Bio-inspired Artificial Intelligence

Exercise: Ant Colony Optimization

Paweł Lichocki, Sabine Hauert Date: 24 November 2009

1. Introduction

In this exercise we focus on using ACO in order to find solutions for TSP.

- Ant colony optimization (ACO) is a population-based metaheuristic that can be used to find approximate solutions to difficult optimization problems.
- In Traveling Salesman Problem (TSP) you are given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city exactly once.

In general TSP is a very difficult computational problem and finding exact best solution often requires many years of CPU time. For this reason people has developed different heuristic approaches for that problem - heuristic means that the algorithm usually finds a very good solution, but not necessarily the optimal one. ACO is exactly one of such heuristic approaches.

ACO was inspired by natural systems of real ant colonies. The main principles of the algorithm were presented on the lectures. We remind them in a condense form
(see http://www.scholarpedia.org/article/Ant_colony_optimization for more detailed information):

(a). In a single iteration of ACO m ants simultaneously create m tours (one tour per each ant) of size n (in our case, n is the size of the TSP problem).

(b). There are k iterations, the best solution (the shortest tour) found during the entire run is considered to be the result of the ACO algorithm.

(c). Ants choose their paths in a guided stochastic way, an ith ant on city src moves to city dst using the following probabilistic rule:

$$p_i(src, dst) \begin{cases} \frac{\tau(src,dst)^\alpha \cdot \eta(src,dst)^\beta}{\sum_{s \in J_i(src)} \tau(src,dst)^\alpha \cdot \eta(src,dst)^\beta} & if\ s \in J_i(src) \\ 0 & otherwise \end{cases}$$

Where $J_i(src)$ is the set of components that do not belong yet to the partial solution of ith ant , and α and β are parameters that control the relative importance of the pheromone versus the heuristic information $\eta(src, dst) = \frac{1}{d(src,dst)}$, where $d(src, dst)$ is the length of the edge (src, dst).

(d). There are two rules to update the pheromone: local and global.

- ((i)) local rule: the pheromone level of each edge visited by an ant is decreased by a fraction $(1-\varphi)$ of its current level and increased by a fraction φ of the initial pheromone level τ_0

$$\tau(src, dst) = (1-\varphi) \cdot \tau(src, dst) + \varphi \cdot \tau_0$$

- ((ii)) global rule: when all ants complete their tours, the length L of the shortest tour in the current iteration of ACO is used to update the pheromone levels of only the edges of this shortest path in inverse proportion to the path length.

$$\tau(src, dst) = (1-\rho) \cdot \tau(src, dst) + \rho \cdot L^{-1}$$

Note: the version of the algorithm presented on the lecture assumed that $\rho = \varphi$.

(e). Typical values for the algorithm are as follow:

- ((i)) $\tau_0 = (n \cdot L_{nn})^{-1}$
- ((ii)) $\rho = \varphi = 0.1$
- ((iii)) $\alpha = 1$
- ((iv)) $\beta = 2$
- ((v)) for TSP we typically use $m = n$ ants (where n is the number of cities)
- ((vi)) normally the ACO runs for at least 100 iterations (but this may as well go in thousands), before it converges to a very good solution.

Note: Just for this exercise purposes, we have artificially decreased the initial level of the pheromone. This was necessary in order to obtain the solution in reasonable time and still be able to do all the tasks required. Thus, in our case $30 \leq k \leq 100$ is sufficient.

Another heuristic for solving TSP is the Greedy algorithm (also called $NearestNeighbors$). It is simple and naive, but also very fast, and quite efficient (meaning it produces relatively good results). It works basing on one straightforward principle:

(a). First randomly pick a starting city.

(b). Then, from all the cities that are not yet on the tour pick one that is nearest to the last added city. Repeat step (b) until you create a whole tour.

2. Tools The main tool in this exercise is Matlab. You are provided with the implementation of ACO algorithm:

(a). *Init.m* file contains the initial settings for both ACO and Greedy algorithm.

(b). *Aco.m* file contains the source code for ACO algorithm.

(c). *Greedy.m* file contains the source code for Greedy algorithm.

(d). *GetCities.m* file contains a function for loading a TSP instance from a file.

(e). *Ulysses22.m* file contains an exemplary TSP instance, which is used during this exercise.

Commands in Matlab:

(a). To run the ACO algorithm type
$> [o, p] = Aco()$.
This runs the ACO and stores the initial settings in o ('o' for options) and the solution in p ('p' for problem). See *Init.m* and *Aco.m* for details.

(b). Later on you will be also using a greedy heuristic to solve TSP. The Greedy algorithm is run by typing:
$> [o, p] = Greedy(o, p)$.
Notice that you need to run *Aco* first, before running the Greedy algorithm (you must pass o and p as arguments).

Note: Both *ACO* and *Greedy* will not yield good results at first! You must set some initials options and slightly change the code (you are asked to do that in the next part of the tutorial).

(c). To see the length of the best found tour by ACO type:
$> min(p.acoBestLens)$

(d). To see the length of the best found tour by Greedy type:
$> p.greedyBestLen$

(e). To plot the fitness of the solutions found by ACO in next iterations type:
$> plot([1 : length(p.acoBestLens)], p.acoBestLens)$

(f). To see the problem (cities) on a map type:
$> plot(p.cities(:, 1), p.cities(:, 2), '+')$

3. Finish the implementation Your first task is to choose proper pheromone updating rules and set properly the initial number of ants.

(a). There is some code in the *Aco.m* file that needs to be commented-out, search for lines marked with *TODO*. Those are the pheromone update rules which you might want to comment-out... or not - it is your task to choose proper ones, the goal is to have an algorithm that acts as the ACS.

Q: What is the main concept behind the local pheromone update rule?
a) allow *exploration* of many different solutions
b) make *convergence* to a good solution more probable

Q: What is the concept behind the global pheromone update rule?
a) allow *exploration* of many different solutions
b) make *convergence* to a good solution more probable

4. Play with the parameters

(a). In file *Init.m* set the number of ants (variable *o.m*) to 1, run Aco and note the length of best tour found.
Best fitness when $o.m = 1$ **equals to** _____

Now set the number of ants to a typical value in case of TSP and run Aco once again (HINT: use as many ants as there are cities - variable *o.n*):
Best fitness when $o.m =$ _____ **equals to** _____

Q: Did it help to use more ants?
A: **YES / NO**

Q*: What do you think, is it reasonable (from practical point of view) to use a huge amount of ants, let say 100000?
A: **YES / NO**

Note: leave *o.m* **as you have just set it.**

(b). In *Init.m* file:
1. Set the number of tours (*o.k*) to 20. Run Aco and note the length of the best tour found.
Best fitness when $o.k = 20$ **is** ____.

2. Then set $o.k = 50$, run Aco and once again note the length of the best tour found.
Best fitness when $o.k = 50$ **is** ____.

3. Then set $o.k = 70$, run Aco and once again note the length of the best tour found.
Best fitness when $o.k = 70$ **is** ____.

Q: In order to find better solutions, does it help to construct more tours (using larger value of *o.k*)?
A: Using $o.k = 50$ **helps / does not help** (in comparison to the case when $o.k = 20$).
A: Using $o.k = 70$ **helps / does not help** (in comparison to the case when $o.k = 20$).
A: Using $o.k = 70$ **helps / does not help** (in comparison to the case when $o.k = 50$).

Q*. Observe the fitness plot for $o.k = 50$ and $o.k = 70$. Can you think of any other stop condition for ACO (other that maximal number of tours)? Focus on the *convergence*.
A:

Q*. What would be a very simple and "practical" stop condition? Focus on the *actual running time*.
A:

Note: when you finish, set *o.k* **back to 20!**

(c). Implement and test different ant behaviors by changing *o.alpha* and *o.beta* so that:

((i)) ants take into account only the pheromone trail
o.alpha=_____
o.beta=_____
best fitness=_____

((ii)) ants take into account only distances between cities
o.alpha=_____
o.beta=_____
best fitness=_____

Q: In order to find better solutions, does it help to use both the pheromone and the distance between cities (compare with results obtained in 4.(*b*))?
A: **YES / NO**

Note: when you finish, set *o.alpha* **back to 1 and** *o.beta* **back to 2!**

5. Comparison with other solutions Now, when you know a lot about the ACO itself, you can compare it with another algorithm - in the file *Greedy.m* you will find the greedy heuristic.

(a). Notice that running the greedy algorithm only once seems not fair (in comparison to ACO), since the ACO uses many ants that construct many tours (uses much more computations and time).

Q: Set the variable *o.greedyReps* (in file *Greedy.m*) so the comparison between Greedy and ACO is fair (HINT: construct the same amount of valid tours in Greedy algorithm as in the case for ACO).
A: **o.greedyReps=_____**

Q: Is the greedy algorithm better or worse than ACO? (compare your result with those acquired in point 4.(*b*))
A: **Best fitness of the greedy algorithm is _____**
A: **Ratio between fitness of the greedy algorithm and fitness of ACO is _____**

(b). The optimal value for the Ulysses22.tsp problem is known. Go find the solution on http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html.

Q: What is the length of the optimal tour? (compare it with the results obtained by Greedy and Aco)
A: **Fitness for the optimal algorithm is _____**
A: **Ratio between fitness of the greedy algorithm and optimal one is _____**
A: **Ratio between fitness of ACO and optimal algorithm is _____**

6. Supplementary questions Try answering these three "open" questions. Every one of them links to one of the three main properties of ACO algorithms: using pheromone, parallelism and randomization (not in this order). Do you see which question relates to which property? Why those properties are important?

(a). Q: Is running a single ant $m * k$ times the same as running m ants for k iterations? Think what is the role of local pheromone decrease.
A:

This question is linked with the concept of
using pheromone / parallelism / randomization

(b). Q: Remind the situation when you were changing *o.alpha* and *o.beta*, so the ants would take into account only the information about distance between cities. Assume also that you use just 1 ant. Is the ACO equivalent to the greedy algorithm? Think about the probabilistic rule that defines ants behaviors.
A:

This question is linked with the concept of
using pheromone / parallelism / randomization

(c). Q: ACO algorithms are considered very efficient when dealing with dynamic problems. Imagine you have a TSP in which nodes sometimes disappear and reappear again. Can you explain why ACO is a good choice to solve this problem? Think about the pheromone trail.
A:

This question is linked with the concept of
using pheromone / parallelism / randomization