

COMPUTATIONAL COMPLEXITY

GRADO EN INGENIERÍA INFORMÁTICA

PRACTICE 5

AUTHOR: TEODORO MARTÍNEZ MÁRQUEZ

Cádiz, 2 de Junio de 2019

1. INTRODUCTION

In this practice I'm going to study the resolution of the Traveling Salesman Problem using an Ant Colony Optimization approach.

The reason behind choosing ACO is that this approach appears to find good results faster than other implementations, because the ants are fittingly searching an optimal path with a stochastic yet visibly intelligent behaviour giving more importance to an edge that has a better heuristic and pheromone value, whereas a GA approach would be much more randomised since it first builds random solutions and then chooses and randomly modifies the best ones. Besides, a GA is generally more time consuming, both in its bigger computation time and the more parameters that need to be adjusted.

2. METHODOLOGY

All of the code can be seen in the `antColony.cpp` file, which takes care of reading a `.tsp` file and finding a result for it.

First of all, we need a structure to store the information about the cities. In this case, a vector of `<city>` types, where a city simply contains an `x` and a `y` float coordinate. Via the `readfile` function, a file of the name of the parameter string is read and all of its cities stored in a vector of that caliber.

Next, we build a square matrix which contains the distances between every pair of cities, via the `getDistances` function, which receives the vector of cities and returns the matrix of distances, which we will call 'map' from now on. This matrix is obviously symmetrical, and its main diagonal is never accessed.

Afterwards, we have to set all the parameters for the ACO algorithm:

- `Randomseed`: A seed to set the randomisation
- `Max_it`: The amount of iterations the algorithm will do before stopping.
- `Num_ants`: The amount of ants the colony will simulate.
- `Decay_factor`: The factor to determine the speed of the pheromones' evaporation.
- `Alpha`: Factor to determine the importance of the heuristic value of the paths for an ant to choose.
- `Beta`: Factor to determine the importance of the pheromone value of the paths for an ant to choose.
- `C_phero`: Amount of pheromones that each ant will leave in its path.
- `C_elite`: Amount of extra pheromones that elite ants will leave in its paths, in the elitist expansion.
- `doExpansion`: Logical flag to determine whether or not the algorithm must use the elitist expansion.

With all previous steps done, it is time to start the main algorithm: the ACO function, receiving all the parameters above plus the vector of cities and the map.

The ACO implementation follows the standard algorithm: find paths, put pheromones, evaporate pheromones, repeat; plus the elitist expansion of extra pheromones for the current best path. The heuristic factor is determined as follows: `MAX_DIST` –

current_dist, where MAX_DIST is the maximum distance of the entire map, and current_dist the distance of the given edge.

However, there is one important factor that needs to be mentioned:

When an ant is forming a path, it tries to find the smallest circuit from the first city to the last, but it doesn't take into account the returning trip required to close the cycle. This is because each ant takes its current city and selects the next one based on a roulette selection with the heuristic and pheromone values of all possible edges from said current city. In order to make up for this loss of information, every ant is assigned one specific starting city, so that every city has an equal amount of ants starting from it (or at least as similar as possible if the division isn't exact) and the colony as a whole can determine a closed cycle.

Lastly, after the algorithm has finished, the best ant of the colony is returned. Its path cost is then processed and showed, along with the theoretical optimal of this instance.

3. RESULTS

Under these circumstances, the behaviour of the algorithm is certainly intriguing.

The optimal solution of the berlin sample is about 7.000, and the algorithm only finds solutions of about 21.000 with "default" values (most coefficients set to 1 except decay, set to 0.6). Changing the amount of iterations doesn't have an impact, as the solution barely changes from 50 iterations to 50.000 iterations. There is very little difference between using or not using the elitist expansion, as well as varying the pheromone coefficients, even with drastic amounts like tens of thousands. Assigning more ants to each city doesn't seem to factor in the results either.

However, the values of alpha and beta seem to be a key factor in the performance. A high beta value of about 10, along with various ants per city, a very high pheromone coefficient (at least at the same scale as a path cost, 10.000) and a high elite coefficient as well, the algorithm is capable of finding better solutions (~12.000) in around 1.000 iterations.

On the other hand, leaving all the previous values at default, and instead increasing alpha up to 30 returns better results much quicker (~9.000 in under 50 iterations).

Lastly, combining both parametrizations the algorithm is able to find slightly better solutions (8.500-8.700), again in around 1.000 iterations.

4. CONCLUSIONS

There is an interesting difference in the two ways that we can adjust the parameters of the algorithm. On one hand, we can give a lot emphasis to the concept of the ACO approach, giving high values to all the parameters regarding pheromone spreads, number of ants and pheromone importance, and eventually it is able to improve the solutions.

On the other hand, we can treat the algorithm like a series of individual local searches (giving most of the importance to the heuristic values), resulting in good solutions with little iterations, that do not improve overtime.

Finding a good balance of both visions is able to yield better solutions than either of them alone, but of course with a larger computation time than a local search, and the solutions found might be even better than the ones shown here if given enough number of iterations.

5. BIBLIOGRAPHY

Quick, generic guide for an ACO implementation:

<http://www.theprojectspot.com/tutorial-post/ant-colony-optimization-for-hackers/10>

Articles about the performance of an ACO approach for the TSP:

http://www.ef.uns.ac.rs/mis/archive-pdf/2011%20-%20No4/MIS2011_4_2.pdf

<http://www.diva-portal.org/smash/get/diva2:1214402/FULLTEXT01.pdf>