Emergence in Complex Systems - Alexandre Abou Chahine
Implementation of an Ant Colony optimisation algorithm applied to the traveling salesman problem

1. **Theory**
2. **Implementation**
3. **Results**

1. Theory

The Traveling Salesman Problem is a very famous problem in both computer science and operational research as it represents a whole class of combinatorial problems that can only be solved in non-polynomial time. Finding an exact solution to these problems is very expensive computationally. Hence heuristics solutions that are close to the optimal solution can sometimes be accepted. In the case of the traveling salesman problem, a heuristic might not be optimal but it will be sufficient for most applications.

The ant colony optimisation can achieve good heuristics on many combinatorial problems, here we implement it for the TSP. The TSP consists of finding the shortest route in a complete weighted graph G with n nodes and n(n-1)/2 edges, so that the start node and the end node are identical and all other nodes in this tour are visited exactly once.

The idea, as we saw in class, is that paths taken by ants are not very informative individually but can generate a good solution collectively. This is mainly achieved using positive feedback in the form of pheromone laid down on edges that are of interest.

The algorithm implemented is presented on the image below and implements the update rule as well as the initialisation of the routing table we saw in class. Multiple variations exist online but were not implemented due to timing constraints.

Algorithm 1: Ant Colony Optimisation for TSP

Result: Shortest Tour of the graph starting from node N
Input : A complete weighted graph
Output : Graph Path and the associated distance
Parameters: $\alpha, \beta, \rho, Q, P, I$

Initialisation: Creates a **complete weighted graph** from nodes and locate them on the euclidean space
Initialisation: Generate a population of ants of **size P**, with each individual is assigned a starting position
Initialisation: For each node initialise pheromone matrices to 1, and visibility matrices to $\frac{1}{d_{i,j}}$ between node i and j
for *I iterations* **do**
    instructions
    for *each* ***ant k*** *in the population* **do**
        Visit all nodes without visiting twice the same node
        Keep track of the visited nodes
        From node i pick the next node j to visit following a probabilistic transition rule:

$$P_{i,j}^{k} = \begin{cases} \frac{\tau_{i,j}^{\alpha} * \eta_{i,j}^{\beta}}{\sum_{k} \tau_{i,k}^{\alpha} * \eta_{i,k}^{\beta}} & j \notin visited \\ 0 & Otherwise \end{cases}$$

    end
    Update the pheromones matrices at each node using the paths from the population:

$$\tau_{i,j}(i+1) = (1-\rho) * \tau_{i,j}(i) + \sum_{k} \begin{cases} \frac{Q}{L_k} & (i,j) \in Path_k \\ 0 & Otherwise \end{cases}$$

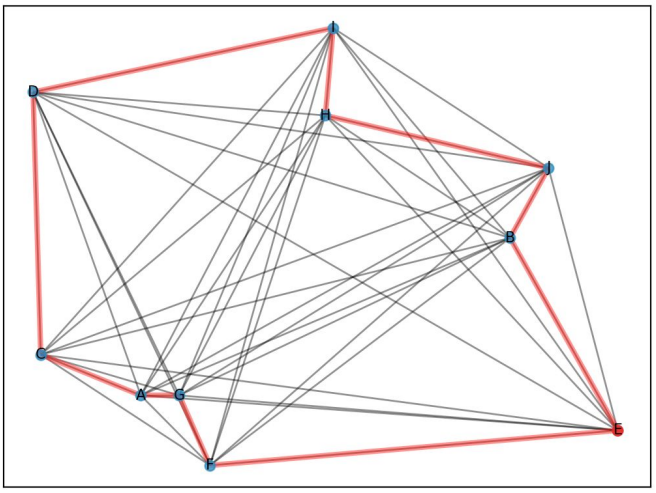
end

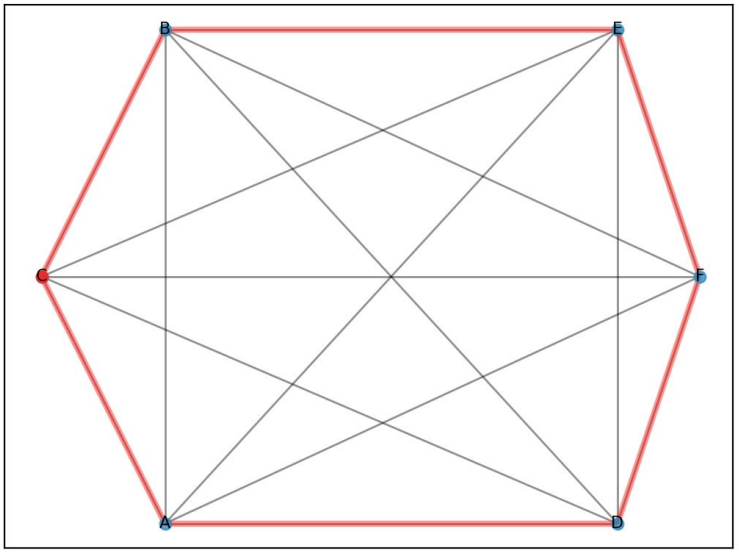
2. Implementation

The implementation was not achieved in evo life but could be integrated with ease as it mimics the classes used in antnet. Indeed 5 python classes are used to implement the algorithm (link, node, network, population and ants).

3. Result

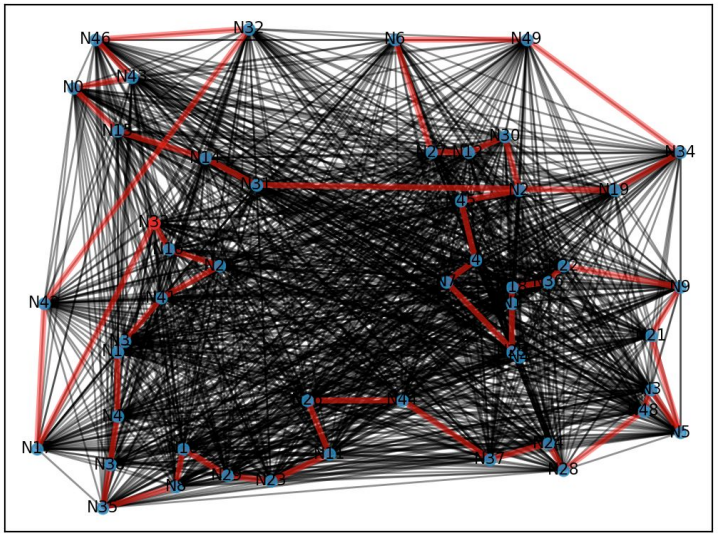
The algorithm seemed to work as expected on small graphs with coherent updates and sound results.

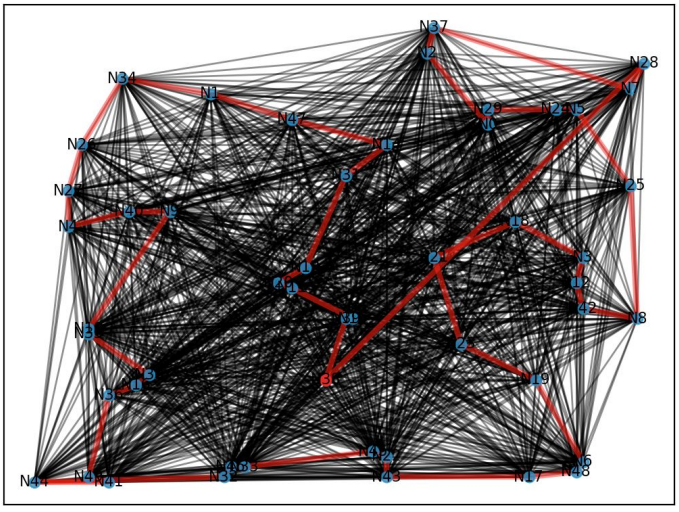


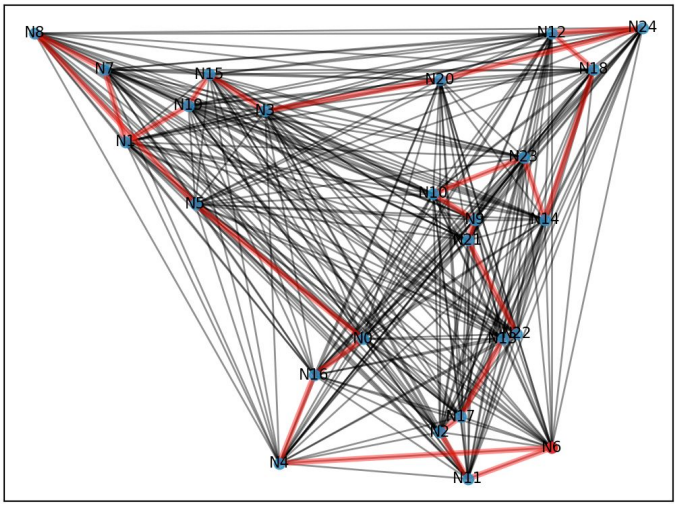


When expanding the number of nodes, I was not able to prove the correctness of the algorithm. Indeed this is where I've been spending most of my time and without conclusive results. Increasing the number of iterations does not seem to lead to better solutions and on the opposite the route taken by a message oscillates between worse solutions that the network had discovered before.

The solutions usually still seem pretty correct and have a good result compared to another solver as we can see on images below, representing graphs picked at random.







However it is the behaviour of the results that is bizarre, having tried many combinations of parameters without success. I decided to re-write the whole update rule thinking I might have

made an error and try different initial values, update mechanism… This was done without success, however trying to understand what could have gone wrong I read this paper (https://www.researchgate.net/publication/264855262_Solving_the_Travelling_Salesman_Problem_Using_the_Ant_Colony_Optimization) which seemed to have experienced something similar (cf. the noisy graphs presented in the result section).

In conclusion I believe the ACO algorithm has some very good properties as it can find good solutions in very few iterations but then it struggles to optimise it's output and ends up oscillating a lot. If I had more time I would have performed a complete benchmark of parameter selection and see how these should be selected with respect to a certain problem. At this point some of the parameters are very static and don't adapt with the graph input (example: the visibility matrix).

References

https://www.sciencedirect.com/science/article/pii/S1002007108002736
https://www.sciencedirect.com/science/article/pii/S0895717707000507
https://www.researchgate.net/publication/228652767_Ant_colony_optimization_method_for_generalized_TSP_problem
https://www.researchgate.net/publication/264855262_Solving_the_Travelling_Salesman_Problem_Using_the_Ant_Colony_Optimization