

ASSIGNMENT #5

Computational Intelligence: SACO
(Simple Ant-Colony Optimization)

Marwan Mohamed Nabil
20200512

Table of Contents

PART 1: Problem formulation & Objectives.....	3
1. The Problem	3
2. Objective:	3
3. About ACO:.....	3
 PART 2: System Overview	5
1. System components:.....	5
2. System Analysis:	5
a. Initializing the initial matrices.	5
b. Tour length using nearest neighbor heuristic from a randomly chosen initial node.	5
c. Set the initial tau matrix.	6
d. The State Transition Rule (i.e., find the next city to visit).....	6
e. Tour Construction.	6
f. Removing cycles:.....	7
g. Apply the pheromone update rules.	7
h. Main ACO Function.	7
 PART 3: Results and Conclusions	8
1. Results Analysis	8
a. Number of ants:	8
b. Alpha and Beta values:.....	9
c. Evaporation rate (Rho):	9
2. Conclusion	10
 PART 4: Code link and additional Screenshots:	11
Link:	11
Screenshots:.....	11

PART 1: Problem formulation & Objectives

1. The Problem

-We have a dataset (TSPDATA.txt), which contains the x and y coordinates of about 100 cities. We are required to implement the Ant Colony System (ACS) to solve the problem of the TSP.

2. Objective:

-We are requested to compute the shortest tour starting at every city, where a tour is described by:

- Starting at a city
- Visit all other cities “only once.”
- Return to the starting city.

3. About ACO:

Ant Colony Optimization (ACO) is a metaheuristic algorithm that is inspired by the behavior of ant colonies in nature. ACO is used to solve optimization problems by imitating the foraging behavior of ants.

In nature, ants communicate with each other by leaving pheromone trails. Ants deposit pheromones on the ground as they move in search of food. These pheromone trails act as signals to guide other ants to the food source. The more ants that follow a particular trail, the stronger the trail becomes due to the increased pheromone concentration. This feedback loop allows ants to effectively explore and exploit their environment and ensure efficient food collection.



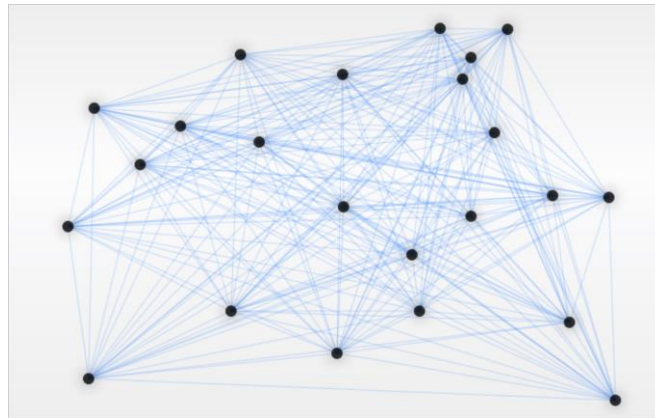
In ACO, the problem is modeled as a graph, where the nodes represent potential solutions, and the edges represent the transition between them. Each ant in the colony represents a potential solution to the problem. Initially, the ants are randomly distributed across the nodes of the graph.

As the ants move through the graph, they deposit pheromone on the edges they traverse. The amount of pheromone deposited is proportional to the quality of the solution represented by that edge. Ants prefer to follow edges with higher pheromone concentrations, which indicates a higher quality solution. Over time, as more ants follow the same path, the pheromone concentration on that path increases, making it more attractive to other ants.

The ACO algorithm uses a probabilistic approach to guide the ants' movement. At each node, an ant chooses the next node to visit based on a combination of the pheromone concentration on the edges and a heuristic function that estimates the quality of the solution represented by that node. The heuristic function is problem-specific and is designed to bias the ants towards more promising solutions.

The ACO algorithm continues until a stopping criterion is met, such as a maximum number of iterations or a satisfactory solution is found. The best solution found by the ants during the search is returned as the output of the algorithm.

ACO has been successfully applied to a wide range of optimization problems, including the traveling salesman problem, the job shop scheduling problem, and the vehicle routing problem. ACO has also been used in combination with other metaheuristic algorithms to improve their performance.



PART 2: System Overview

1. System components:

-Let's first look at our system environment:

System	Entity	Attributes	Activities	Events	State Variable
TSP	Ants	Pheromone Conc.	Routing	Initializing Population	Best_tours
	Cities	Position	Calculating probability	Cycling	Best_tour_lengths
		Attractiveness	Comparing probabilities	Reaching Optimal solution	

2. System Analysis:

-To solve this problem, we're going to use Python programming language on a Jupyter Notebook.

-We're using the NumPy library to ease some of the calculations and with initializing random numbers for the initial population.

-Here is a list of the functions used and their usage:

a. Initializing the initial matrices.

```
def initialize_data(df):
```

-In this part of the code, we start by importing the dataset and fetching the number of cities we want to reach "n". In our case it was 30.

-We then assign the x coordinates to a list X, and likewise for the y coordinates.

-We then initialize a matrix "dist" of size n x n, which contains the Euclidean distance from each city to the rest, which will be used in deciding on the tour of the ants.

-and finally, a matrix "eta" of size n x n which contains the inverse of these Euclidean distances.

b. Tour length using nearest neighbor heuristic from a randomly chosen initial node.

```
def nearest_neighbor_tour(df, dist):
```

-This function is used to find the shortest route from a random node.

-We use this random shortest path to determine the initial pheromone concentration, which will fill the "tau" matrix we will construct, that contains all pheromone values.

c. Set the initial tau matrix.

```
def initialize_pheromones(df, Lnn):
```

-In this function, we use the random shortest route we previously calculated to determine the initial pheromone concentration on each path of the n cities.

d. The State Transition Rule (i.e., find the next city to visit)

```
def state_transition_rule(current_city, unvisited, tau, eta, alpha, beta):
```

-While ants have not constructed a complete tour, we apply the state transition rule to each one.

-The state transition rule in Ant Colony Optimization (ACO) is a probabilistic rule used by each ant to determine the next node to visit during the construction of a solution. The state transition rule is based on two factors: the pheromone trail strength and a heuristic value.

- The heuristic value is a problem-specific value that estimates the desirability of visiting a particular node.

- The probability for each path is determined by the following equation:

$$\bullet \quad P_{ij}^k(t) = \frac{t_{ij}^{\alpha}(t) \cdot n_{ij}^{\beta}(t)}{\sum_{j \in N_t^k} t_{ij}^{\alpha}(t) \cdot n_{ij}^{\beta}(t)}$$

- α and β are control parameters, where $\alpha \geq 0$ and $\beta \geq 1$.

e. Tour Construction.

```
def construct_tour(ant, df, tau, eta, alpha, beta):
```

-This is the function that is responsible for planning the next node ant “k” is going to visit.

-The function first determines the number of cities n in the DataFrame df, and if the starting city index ant is greater than or equal to n, it resets ant to the equivalent index within the range of n. The function then sets the start_city variable to ant, creates a set of unvisited cities unvisited, and initializes the tour list with the start_city.

-The function then enters a while loop that continues until all cities have been visited. In each iteration of the loop, the function determines the current city current_city, uses the state_transition_rule function with the current city, unvisited cities, the pheromone and heuristic arrays, and the alpha and beta parameters to determine the next city next_city to visit. The next_city is added to the tour list, and removed from the unvisited set.

-Finally, the start_city is added to the end of the tour list to form a complete tour, and the tour list is returned.

f. Removing cycles:

```
def remove_cycles(tour):
```

-In this function, we observe the returned tour after constructing one previously, and check for cycles.

-If a cycle exists, it is removed from the tour and the tour is fixed up

g. Apply the pheromone update rules.

```
def update_pheromones(tau, tours, dist, rho):
```

-The function begins by initializing a matrix `delta_tau` of the same shape as `tau` with all entries set to zero. This matrix will accumulate the amount of pheromone deposited on each edge by all ants.

-The function then loops over each tour taken by all ants, and for each edge in the tour (except the last one), it updates the corresponding entries in `delta_tau`. Specifically, for an edge between nodes `a` and `b`, the function adds $1/\text{dist}[a, b]$ to `delta_tau[a, b]` and `delta_tau[b, a]`. This means that the amount of pheromone deposited on an edge is proportional to the inverse of its distance.

-Finally, the function updates the pheromone matrix `tau` by combining the previous pheromone levels with the newly deposited pheromone as follows: $\text{tau} = (1 - \text{rho}) * \text{tau} + \text{delta_tau}$. This means that the pheromone evaporates by a factor of `rho` and the new pheromone deposited by the ants is added to the remaining pheromone.

h. Main ACO Function.

```
def ant_colony_system(df, m, alpha, beta, rho, iterations):
```

-The function begins by initializing the distance matrix `dist` and the heuristic information matrix `eta` using the `initialize_data()` function. It also initializes the pheromone matrix `tau` using the `initialize_pheromones()` function, and computes a nearest neighbor tour using the `nearest_neighbor_tour()` function.

-The function then sets up a loop to run the ACS algorithm for the specified number of iterations. In each iteration, it constructs a tour for each ant using the `construct_tour()` function, which uses a combination of pheromone trail and heuristic information to select edges. It then removes any cycles in the resulting tours using the `remove_cycles()` function.

-The function also keeps track of the best tour found so far for each ant and updates it if a better tour is found. This information is stored in the `best_tours` dictionary and `best_tours_lengths` list.

-After all ants have completed their tours, the function updates the pheromone matrix using the `update_pheromones()` function.

-Finally, the function returns the best tours found for each ant and their corresponding lengths as a tuple of dictionaries (`best_tours` and `best_tours_lengths`).

PART 3: Results and Conclusions

1. Results Analysis

-In this section we will observe the outputs of this run.

-And then discuss whether they reflect a good system or not.

-We also need to decide on the values of the used parameters, which are:

1. Number of ants
2. Alpha value
3. Beta value
4. Rho value

a. Number of ants:

Changing the number of ants in Ant Colony Optimization (ACO) can have a significant effect on the algorithm's performance.

Increasing the number of ants generally leads to better exploration of the search space and can help the algorithm find better solutions. This is because more ants mean more tours are constructed, and each tour represents a potential solution to the problem.

However, increasing the number of ants can also make the algorithm slower and more resource intensive. This is because each ant needs to construct a tour, which involves evaluating the pheromone trail and heuristic information for each edge in the graph. This process can be computationally expensive, especially for large graphs.

On the other hand, decreasing the number of ants can lead to faster execution times, but may result in the algorithm getting stuck in local optima and failing to find the global optimum.

Therefore, the number of ants in ACO should be chosen carefully based on the problem at hand and the available computing resources. In practice, the optimal number of ants is often found through experimentation and tuning.

(Number of Iterations: 20, Alpha value: 1, Beta value: 5, Rho value: 0.1)

1. Number of ants $(k) < n$

-Since the number of ants is less than the number of cities, this choice will only work on only the first 'k' cities.

-Which makes this the worst option.

2. Number of ants $(k) = n$ "30"

-This yielded an average tour length of 44.530276546434564

3. Number of ants ($k > n$ "60")

-This yielded an average tour length of 46.32253516880254

-but twice the computation time of $n = 30$

4. Number of ants ($k > n * 2$ "90")

-This yielded an average tour length of 45.648490217244756

-And 3 times more computation time.

Therefore, the best number of ants should be between n and $2n$, to keep computation time limited in a viable region.

b. Alpha and Beta values:

-The best alpha and beta values for Ant Colony Optimization (ACO) depend on the specific problem being solved and the characteristics of the problem instance. Alpha and beta are parameters that control the relative influence of the pheromone trail and heuristic information in the ant's decision-making process.

-In general, a higher value of alpha places more emphasis on the pheromone trail, while a higher value of beta places more emphasis on the heuristic information. The optimal values of alpha and beta can be found through experimentation and tuning.

-A common approach to finding good values for alpha and beta is to use a grid search or a random search over a range of possible values. Simulated annealing or other optimization techniques can also be used to find the optimal values.

-In practice, it is often a good idea to start with a range of values that have been shown to work well in similar problems, and then refine them through experimentation on the specific problem instance.

c. Evaporation rate (Rho):

-The evaporation rate is another important parameter in Ant Colony Optimization (ACO), as it determines the rate at which the pheromone trail evaporates.

-A high evaporation rate can help prevent premature convergence, but it can also lead to a loss of information about good solutions. A low evaporation rate can help preserve good solutions, but it can also lead to stagnation and slow convergence.

-In general, a good starting point for the evaporation rate is to set it to a value between 0.1 and 0.5. From there, the evaporation rate can be adjusted through experimentation and tuning to find the optimal value for the specific problem instance.

2. Conclusion

-It is important to note that the optimal values of alpha, beta, and the evaporation rate are interdependent, and adjusting one parameter may require adjusting the others as well to achieve the best results.

-Therefore, a systematic approach to parameter tuning is recommended to ensure the best possible performance of the ACO algorithm.

But generally:

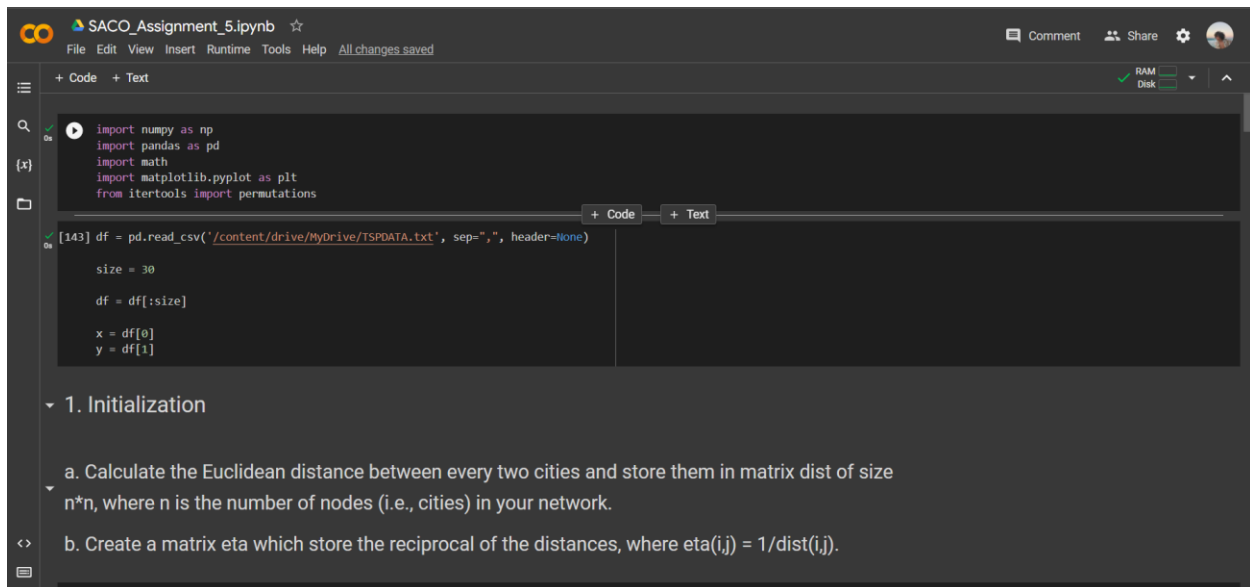
- Number of ants: The number of ants should be proportional to the number of cities in the problem. A common rule of thumb is to use 10-20 ants per city, so for a problem with 30 cities, the number of ants could be set to 300-600.
- Alpha and beta: The values of alpha and beta control the relative influence of the pheromone trail and heuristic information in the ants' decision-making process. Alpha should be higher than beta to give more weight to the pheromone trail. A common starting point for alpha is 1.0, and for beta is 2.0, but these values can be adjusted through experimentation.
- Rho (evaporation rate): The value of rho determines the rate at which the pheromone trail evaporates. A high value of rho (e.g., 0.9) can help prevent premature convergence, while a low value of rho (e.g., 0.1) can help preserve good solutions. A common starting point for rho is 0.5, but this value can also be adjusted through experimentation.

PART 4: Code link and additional Screenshots:

Link:

[SACO Assignment 5.ipynb - Colaboratory \(google.com\)](#)

Screenshots:



The screenshot shows a Jupyter Notebook titled "SACO_Assignment_5.ipynb". The code cell contains the following Python code:

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from itertools import permutations

[143]: df = pd.read_csv('content/drive/MyDrive/TSPDATA.txt', sep=",", header=None)

size = 30

df = df[:size]

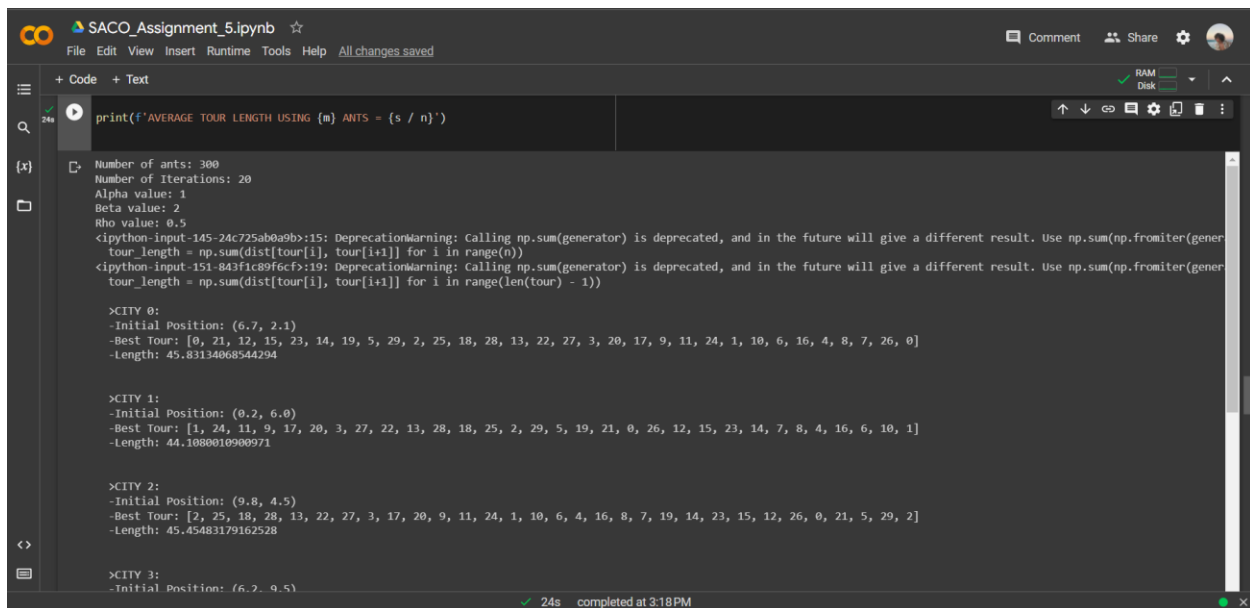
x = df[0]
y = df[1]
```

Below the code cell, the notebook displays the following text:

1. Initialization

a. Calculate the Euclidean distance between every two cities and store them in matrix dist of size $n \times n$, where n is the number of nodes (i.e., cities) in your network.

b. Create a matrix eta which store the reciprocal of the distances, where $\eta(i,j) = 1/\text{dist}(i,j)$.



The screenshot shows the same Jupyter Notebook interface, but with the output of the algorithm displayed. The code cell contains the following Python code:

```
print(f'AVERAGE TOUR LENGTH USING {m} ANTS = {s / n}')
```

The output of the code cell is as follows:

```
Number of ants: 300
Number of Iterations: 20
Alpha value: 1
Beta value: 2
Rho value: 0.5
<ipython-input-145-24c725ab0a9b>:15: DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future will give a different result. Use np.sum(np.fromiter(generator), dtype='float', axis=0).
tour_length = np.sum(dist[tour[i], tour[i+1]] for i in range(n))
<ipython-input-151-843f1c89f6cf>:19: DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future will give a different result. Use np.sum(np.fromiter(generator), dtype='float', axis=0).
tour_length = np.sum(dist[tour[i], tour[i+1]] for i in range(len(tour) - 1))

>CITY 0:
-Initial Position: (6.7, 2.1)
-Best Tour: [0, 21, 12, 15, 23, 14, 19, 5, 29, 2, 25, 18, 28, 13, 22, 27, 3, 20, 17, 9, 11, 24, 1, 10, 6, 16, 4, 8, 7, 26, 0]
-Length: 45.83134068544294

>CITY 1:
-Initial Position: (0.2, 6.0)
-Best Tour: [1, 24, 11, 9, 17, 20, 3, 27, 22, 13, 28, 18, 25, 2, 29, 5, 19, 21, 0, 26, 12, 15, 23, 14, 7, 8, 4, 16, 6, 10, 1]
-Length: 44.1088010900971

>CITY 2:
-Initial Position: (9.8, 4.5)
-Best Tour: [2, 25, 18, 28, 13, 22, 27, 3, 17, 20, 9, 11, 24, 1, 10, 6, 4, 16, 8, 7, 19, 14, 23, 15, 12, 26, 0, 21, 5, 29, 2]
-Length: 45.45483179162528

>CITY 3:
-Initial Position: (6.2, 9.5)
```

The notebook interface shows that the code was executed successfully, with a status bar at the bottom indicating "24s completed at 3:18PM".

```
SACO_Assignment_5.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

24s
✓
X-CITY 25:
-Initial Position: (8.9, 5.0)
-Best Tour: [25, 2, 29, 5, 19, 21, 0, 26, 12, 15, 23, 14, 7, 8, 4, 16, 6, 10, 1, 24, 11, 9, 17, 20, 3, 27, 22, 13, 28, 18, 25]
-Length: 44.1088010900971

X-CITY 26:
-Initial Position: (6.4, 3.0)
-Best Tour: [26, 12, 15, 23, 14, 7, 8, 4, 16, 6, 10, 1, 24, 11, 9, 17, 20, 3, 27, 22, 13, 28, 18, 25, 2, 29, 5, 19, 21, 0, 26]
-Length: 44.10880109009708

X-CITY 27:
-Initial Position: (7.7, 6.8)
-Best Tour: [27, 22, 13, 28, 18, 25, 2, 29, 5, 0, 21, 26, 12, 15, 23, 14, 19, 7, 8, 4, 16, 6, 10, 1, 24, 11, 9, 17, 20, 3, 27]
-Length: 45.11407571343239

X-CITY 28:
-Initial Position: (7.2, 5.2)
-Best Tour: [28, 18, 25, 2, 29, 5, 19, 14, 23, 21, 0, 26, 12, 15, 10, 6, 7, 8, 4, 16, 1, 24, 11, 9, 20, 17, 3, 27, 22, 13, 28]
-Length: 45.13670429491479

X-CITY 29:
-Initial Position: (10.0, 0.5)
-Best Tour: [29, 5, 19, 21, 0, 26, 12, 15, 23, 14, 7, 8, 4, 16, 6, 10, 1, 24, 11, 9, 17, 20, 3, 27, 22, 18, 28, 13, 25, 2, 29]
-Length: 44.38173343993388

AVERAGE TOUR LENGTH USING 300 ANTS = 44.933150992304924

24s completed at 3:18PM
```

