UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

Development of an automatically configurable ant colony optimization framework

Aldar Saranov

Promoter : Prof. Thomas Stützle

Master Thesis in Computer Sciences

Academic year 2016 - 2017+1

*Dedicated to my mother Lena, who
was always sincerely supporting me*

"*If one does not know to which port one is sailing, no wind is favorable.*"

Lucius Annaeus Seneca, 1st century AD

"*The general, unable to control his irritation, will launch his men to the assault like swarming ants, with the result that one-third of his men are slain, while the town still remains untaken. Such are the disastrous effects of a siege.*"

Sun Tzu, "The Art of War", 5th century BC

Acknowledgment

I want to thank my promoter, prof. Thomas Stützle, whose determination and competencies were leading me to the goal of the paper, and my friend, Alain, who made this precious time of my education possible.

Contents

Chapter 1

Introduction 4-5

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.

In the chapter 2, we describe the theoretical foundations of the framework, that is required to develop. However unreasonable selection of a running configuration may lead to producing of low-quality results even after a long runtime execution. A simple decision is to perform certain configuration tuning before the actual executing. This chapter also introduces an efficient configuration software, that allows to obtain high-performing configurations for further framework exploiting.

The development of such framework would also require its application to some particular problem for testing and analysis purposes. Vehicle Routing Problem was chosen as the one. It is a widely researched NP-hard combinatorial optimization problem, therefore, it will allow us to perform benchmark comparison of the public available problem instances and its solutions with the solutions, that we obtain. In the chapter 3, we introduce formulation of Vehicle Routing Problem and also mention several variations thereof.

In the chapter 4, we describe thorough implementation details and decisions for the ACO framework, the way we adapt it specifically to VRP-problems and also show line-by-line description of the parameter space defined for the framework. We also explain there how do we attain high flexibility of the framework in order to adapt it for various problem types besides VRP.

In the chapter 5, one can see the experimental set-up, tuning results and interpretations that are made.

[TODO: Add more?]

[TODO: fix citing]

Chapter 2

Background 20

2.1 Combinatorial Optimization Problems and Constructive Heuristics 5

Combinatorial optimization problems (COPs) are a large 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).

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 formulation describes an 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.

Any combination of solution components is called a candidate solution (not necessarily satisfying all problem constraints). Oppositely, solution is a candidate solution, that satisfies all problem constraints.

Since solving of 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.

[TSP and QAP description here, or not needed???]

2.2 Ant Colony Optimization 10

[rip-off the previous year paper, describe only those components that are used]

ACO algorithms are a subclass 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. One has to remark several necessary features of the ACO algorithms:

- 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 2.1: General ACO pseudo-code

```
procedure ACO-Metaheuristic
repeat
  foreach ant do
    repeat
      extend-partial-solution-probabilistically()
    until solution-is-complete()

  foreach ant in select-ants-for-local-search() do
    apply-local-search(ant)

  evaporate-pheromones()
  deposit-pheromones()
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 2.1). 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.

2.2.1 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.
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.

Similarly heuristic values are numerical attributes of the solution components. However, generally heuristic values are assigned as constants at the start of the solution construction, although 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. 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.

2.2.2 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:

$$Pr(c_j) = \frac{t_j^\alpha \times \eta_j^\beta}{\sum\limits_{c_k \in N_i} t_k^\alpha \times \eta_k^\beta} \forall c_j \in N_i \tag{2.1}$$

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 [?], which combines the pheromone trails and heuristic information in a linear way.

$$Pr(c_j) = \frac{\alpha \times \tau_j + (1-\alpha) \times \eta_j}{\sum\limits_{c_k \in N_i} \alpha \times \tau_k + (1-\alpha) \times \tau_k} \forall c_j \in N_i \tag{2.2}$$

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 [?] 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 (2.1). Otherwise, the solution component is picked as:

$$c_j = \operatorname*{argmax}_{c_k \in N_i} t_k^{\alpha} \times \eta_k^{\beta} \tag{2.3}$$

Apparently, larger q_0 gives more greedy choice.

2.2.3 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 [?]. 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 [?]. 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 [?]. 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.

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 [?].

Heuristic precomputation of the values $t_j^{\alpha} \times \eta_j^{\beta}$ for each of the solution components which are used in formula (2.3). 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 [?]. This iterated greedy extension was inspired by genetic algorithms and described in [?]. It uses reinforcement procedures of the elite solutions with deferred reintroducing of solutions segments in following iterations (see listing 2.2). 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 [**?**]. 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 2.3. **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 [**?**]. 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 2.2: 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 2.3: 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
```

2.2.4 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.

$$\tau_{new} = evaporation(\tau_{old}, \rho, S^{eva}) \tag{2.4}$$

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:

$$\tau_j = (1 - \rho) \times \tau_j \quad \forall \tau_j \in T | c_j \in S^{eva} \tag{2.5}$$

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:

$$\tau_j = \tau_j + \sum_{s_k \in S^{upd}} w_k \times F(s_k) \tag{2.6}$$

- $S^{upd} \subseteq S^{eva}$ - 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.

$$w_k \times F(s_k) = 1/f(s_k) \tag{2.7}$$

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 2.1 or 2.3 for solution construction. It also uses local pheromone update according to formula 2.13. 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 2.8

   $$\tau_j = (1-\rho) \times \tau_j + \rho \times \Delta\tau_j^*, \forall j \in S^* \tag{2.8}$$

   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 2.10 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 value called branching factor [**?**].

$$\tau_i' = \rho\tau_i + \sum_{\forall ants} \delta t_i^k \tag{2.9}$$

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*

$$\tau_j = (1 - \rho) \times \tau_j, \forall j \in N \tag{2.10}$$

A special *pheromone trail smoothing* mechanism is often used hand in hand with MMAS update. It is a tool to oppose the effect of convergence, thus, it is useful for long runs that are excessively exploiting. It is applied in case, if fraction Ω (branching factor) of solution components, whose pheromone trail values surpass the computed threshold τ_t, exceeds predefined fraction Ω'. Higher Ω values correspond to higher grade of convergence. Computing of τ_t is shown in formula 2.11.

$$\tau_t = \tau_{min} + (\tau_{max} - \tau_{min}) \times \lambda \tag{2.11}$$

where τ_{min} *- minimum pheromone trail border,* τ_{max} *- maximum pheromone trail border.* λ *- threshold determining factor in [0;1].*

When applied, the value of each pheromone trail is increased according to formula 2.12, and thus, exploration is boosted.

$$\tau_{ij}^* = \tau_{ij} + \delta \times (\tau_{max} - \tau_{ij}) \tag{2.12}$$

where δ *- is elevation parameter in [0;1].*

4. **Rank-based Ant System** uses the notion of rank by depositing [**?**] 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 $w_{gb} \times F(s_{gb}) = Q \times \frac{m_{elite}}{f(s_{gb})}$ [**?**], 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) [**?**]

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.

2.2.5 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.

2.2.6 Local pheromone update

For sake of making the behavior more exploratory one can apply [?] local pheromone update in ACS according to the formula:

$$\tau_j = (1-\epsilon) \times \tau_j + \epsilon \times \tau_0 \quad , \forall \tau_j | c_j \tag{2.13}$$

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.

2.2.7 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.

$$\tau_{min} = \frac{\tau_{max} \times (1 - \sqrt[n]{p_{dec}})}{n' \times \sqrt[n]{p_{dec}}} \tag{2.14}$$

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 2.14. 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.

2.2.8 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, followed by evaluation of the solutions. Generally there are two neighborhood scanning strategies:

- **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. It is not guaranteed to converge on an optimal solution, however, it may converge on a relatively good local optimum. 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. More thoroughly this is explained in chapter 3.

In many COPs it is possible to optimize computation of objective function value of a newly obtained solution, based on objective function value of an old solution. From feasibility point of view one can apply a local move and check whether this is a solution after. Another option is to define a neighborhood operator in such way, that all solution's neighbors are also feasible solutions.

In many problem-depending cases perturbative search can be embedded into the local search heuristics. The aim of such mechanism is to allow escaping from local search space "peaks" which might be not as good as the others.

One of the most efficient metaheuristics in local search domain is Iterated Local Search (ILS) [?]. Usually it leads to much better solutions, than the ones obtained by repeated random trials. It uses custom components as local search procedure, perturbation procedure and acceptance criterion. The pseudo-code is shown in 2.4.

Listing 2.4: Iterated Local Search pseudo-code

```
procedure IteratedLocalSearch
  $s_0$ = GenerateInitialSoluition()
  $s^*$ = LocalSearch($s_0$)
  repeat
    $s'$ = Perturbation($s^*$, history)
    $s^{*\prime}$ = LocalSearch($s'$)
    $s^*$ = AcceptanceCriterion($s^*$, $s^{*\prime}$, history)
  until TerminationCondition()
end
```

The simplest acceptance criteria is choice of a better solution, however one may want to use a probabilistic approach. It prevents major perturbation moves, that do not effectively contribute to objective goal improving, and thus, attains good trade-off between exploration and intensification. History can be taken into account in this heuristics, so the whole procedure is in fact a "Markovian" process, which implies that further moves completely depend on the current state.

[Illustration of local search. Allowed to copy-paste images???]

[ACOTSPQAP brief description???]

2.3 Iterated F-Race 5

[description similar to the slides of I-RACE description]

Chapter 3

Vehicle Routing Problem 5

[correct formulas below under the VRP definition in the future]

For the VRP, 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.

$$nextNode = \operatorname*{argmin}_{j \in N, j \notin \pi} (\pi_{ij}) \tag{3.1}$$

[basically explain all blue components from the UML]

[maintaining feasibility during the run]

[candidate list determining for VRP]

[preselecting in VRP]

[local search in VRP]

[perturbation double-bridge]

Chapter 4

Implementation 20-25

[framework class-level description, no class diagram due to large size (however no descriptions for every class method. it's not a documentation)]

[mention the back-bone classes that provide generality and maintaining of VRP instance generality]

[framework parameters space]

[mention unit testing???]

Chapter 5

Experimental 20-25

[public samples description, some publicly know plots]

[configuration of i-race]

[obtained result configuration for the framework]

[description of the configuration]

Chapter 6

Conclusion 4-5

6.1 Literature