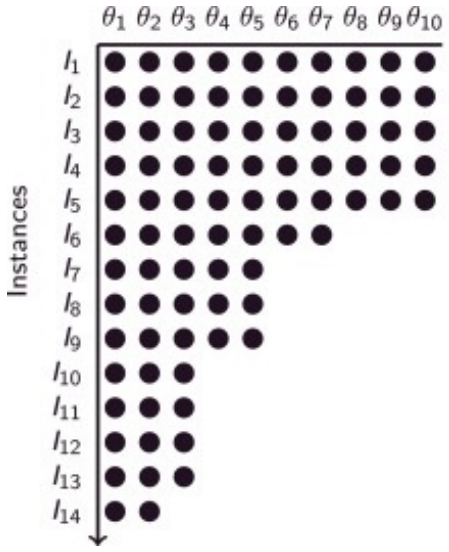

Figure 2: I-RACE execution illustration

I-RACE (iterated race) configuration implementation was developed and described in [17]. It was implemented in R with taking into account the parallel programming techniques and an initial candidate set-up. The feature of the IRACE is based on iterated generation of new configurations and removing of solutions with lower quality for further evaluating on the problem instances.

In order to tune the configurations that are sampled, IRACE algorithms uses independent sampling distributions for each of the parameters. For numeric values those are normal distributions and discretely-defined distributions for the rest. The configuration biasing procedure is based on modifying

the sampling distributions.

Iterated racing is an automatic configuration implementation that consists of three steps:

- Sampling new configurations according to a particular distribution.
- Selecting the best configurations from the newly sampled ones by means of racing.
- Updating the sampling distribution in order to bias the sampling towards the best configurations.

After new configurations are sampled comes the selection stage. At the configuration selection stage, IRACE runs each of the configurations on a single problem instance from the predefined instance set. After each racing iteration the worst configurations are discarded. Then the rest of the configurations update parameter sampling distributions. The racing stops when the number of the survived configurations becomes small enough (defined by termination criteria).

Some IRACE extensions can be applied. **Initial configurations** can be set before the run of the IRACE. **Soft-restart** is used for preventing premature convergence. Such convergence may suppresses configuration diversity and, therefore some good configurations may be lost. This restart is triggered if the value of an ad hoc function of configuration distance is lower than a certain margin. Then reinitialization is applied to the elite configurations.

7 Research state

The mentioned ACOTSPQAP framework accompanied by IRACE R script have already produced results for the TSP and the QAP problems. The results are described in the following two subsections. General conclusion during this research is that solutions obtained by the optimal configurations are better than those obtained by the default configurations. The comparison was carried out by measuring the deviation from the optimal solution for each instance.

In cases where MMAS is used, user can specify the frequency of using restart-best solutions, instead of iteration-best.

7.1 Finding a better ACO configuration for the TSP

The optimal configuration is mentioned in the table 1, where it was computed for different algorithms. Various instances of different sizes were generated for the configuration process. This ACO algorithms include using of heuristic information, candidate lists and multiple types of local search.

algo	m	α	β	ρ	q_0	ϵ	cl	$nnls$	$ph-limits$	$slen$	$restart$	res_{it}
ACS	28	3.07	5.09	0.32	0.53	0.21	22	9	-	-	branch-factor ($res_{bf} = 1.74$)	212
MMAS	40	0.94	4.11	0.72	0.14	-	18	12	yes	191	branch-factor ($res_{bf} = 1.91$)	367

Table 1: Optimal configuration for the TSP problem.

With local search as 3-opt + dlb-bits.

7.2 Finding a better ACO configuration for the QAP

ACO algorithms do not include heuristic information, nor candidate lists, since these techniques did not prove to be useful for the QAP. Problem instances were implemented in two forms - as RR or RS. In RR the distance matrix is computed as paired euclidean distances of points distributed on a square of length 300 in uniform way. The flow matrix is generated as a matrix of uniform random values within certain range. The RS distance matrix is generated in the same way, but the flow matrix adheres to real-world flow values.

100 instances were considered where half of them was used for training and the another half for testing. The best found configuration are shown in the table 2.

	algo	m	α	ρ	q_0	$dlb-bits$	$ph-limits$	$slen$	$restart$	res_{it}
RR	MMAS	6	0.324	0.29	0.062	yes	no	153	distance ($res_{bf} = 0.051$)	22
RS	MMAS	4	0.164	0.342	0.284	yes	no	170	branch-factor ($res_{bf} = 1.822$)	40

Table 2: Optimal configuration for the TSP problem.

With local search as 2-best-opt.

8 Conclusions

Vast research work has been already conducted in terms of ACO algorithms. Many types of ACO algorithms have already been developed and tested. In

addition, various technical details can be added in order to improve resolution performance or the development experience.

Various ways of further research in this area can be proposed. The possible contribution is to implement algorithms for some other NP-hard problems in terms of ACO metaheuristic. Those can be such problems as Vehicle Routing Problem, Subset Sum Problem or Knapsack Problem.

From a technical point of view we can propose implementing a new framework using object-oriented paradigm, which will ensure high modularity and modification-proneness. Another option is to implement the ACO algorithm stages as parallel algorithms (stages like solution generation or local search).

References

[1] Manuel López-Ibáñez and Thomas Stützle (2015) Ant Colony Optimization: A Component-Wise Overview. IRIDIA - Technical Report Series.

[2] Papadimitriou CH, Steiglitz K (1982) Combinatorial Optimization Algorithms and Complexity. Prentice Hall, Englewood Cliffs, NJ

[3] Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS Journal on Computing 11(4):358-369

[4] Dorigo M, Gambardella LM (1997) Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1):53-66

[5] Michel R, Middendorf M (1998) An island model based Ant System with lookahead for the shortest supersequence problem. In: Eiben AE, Bck T, Schoenauer M, Schwefel HP (eds) Parallel Problem Solving from Nature, PPSN V, Lecture Notes in Computer Science, vol 1498, Springer, Heidelberg, Germany, pp 692-701

[6] C. Gagne, M. Gravel, and W. Price (2001) A look-ahead addition to the ant colony optimization metaheuristic and its application to an industrial scheduling problem: in Proceedings of the 4th Metaheuristics International Conference (MIC 01), J. Sousa, Ed., pp. 79-84, Porto, Portugal

[7] Dorigo M, Di Caro GA (1999) The Ant Colony Optimization meta-heuristic. In: Corne D, Dorigo M, Glover F (eds) New Ideas in Optimization, McGraw Hill, London, UK, pp 11-32

[8] Alba E, Chicano F (2007) ACOhg: dealing with huge graphs. In: Thierens D, et al (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2007, ACM Press, New York, NY, pp 10-17

[9] Ruiz R, Thomas Stützle (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3):2033-2049

[10] Wiesemann W, Stützle T (2006) Iterated ants: An experimental study for the quadratic assignment problem. In: Dorigo M, et al (eds) Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Lecture Notes in Computer Science, vol 4150, Springer, Heidelberg, Germany, pp 179-190

[11] Tsutsui S (2007) Ant colony optimization with cunning ants. Transactions of the Japanese Society for Artificial Intelligence 22:29-36

[12] Thomas Stützle and Hoos HH Sttzle T, Hoos HH (2000) MAXMIN Ant System. Future Generation Computer Systems 16(8):889-914

[13] Bullnheimer B, Hartl R, Strauss C (1999) A new rank-based version of the Ant System: A computational study. Central European Journal for Operations Research and Economics 7(1):25-38

[14] Dorigo M (1992) Optimization, learning and natural algorithms. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, in Italian

[15] Cordón O, de Viana IF, Herrera F, Moreno L (2000) A new ACO model integrating evolutionary computation concepts: The best-worst ant system. In: Dorigo M, et al (eds) Abstract proceedings of ANTS 2000 From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, IRIDIA, Université Libre de Bruxelles, Belgium, pp 22-29

[16] Dorigo M, Gambardella LM (1997) Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1):53-66

[17] Manuel López-Ibáñez and Jérémie Dubois-Lacoste and Leslie Pérez Cáceres and Thomas Stützle and Mauro Birattari (2013) Iterated Racing for Automatic Algorithm Configuration. IRIDIA - Technical Report Series, vol 3, Operations Research Perspectives, pp 43-58.

[18] Socha K, Dorigo M (2008) Ant colony optimization for continuous domains. European Journal of Operational Research 185(3):1155-1173

[19] Liao T, Montes de Oca MA, Aydin D, Stützle T, Dorigo M (2011) An incremental ant colony algorithm with local search for continuous optimization. In: Krasnogor N, Lanzi PL (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011, ACM Press, New York, NY, pp 125-132

[20] López-Ibáñez M, Stützle T (2012) The automatic design of multiobjective ant colony optimization algorithms. IEEE Transactions on Evolutionary Computation 16(6):861-875

[21] Leguizamón G, Alba E (2013) Ant colony based algorithms for dynamic optimization problems. In: Alba E, Nakib A, Siarry P (eds) Metaheuristics for Dynamic Optimization, Studies in Computational Intelligence, vol 433, Springer, Berlin/Heidelberg, pp 189-210

[22] Bianchi L, Gambardella LM, Dorigo M (2002) An ant colony optimization approach to the probabilistic traveling salesman problem. In: Merelo JJ, et al (eds) Parallel Problem Solving from Nature, PPSN VII, Springer, Heidelberg, Germany, Lecture Notes in Computer Science, vol 2439, pp 883-892

[23] Guntsch M, Branke J (2003) New ideas for applying ant colony optimization to the probabilistic tsp. In: Cagnoni S, et al (eds) Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2003, Springer, Heidelberg, Germany, Lecture Notes in Computer Science, vol 2611, pp 165-175

[24] Marie-Eleonore Marmion, Franco Mascia, Manuel López-Ibáñez, and Thomas Stützle (2013) IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium, pp 144-158

[25] R. KhudaBukhsh, L. Xu, H. H. Hoos, K. Leyton-Brown (2009) SATenstein: Automatically Building Local Search SAT Solvers From Components, Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)

[26] Acan A (2004) An external memory implementation in ant colony optimization. In: Dorigo M, et al (eds) Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004, Lecture Notes in Computer Science, vol 3172, Springer, Heidelberg, Germany, pp 73-84