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

Aldar Saranov

May 19, 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 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 (1 page)

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 the Combinatorial Optimization Problems. These problems are described by means of two particular NP-hard problems - the Traveling Salesman Problem and the Quadratic Assignment Problem. For them, we define the problem formulations as well as the objective functions.
In the section 3 we provide a description of the Ant Colony Optimization. One describes the ACO in terms of constructive heuristics. Besides, we introduce the notion of pheromone trails and explain how they affect the solution construction. In addition to the general ACO resolution algorithm, we conduct an overview of various algorithm extensions which are meant to increase its performance. Two types of pheromone update are presented - global and local. Different Ant Systems are briefly described.
In the section 4 we describe the application of ACO for different problem classes. Those are Continuous Optimization Problems, Multi-objective Problems, Dynamic Problems and Stochastic Problems.
The existing optimization frameworks are described in the section 5 where

we mainly focus on single-objective optimization problem and ad hoc ACOT-SPQAP framework. However, several other frameworks with different purposes and ideas are mentioned.
Automatic configuration process which is applied ”above” the developed ACO algorithms is explained in the section 6. The top-level outline of this process is shown as well as the I-RACE implementation which is already developed and tested.
In section 7 we list the configurations that were rendered by the configuration process on a test set for both TSP and QAP problems.
The section 8 concludes the report and proposes further possible contributions into the area.

2 Combinatorial Optimization Problems and Constructive Heuristics (11 pages)

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 have been thoroughly researched at the moment, belong to NP-hard discrete optimization problems. NP-hard problem means that we cannot decompose a large instance of such problems into a smaller one. The best performing algorithms known today solve such problems have a worst-case run-time larger than polynomial (e.g. exponential).

Definition

An Optimization Problem is a tuple (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
- $f : \Phi \rightarrow R$ 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). 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 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 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 the 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 flows.

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

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.

$$nextNode = \operatorname*{argmin}_{j \in N, j \notin \pi} (d_{ij}) \tag{3}$$

For the QAP, one tends to place the facilities which exchange a lot of flow at the locations that are more "central". The algorithm computes $f = (f_1, ..., f_n)^t$ where $f_i = \sum\limits_{j=1}^{n} f_{ij}$ and $d = (d_1, ..., d_n)^t$ where $d_k = \sum\limits_{l=1}^{n} d_{kl}$. 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. This is called *adaptive* heuristics and normally it consumes larger computer resources although leads to better quality of the 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 if 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 characteristics of 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 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 production - 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 of choosing.
$\tau'_c \in T'$ - pheromones of considering order.
π - candidate solution.
$\eta_c \in H$ - heuristic information (constant in time).

Higher values of τ_c stand for higher probability of that the component c will be added to the solution. Additional problem-specific pheromones as τ'_c are 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, 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:

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

α and β are 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.

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

Since it does not use exponentiation operations 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 (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 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.

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 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 (6). 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 (iterated greedy 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.

External memory extension is inspired by genetic algorithms and described in [2]. It uses reinforcement procedures of the elite solutions with deferred reintroducing of solutions segments in following iterations (see 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.

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
```

Iterated ants. Based on the following additional notions. Destruct() removes some solution components from a complete solution. Constuct() constructs a complete solution from initially partial solution. Acceptancecriterion 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 3

Listing 3: General ACO 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
```

Cunning ants.

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. 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.

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 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

Classic linear reduction:

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

Hence $\rho = 1$ stands for the pheromone trails are being reset completely. $\rho = 0$ stands for complete missing of evaporation. Other values cause geometrically reducing sequence of the pheromone trail. Usually all the solution components are being selected for the evaporation, however some modifications perform distinctive selection of the components based on a fixed algorithm. 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, the pheromone deposit increases pheromone trail values of the 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

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.

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

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 formula 6 for solution construction and only those pheromone trails are evaporated that are used for deposit either.

3. **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.

   $$\tau_i' = \rho\tau_i + \sum_{\forall ants} \delta t_i^k \tag{11}$$

   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*

4. **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

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

6. **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 in ACS according to the formula:

$$\tau_j = (1-\epsilon) \times \tau_j + \epsilon \times \tau_0 \quad , \forall \tau_j | c_j \tag{12}$$

ϵ - update strength. τ_0 - initial pheromone trail.

Therefore already explored components become less attractive. Important part is whether the algorithm will work sequentially or in parallel. However it does not behave very efficiently with other ACO algorithms but ACS.

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.

$$\tau_{min} = \frac{\tau_{max} \times (1 - \sqrt[n]{p_{dec}})}{n' \times \sqrt[n]{p_{dec}}} \tag{13}$$

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].
- **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. In practice such problems normally do not have a preference model over the pareto-optimal solutions. Several algorithms for resolving such problems were reviewed

in the paper of López-Ibáñez and Thomas Stützle [77]. In that work a MOACO framework has been described which operated 10 multi-objective parameters and 12 ACO parameters. There are two common ways: maximizing of each of objectives (COMPETants) or finding non-dominated solutions (BicreterionAnt, MACS, mACO-3). The feature of MOACO framework is that it stores multiple pheromone trail data for each of the specified objectives. In the same time the weighted sum of the objective functions' values composes a unified scalar objective function.

- **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 optimization frameworks (5 pages)

5.1 ACOTSPQAP

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). All these parameters are to be tuned by an external configuration software.

5.2 Other frameworks

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

Multi-objective Bat Algorithm (MOBA) is based on the natural traits of bat behavior. Bat emits waves in the space which allow it to find the prey location. The bat speed, the emission frequency and emission amplitude are the parameters which the bat is able to steer. When a bat finds a potential prey it increases the frequency and decreases the amplitude and therefore is able to locate it more precisely. In optimization problem a similar action is being done by a bat when it ”flies” within the parameter search space. The general purpose of MOBA is to find the Pareto Front of the parameters.

Another remarkable framework is SATenstein. It can solve single-objective optimization problems. It is based on solving satisfiability problem of a boolean formula.

Hybrid Stochastic Local Search (SLS) algorithms are based on decomposition of single-point SLS methods into separate components with generalized metaheuristic structure.

6 Automatic configuration in IRACE (3 pages)

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 - define the choice of constructive procedure, choice of branching strategies (i. e. algorithmic blocs) and so on.
- **ordinal** parameters - define lower bounds, neighborhoods.
- **numerical** parameters - define integer or real values/constants such as weighting factors, population sizes, temperature. They can be optional according to different categorical parameter values.

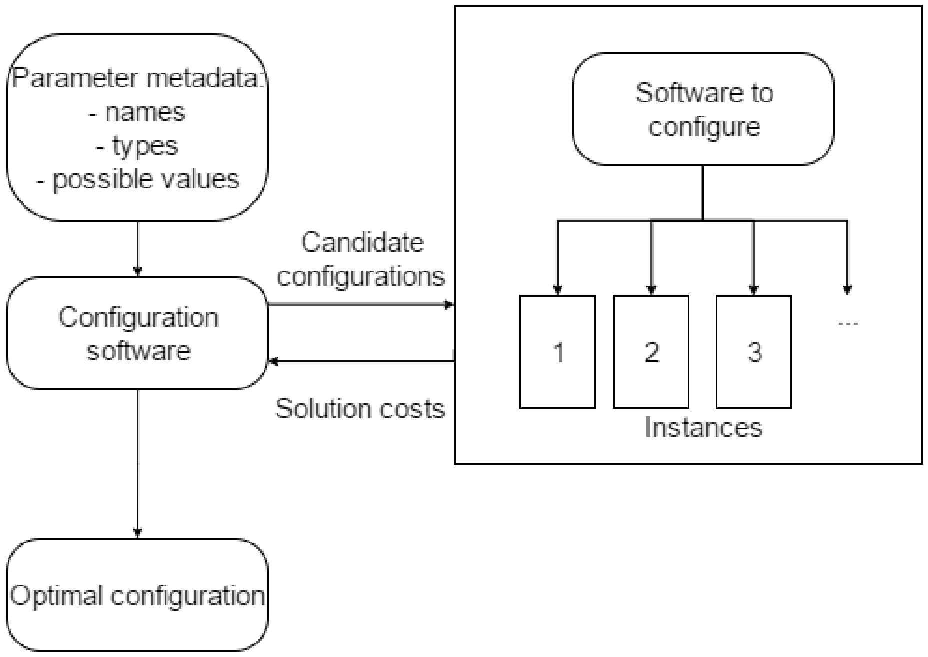


Figure 1: Automatic configuration scheme

Figure 1 shows software composition of the analyzed program and configuration script. Configuration script has parameter metadata in its disposal. Based on them, the configuration software runs the software to configure with candidate configurations one by one according to the higher-mentioned racing algorithm. The result of the run is the solution cost of cumulative runs of the algorithm on the defined problem instances. After the racing process finishes, the configuration software renders the best configuration obtained. In the most general software case there are two measures of the performance - solution quality (to maximize) and computation time (to minimize).

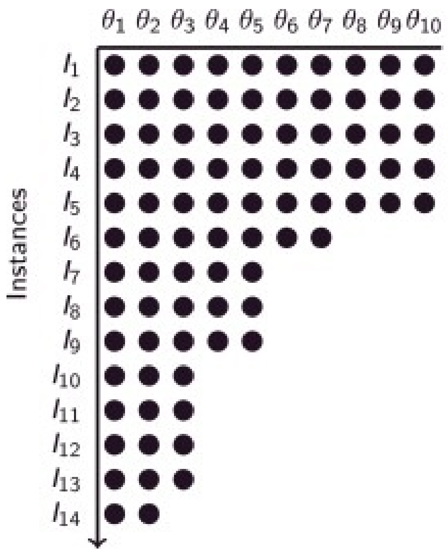
There are two application modes:

- Offline tuning - introduces a learning stage on training instances before learning on the actual set.
- Online tuning - tunes the parameters while solving the actual instance set.

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



θ_1 θ_2 θ_3 θ_4 θ_5 θ_6 θ_7 θ_8 θ_9 θ_{10}
I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8 I_9 I_{10} I_{11} I_{12} I_{13} I_{14}
Instances

Figure 2: I-RACE execution illustration

I-RACE(iterated race) implementation has already been developed and applied for the ACO problem in [1]. It was implemented in R with taking into account the parallel programming techniques and initial candidate set-up. The feature of the IRACE is based on iterated generation of new configurations and removing of solutions with less fitness for further evaluating on the problem instances.

Two conceptual approaches can be remarked (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.

7 Research state

The higher-mentioned ACOTSP 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.

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.

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

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 and their extensions were developed and tested. However there are still many COPs untouched. In addition, various technical details can be added in order to improve resolution performance or the development experience.

Various ways of further area research can be proposed. The possible way is to implement new NP-hard problems resolution algorithms within ACO framework. Such practical problems as Vehicle Routning Problem, Subset Sum Problem or Knapsack Problem.

From 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 Jérémie Dubois-Lacoste and Leslie Pérez Cáceres and Thomas Stützle and Mauro Birattari *Iterated Racing for Automatic Algorithm Configuration* 2013: IRIDIA - Technical Report Series.

[2] Dorigo, Marco and Birattari, Mauro and Blum, Christian and Gambardella, Luca Maria and Mondada, Francesco and Stützle, Thomas *IAn External Memory Implementation in Ant Colony Optimization* 2004: Springer Berlin Heidelberg - Proceedings.