Solving TSP using Ant Colony Optimization

Gustave CORTAL (ID: F10815023)

June 18, 2020

1 Introduction

The travelling salesman problem (TSP) is a NP-hard problem in combinatorial optimization. The goal is to find the shortest possible route between a list of cities knowing that we have distance between each pairs of cities. It is a NP-hard problem because finding the optimal solution is in a set of $n!$ different routes. Hence, if we use a naive approach such as bruteforcing all the possibilities, it would require n! steps. The TSP is similar to finding the shortest Hamiltonian circuit in a set of nodes, which is a famous problem in graph theory.

To solve this problem, we will implement an Ant Colony Optimization (ACO) which is part of swarm intelligence methods. ACO was initally proposed by Marco Dorigo in 1992 in his PhD thesis[1]. ACO algorithms try to mimic biological behaviors of ants in their colony. Ants use pheromones to communicate with each other and the colony in general. While exploring their environment, they lay down pheromones towards possible ressources. The goal is to simulate ants to solve the TSP. Different ants will explore the solutions space in order to find a cycle, they will then lay down pheromones proportional to the quality of their solutions. Through iterations, an optimal solution has to emerge from different solutions proposed by ants.

[1]M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano, Italy, 1992.

2 A randomized algorithm

2.1 Algorithm mechanism

We provide a common algorithmic skeleton for ACO algorithms :

```
procedure ACOBasic
  Set parameters, initialize pheromone trails
  while (termination condition not met) do
    ConstructAntsSolutions
    DaemonActions % optional
    UpdatePheromones
  end
end
```

The pheromone trails are associated with arcs. We will have a pheromone matrix whose elements are τ_{ij}. It refers to the desirability of visiting city j directly after city i. The heuristic information is chosen as $\eta_{ij} = 1/d_{ij}$, that is, the heuristic desirability of going from city i directly to city j is inversely proportional to the distance between the two cities.

Each ant will start at a random node. They will use pheromone and heuristic values to probabilistically construct a tour by iteratively adding cities that they have not visited yet. Each ant stop when they finally found a hamiltonian cycle and they may deposit pheromone to the followed tour. They will repeat the process until a termination condition is met - for example, a limited number of iteration.

2.2 Probabilistic action choice rule

The probability with which ant k, currently at city i, chooses to go to city j is :

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, if j \in N_i^k \tag{1}$$

η_{ij} $(= 1/d_{ij})$ is the heuristic desirability of visiting city j directly after city i, τ_{ij} is the pheromone desirability of visiting city j directly after city i, α and β determines the relative influence of the pheromone trail and the heuristic information, and N_i^k is the set of cities that ant k has not visited yet.

If $\alpha = 0$, the closest cities are more likely to be selected: this corresponds to a classic stochastic greedy algorithm such as the nearest neighbor algorithm. If $\beta = 0$, we only take in account pheromone desirability without any heuristic information. It will rapidly lead our algorithm to be in a stagnation situation where all ants follow the same path. We have to find a good trade-off between exploration and exploitation by fine-tuning those parameters.

2.3 Update of pheromone trails

We introduce pheromone evaporation during the pheromone matrix update :

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \forall (i,j) \in L \tag{2}$$

The parameter ρ is used to avoid unlimited accumulation of the pheromone trails and it enables the algorithm to forget bad decisions previously taken because if an arc is not chosen by ants, the corresponding pheromone value will decrease exponentially.

During the pheromone update, all ants deposit pheromone on the arcs they have crossed :

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \forall (i,j) \in L \tag{3}$$

$\Delta\tau_{ij}^{k}$ is the amount of pheromone ant k deposits on the arcs it has visited. $\Delta\tau_{ij}^{k} = 1/C^{k}$ if arc (i,j) belongs to T^{k} where C^{k} is the length of the tour T^{k} built by the k-th ant. Arcs that are used by many ants receive more pheromone and are therefore more likely to be chosen by ants in future iterations of the algorithm.

3 Improvements

3.1 Elitist Ant System

Elitist strategy for Ant System (ASE) has been introduced by Dorigo in 1992. The idea is to implement a specific daemon action which will add more pheromones to the arcs belonging to the best tour found since the start of the algorithm. It is our new pheromone update equation :

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs}, \forall (i,j) \in L \tag{4}$$

Where, $\Delta\tau_{ij}^{bs} = 1/C^{bs}$ if arc (i,j) belongs to T^{bs} where C^{bs} is the length of the best-so-far tour T^{bs}. e is a parameter that defines the weight given to the T^{bs}.

3.2 Initial pheromone values

For basic Ant System (AS), we initialize our pheromone matrix as followed:

$$\tau_0 = m/C^{nn} \tag{5}$$

For Elitist Ant System (ASE), we initialize our pheromone matrix as followed:

$$\tau_0 = (e+m)/\rho C^{nn} \tag{6}$$

Where C^{nn} is the tour length found by applying nearest neighbor (NN) algorithm at a random node. This is useful to give us an idea about the optimal solution length, hence the convergence towards good solutions is faster as the pheromone matrix is useful even at first iterations.

3.3 Detect stagnation

We analyze several ways of detecting stagnation. One of the simplest possibilities is to compute the standard deviation σ_L of the length of the tours the ants construct after every iteration. If σ_L is zero, it means all the ants follow the same path. A better choice is to use the variation coefficient which is independent of the scale. It is defined as the tour lengths divided by the average tour length. Experimental results show that it's still not a good way to detect stagnation for ACO algorithm.

The Shannon entropy is a useful index. We can use the average $\overline{\varepsilon} = \sum_{i=1}^{n} \varepsilon_i / n$ of the entropy ε_i of the selection probabilities at each node :

$$\varepsilon_i = -\sum_{j=1}^{l} p_{ij} log(p_{ij}) \tag{7}$$

Where p_{ij} is the probability of choosing arc (i,j) when being in node i, and l, $1 \leq l \leq n-1$, is the number of possible choices.

We can also verify empirically the stagnation by looking at the pheromone matrix heatmap. Images are provided in the project presentation.

3.4 A Neural Network idea: Dropout

We study a new way of avoiding stagnation. Our novel idea comes from neural networks research. The famous problem of overfitting can be seen as a stagnation problem where the model converges towards a local optimum. To prevent neural network from overfitting, this paper[2] recommends using Dropout. The key idea is to randomly drop units and their connections from the neural network during training. By using this method during the training stage, a fully connected layer model becomes more robust and the test error decreases. We can use this idea to enhance our algorithm. We use a special daemon action which will randomly vanish some arcs of the best iteration tour during the tour construction phase of each ants. We create a new pheromone matrix whose dropped arcs are temporary equal to 0. It means that during the construction tour, ants will not select those arcs because their corresponding probability will

[2] S. Nitish, C.G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from overfitting*, Journal of Machine Learning Research, 2014.

be 0. Another idea is a analogy from the concept of DropConnect[3]. DropConnect is the generalization of Dropout in which each connection in a neural network, rather than each output unit, can be dropped with probability $1 - p$. In other words, the fully connected layer with DropConnect becomes a sparsely connected layer in which the connections are chosen at random during the training stage. We find a similar idea for ACO algorithm. During the pheromone update stage, we will choose to randomly add or not new pheromones on the best iteration tour arcs.

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs} * B_{ij}^{bs}, \forall (i,j) \in L \tag{8}$$

Where $B_{ij}^{bs} \sim Bernoulli(p)$. p is a new parameter to tune which decides whether or not we add pheromones to a specific arc ($B_{ij}^{bs} = 1$ or 0) with a certain probability.

The idea is similar to adding noises to the tour construction procedure. The entropy will be higher and therefore we will increase the search space around good solutions and avoid the ants to always follow the same path.

Experimental results show that we get same results by using Normal distribution function instead of Bernoulli distribution function.

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs} * N_{ij}^{bs}, \forall (i,j) \in L \tag{9}$$

Where $N_{ij}^{bs} \sim N(1, \sigma^2)$. The mean is set to 1 and σ, the standard deviation, is a new parameter to tune which decides the amount of noises we want around the best iteration tour. Our next section shows that the idea of Dropout apply to ASE increase efficiency and may be a good improvement for other ACO algorithms.

[3]Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. 2013. *Regularization of neural networks using DropConnect.* In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28.

4 Experimental results: benchmarks

We run basic Ant System (AS), Elitist Ant Sytem (ASE) and Elitist Ant System with Dropout (ASE+Drop) on three different benchmarks provided by TSPLibrary[4] (eil51, berlin52 and eil76). We also provide results from two famous algorithm combined (NN+2-opt): we use nearest neighbor (NN) and then apply 2-opt on the solution found by the first algorithm. We compare the gap between the optimal solution (opt) and our algorithm solutions. All experiments have been run 10 times with $k = 300$ iterations. Gap error is calculated as followed : $(Avg - opt)/opt * 100$, where Avg is the average minimum tour length.

For our experimental study, we are using the followed parameters values: $\alpha = 1$, $\beta = 3.5$, $\rho = 0.5$, $m = n$ (number of cities). Those values are known to provide good results with a lot of TSP instances[5].

TSP (opt)	Algorithm	Best	Worst	Avg	Std	Gap
eil51 (426)	ASE+Drop	430.35	437.60	433.66	2.07	1.79%
	ASE	432.33	440.83	435.74	3.12	2.29%
	AS	443.35	455.26	450.76	3.39	5.81%
	NN+2-opt	438.72	514.62	476.61	22.57	11.89%
berlin52 (7542)	ASE+Drop	7544.37	7544.66	7544.46	0.14	0.03%
	ASE	7544.37	7663.21	7558.22	36.96	0.22%
	AS	7549.29	7677.12	7622.91	53.62	1.07%
	NN+2-opt	8042.95	8894.5	8448.54	326.69	12.02%
eil76 (538)	ASE+Drop	546.19	555.90	550.89	3.49	2.39%
	ASE	547.33	558.25	551.96	3.41	2.59%
	AS	558.17	570.54	565.45	3.81	5.10%
	NN+2-opt	568.46	630.15	597.18	22.26	11.00%

Table 1: Experimental results on three different benchmarks

We can observe that NN+2-opt is outperformed by all AS algorithm as the gap error is very high. Moreover, by combining Dropout and ASE, our algorithm is improved. It always find the best tour - but still can't reach any optimal solution. We can also observe that the average minimum tour and the standard deviation are reduced. It means that not only ASE+Drop can find better solutions, but the solutions provided are also stabler - with berlin52, ASE+Drop almost always find the best solution. In the future, the concept of DropOut may be implemented in other ACO algorithms.

[4]`http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/`

[5]M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano, Italy, 1992, p. 71.

5 Solve a practical problem

5.1 Taipei Youbike Stations

We propose to solve a real life problem using our improved algorithm, the Elitist Ant System with Dropout (ASE+Drop). We use the Taipei Youbike stations dataset provided by Taipei City[6]. Youbike is a public bicycle sharing service offered by the Taipei City. The dataset contains hundred of rental bike stations around Taipei. Our goal is to find the shortest path between those station. It is a symmetric TSP because the distance between two stations is the same in each opposite direction, forming an undirected graph. Our result may be useful to monitor different stations in case they need physical maintenance.

We use the Google Maps API[7] to get the distance matrix between the stations. It cost 5$ for 1000 elements, which is equivalent to a distance matrix of $\sqrt{1000} = 31$ cities. As the service is not free, we use a subset which contains 50 stations selected randomly.

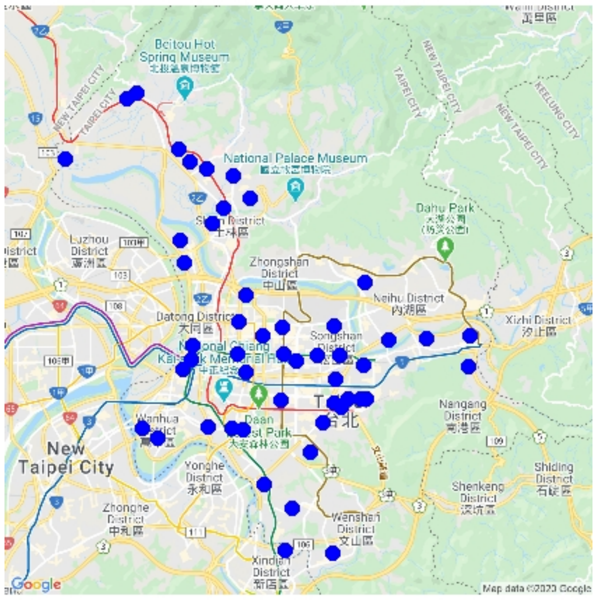

Figure 1: Youbike stations (blue nodes) in Google Maps

[6]https://data.taipei/#/dataset/detail?id=8ef1626a-892a-4218-8344-f7ac46e1aa48
[7]https://developers.google.com/maps/documentation

5.2 Results

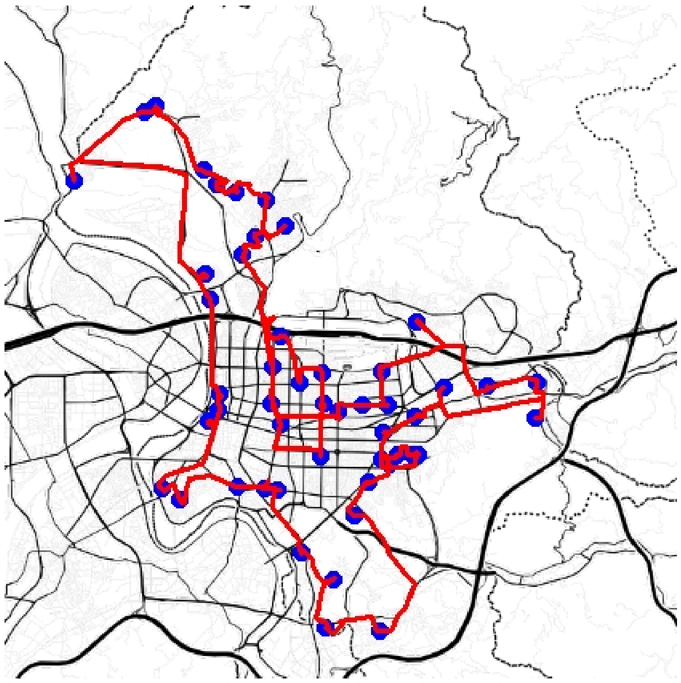

Figure 2: Youbike stations (blue nodes) with best found tour in Google Maps

We run our algorithm one time using our subset of stations with 1000 iterations. We keep the parameters used for testing our benchmarks. The best solution length found is 110961 meters. By tracing a real route between the stations using the best tour found, we obtain the following result. We calculate the real distance using the distance of each road used. Using Google Directions API[8], our result is 112961 meters and it takes 335.83 minutes to reach every nodes. The real distance is larger than the distance calculated by our algorithm because we assume that the distance between the stations are Euclidean whereas for city routes it may be more relevant to use Manhattan distance.

[8]https://developers.google.com/maps/documentation/directions/intro?hl=en