Ant Colony System

Casnici Davide

November 2022

1 Algorithm implementation

1.1 Initialization

In order to implement the algorithm, I have strictly followed the steps in the given paper (ACS).
First of all, I had to make a change in the code, since as explained by Prof. Gambardella in the lecture the pheromone is an n by n matrix, regardless the candidates lists, as well as the eta matrix.
I then added to the initial variables a vocabulary in order to keep track of the cities still valid for each ant (i.e. J_k in the paper) and the variables to keep track of the relevant statistics.

1.2 'restart variables'

In this function I reinitialize the variables such as the position of each ant, their tours and the set of available cities for each of them, this function has to be called each time we initialize a new iteration.

1.3 'pseudo random proportional rule'

In this function I first sample $q \sim Un[0,1]$. The value of q is used to decide the next city to be visited by the ant according to the ACS state transition rule:

$$s = \begin{cases} argmax_{u \in J_k(r)}[\tau(r,s)] \cdot [\eta(r,s)]^\beta & \text{if } q \leq q_0 \ (exploitation) \\ S & \text{otherwise} \ (biased\, exploration) \end{cases} \tag{1}$$

$$S \sim p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]\cdot[\eta(r,s)]^\beta}{\sum_{u \in J_k(r)}[\tau(r,u)]\cdot[\eta(r,u)]^\beta} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The cities in $J_k(r)$, where r is the current city of the ant, is the set of all the cities not yet visited by the ant (among all the graph). If the current city r still has some city in the candidates list which have not been visited by the ant yet, the $J_r(r)$ set is set of only the candidate cities not yet visited.
The value of q0 has been chosen after some experiment (i.e. averaging 3 minutes

of running on three different seeds), which led to a best value of $(1-13)/n$ (among the suggested values 0.5, 0.98, (1-13)/n).

1.4 'local updating'

While ants construct a solution a local pheromone updating rule is applied calling this function. The local updating is made following the formula in paper, shown below:

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \tag{3}$$

$$\Delta\tau(r,s) = \tau_0 = (n \cdot L_{nn})^{-1} \tag{4}$$

$$\rho = 0.1 \tag{5}$$

1.5 'best tour evaluation'

This function is used to evaluate the length of all the ants' tours. Then, the shortest one is select as the best of the current iteration (and successively used in the global update). If the current best is better than the best reached so far, I also update some statistics such as the counter of how many tours have been generated to reach this optimal solution, the new absolute best length and the new absolute best tour.

1.6 'global update'

The global update function performs the global update of the pheromone according to the formula shown in the paper, which is reported below:

$$\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s) \tag{6}$$

$$\Delta\tau(r,s) = \begin{cases} (L_{best})^{-1} & \text{if } (r,s) \in \textit{Global best tour} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

$$\alpha = 0.1 \tag{8}$$

Only the the edges in the best tour get additional pheromone, while all the other lose pheromone according to the pheromone-decay parameter α. Before the global update, the two opt function is called to improve the best tour.

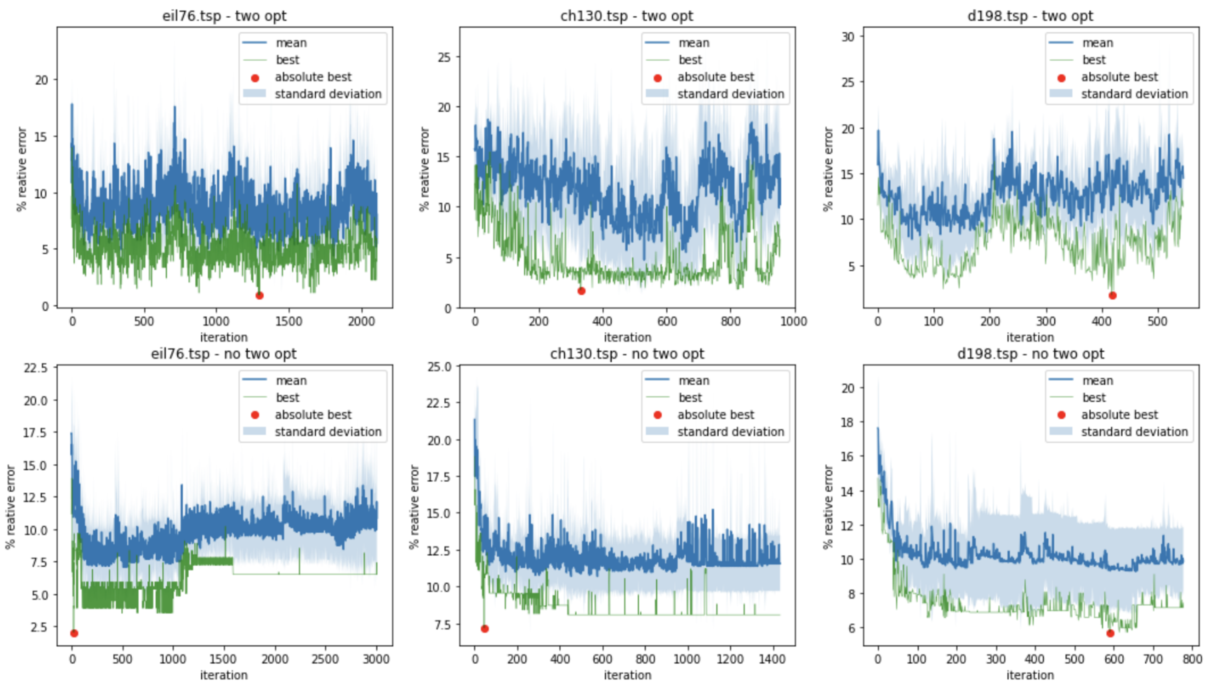
1.7 'solve'

This function calls iteratively the previous functions, as well as the two opt for best ant (with cl), in order to perform the ACS algorithm, as shown at the end of the paper. After 180 seconds (it can be changed) the function stops its execution.

2 Results

The results obtained are shown in the Colab notebook more in detail, anyway I report here the main table and the plots. The results (all) have been averaged among 5 different random seeds to have more statistically significant results.

Problem name	Two-Opt	Cities	Optimum	Best length	Tours generated until best	Relative error (%)
eil76.tsp	yes	76	538.0	545.8	774.8	1.4498
	no	76	538.0	569.0	399.2	5.7620
ch130.tsp	yes	130	6110.0	6271.8	560.8	2.6481
	no	130	6110.0	6688.2	188.8	9.4631
d198.tsp	yes	198	15780.0	16333.8	274.2	3.5095
	no	198	15780.0	17122.8	311.0	8.5095



I also report the table showing the absolute best result reach for each problem instance so far.

Problem name	Two-Opt	Cities	Optimum	Best length	Tours generated until best	Relative error (%)
eil76.tsp	yes	76	538.0	543.0	1294	0.9293
ch130.tsp	yes	130	6110.0	6215.0	333	1.7184
d198.tsp	yes	198	15780.0	16051.0	419	1.7173

3 Comments

As we can notice from the results given above, the two-opt (with candidate lists) is the one which gives the best performance. This is due to the property of the local search of improving a generated solution reaching his basin of attraction in the solution space (papers one and two of the course). Running the algorithm for only 180 seconds gives good approximations of the optimal solution, even for for the biggest problem instance, reaching in all three instances a best relative lower than 2%.