Université de Genève

Metaheuristics for optimization

TP 4: Ant System for Travelling Salesman Problem

Author: Joao Filipe Costa da Quinta

E-mail: Joao.Costa@etu.unige.ch

November 15, 2021

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

1 Introduction

During this TP we will be working on the Travelling Salesman Problem (TSP), this is a well known problem. It consists of a Salesman that is given n cities/places that he has to visit, we also assume that from any city, there is a path to visit any other city with a given distance. The goal is to find the best (shortest) path such that every city is visited exactly once, this path is also called the Hamiltonian cycle of minimal length on a fully connected graph.

2 Ant System

Ant System (AS) is an algorithm that is designed to copy the effect of nature itself, this is what we did during the last TP as well. This time we will copy the behaviour of a group of animals, a group of ants to be more specific, hence the name of the algorithm.

Ants are not extraordinarily smart individuals, however, a group of ants is capable of incredible things. More specifically, they are capable, as a group, of finding the shortest path between their nest and a food source. Many scientific studies were done to try and understand how they were capable of such things, once we found out, we simply tried to implement the same logic into an algorithm.

What happens in nature, is that ants will randomly take a path to try and find food, at every interval of time, let's say each second, they leave behind pheromones. When new ants are coming to try and find food, they won't choose a path completely at random, they will more likely choose a path that has more pheromone, as this presence of pheromone indicates that more ants took this path than any other path.

3 Ant System for TSP

Let's try and apply this algorithm to TSP problem, the goal is to start the ants in one given city, and define a path as complete, if they went through each city once, and ended up at the beginning, in this analogy, the food source is in the same place as the nest. The goal is to find the shortest path that respects the conditions.

If there are n cities we want to visit, a state will be a representation of which the Travelling Salesman (the Ant) visits in order. $S = [1, 2, 3, 1]$ this state means we start at city 1, go to city 2, then 3, and finally we comeback to city 1. (We want to finish were we started). In this example n = 3, but the length of our state is 4 = n + 1. The start and end are predetermined, which means the search space is $(n-1)!$.

The total distance for a given path, is as follows : $\sum_{i=1}^{n} D(s_i, s_{i+1})$ where $D(a, b)$ is the distance between city a and b, the energy of a state will be represented by the total distance. A neighbour of a given state, is a simple city permutation in the path.

Let's see how we implement these natural mechanisms in our algorithm.

In our algorithm, a given number of ants, will try and find the best path individually, each ant attempts to find the path a certain number of times, and we goal, is that in the last iteration one of the ants finds the best path possible. The pheromone will be left at the end of an iteration, and for the next iteration the ants can make an informed decision on which path to take, based on the amount of pheromone. However, we must keep in mind that the first iteration the path is chosen at random.

(1) Define set J, this set contains a list of the cities that are unvisited by a given ant. When we add a city to our path, we must delete it from J, which means that when J is empty, the ant has found a path that respects the conditions.

(2) First task is to compute the matrix D where D_{ij} is the distance between cities i and j.

(3) Compute η where $\eta_{ij} = \frac{1}{D_{ij}}$.

(4) We create a function that returns $\tau_{ij}(t)$, this is the intensity of the path between i and j at iteration t. This intensity is the amount of pheromones.

(5) Implement a function that updates the pheromones τ_{ij}. The pheromones are updated by following the next equation:

$$\tau_{ij}(t+1) = (1-\rho) * \tau_{ij}(t) + \sum_{k=1}^{\#ants=m} \Delta\tau_{ij}^{k}(t)$$

where ρ is the evaporation rate of the pheromones between each iteration, and $\Delta\tau_{ij}^{k}(t)$ is defined by the following equation:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L^k(t)} \text{ if ant } k \text{ used edge } (i,j) \text{ in its path} \\ 0 \text{ otherwise} \end{cases}$$

where Q is a predefined constant and L^k is the length of the path taken by the ant. This means that a longer path will have a smaller $\frac{Q}{L^k(t)}$, which results in less pheromone for the next iterations.

(6) Finally we have to implement a function that chooses which city we visit next in our path, this is done by following the next formula:

$$p_{ik}^{k}(t) = \begin{cases} \frac{(\tau_{ij}^{k}(t))^{\alpha} * (\eta_{ij})^{\beta}}{\sum_{l \in J} (\tau_{il}^{k}(t))^{\alpha} * (\eta_{il})^{\beta}} \text{ if } j \in J \\ 0 \text{ otherwise} \end{cases}$$

There are a couple of things to note in this formula. First there is a probability of 0 of choosing a city c such that $c \notin J$. Then there are the variables α and β, they can be tuned to augment the importance we git to the pheromone in a given path or the visibility of a given city.

Now we simply put all the puzzle pieces together, and we have a working algorithm that copies a colony of ants.

4 Results

We will run the algorithm in a unit circle of n cities, all at the same distance from the center, so the best solution is a path in shape of a circle, and then we will run the algorithm for 2 sets of cities that are given. Finally we will generate sets of larger cities, and see what happens.

For benchmark, we will use a simple greedy algorithm, this algorithm will go from city to city, always choosing the next village as the closest one that it hasn't visited yet. This algorithm is obviously deterministic.

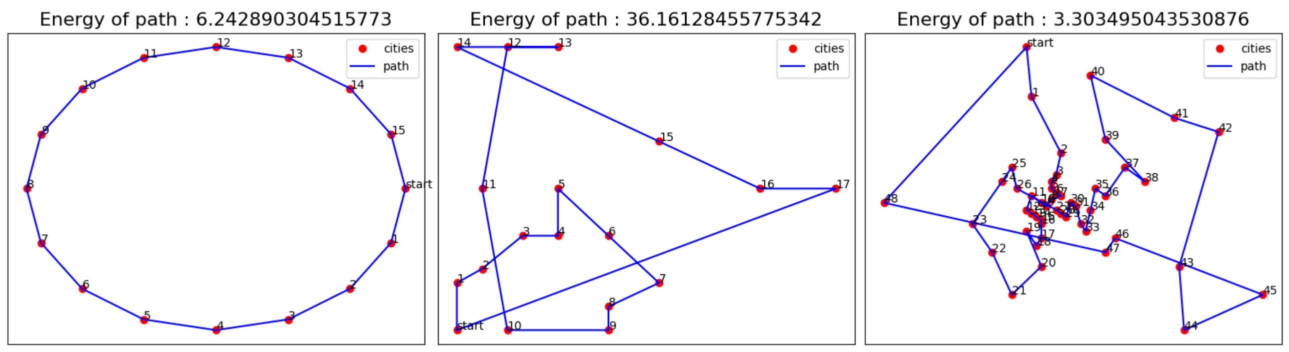

Figure 1: Path found by greedy algorithm. From left to right we have : unit circle, cities.dat, cities2.dat

We can see that the algorithm only works for a perfect case scenario, and as we increase difficulty, it begins to make a mess of its path.

Let's now recall the results that we got from Simulated Annealing algorithm for the TSP problem.

Unit circle mean energy for 10 runs was: 6.69

File cities.dat mean energy for 10 runs was: 28.68

File cities2.dat mean energy for 10 runs was: 2.88

Random problem generator for $n = \{50, 100, 150\}$

	greedy algorithm	SA best result	SA mean result
n = 50	136.10	115.85	123.65
n = 100	205.24	188.22	200.66
n = 150	243.78	240.92	259.68

For our algorithm there are a lot of variables, some of them were imposed, and others up to us, so lets make a list of the variables and their values, as well as why (if they are not imposed):

$\alpha = 1 \longrightarrow$ imposed

$\beta = 5 \longrightarrow$ imposed

$\rho = 0.1 \longrightarrow$ imposed

$Q = L_{nn} =$ solution determined by greedy algorithm $\longrightarrow$ imposed

$\tau_0 = \frac{1}{L_{nn}} \longrightarrow$ imposed

$t_{max} = 5 \longrightarrow$ not imposed. I choose this value because the algorithm wouldn't change the current best solution after the 5th iteration, for $m = n^2$

$m = n^2 \longrightarrow$ not imposed. I choose this value because it seemed to work well with 5 iterations, this way in the first iteration we maximize the search space we visited.

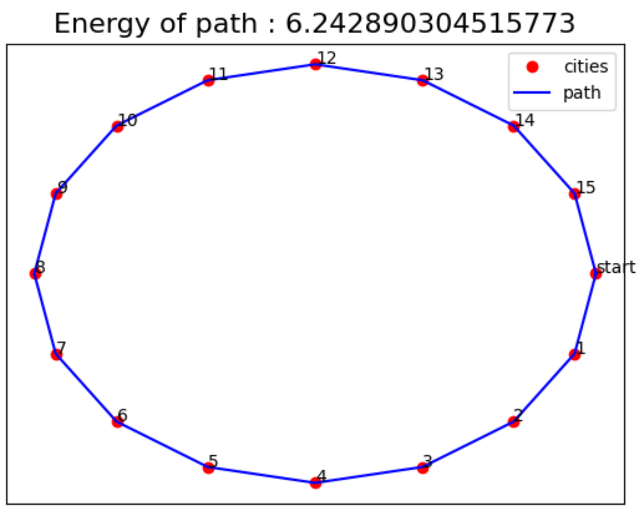

Figure 2: Found solution found by SA algorithm for unit circle. The energy mean for 10 runs was : 6.24

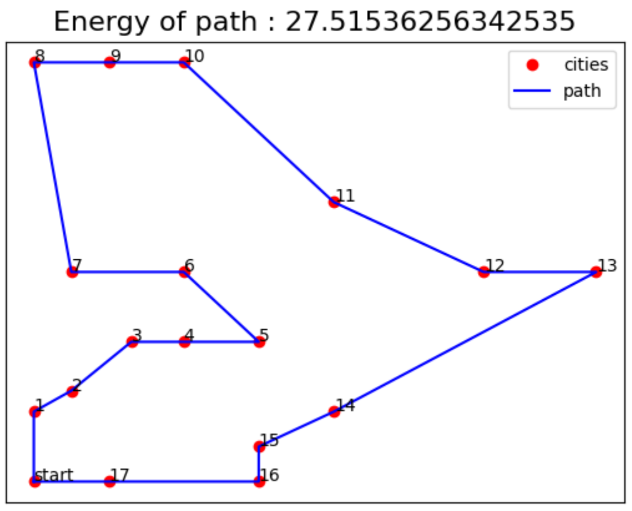

Figure 3: Found solution found by SA algorithm for cities.dat. The energy mean for 10 runs was : 27.51

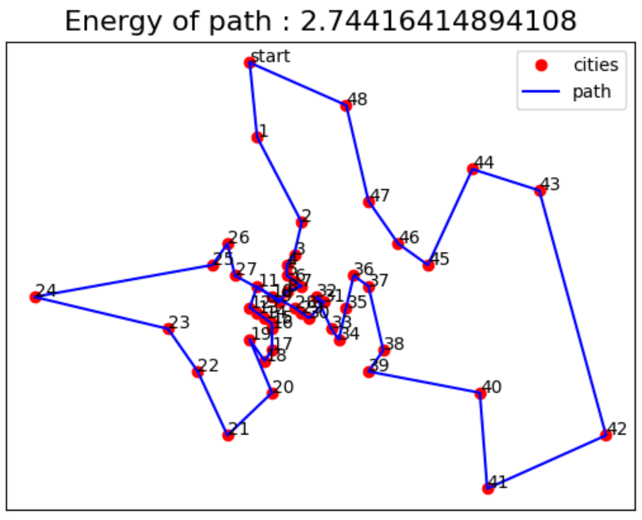

Figure 4: Found solution found by SA algorithm for cities.dat. The energy mean for 10 runs was : 2.79

As we can see, Ant system performed better than Simulated annealing in every problem. However, there is a cost to this improvement, it is time, lets compare both algorithms in time:

Time in seconds taken by both algorithms to solve each problem :

	Unit Circle	cities.dat	cities2.dat
Simulated Annealing	1.19	1.47	27.66
Ant System	1.33	2.60	116.01

We can easily see that there is a cost associated with the better results fond by Ant System ..

We could obviously argue that if we change the parameter m we would have a faster algorithm, but then it wouldn't be as good, and if it isn't as good, then there is less incentive to use it over Simulated Annealing.

For the final part of this results section, we will generate random problems of size $n = \{20, 40, 60, 80, 100\}$, and run each problem 10 times, and compare it to a greedy implementation. However, as the parameter $m = n^2$, it will just be too large, and take too much time, so for the purpose of these simulations I will set $m = 10 * n$.

Energy for each randomly generated problem, the AS result is the mean of 10 attempts at the same problem :

	20	40	60	80	100
Greedy	100.42	120.81	143.66	199.74	205.69
Ant System	84.48	104.55	133.22	155.94	174.65

Ant system performed better than Greedy, as expected, even with $m = 10 * n$. If we go back and look at the results of SA, we see that even with $m = 10 * n$ AS performed better than SA for the randomly generated problems.