Course Name: Computational Intelligence.
Course Code: DS313/DS351.
Problem title: Routing problem.
Member Names:

ID	Name
20200039	Ahmed Mohamed TONSY
20200007	Ahmed Ibrahim Mahmoud
20201227	Youssef Atallah Mohamed
20201020	Asala Ehab Mohammed

Table of Contents

Routing By Particle swarm optimization.

What is Routing problem:

Routing problem refers to the challenge of determining the optimal or most efficient route for moving objects or information from one location to another within a network or transportation system. It is a fundamental problem in various domains, including transportation, logistics, telecommunications, and computer networks. In transportation and logistics, routing problems typically involve finding the best path for vehicles to follow to deliver goods or passengers while considering constraints such as time, distance, vehicle capacity, and delivery priorities. The goal is to minimize costs, reduce travel time, maximize resource utilization, or optimize other relevant objectives. In the context of computer networks, routing problems involve determining the optimal path for data packets to travel from a source to a destination across a network of interconnected nodes or routers. The routing algorithm used determines the best path based on factors such as network congestion, link quality, and various routing metrics. The objective is to efficiently transmit data while avoiding network congestion, minimizing delays, and optimizing the use of available network resources. Effective solutions to routing problems require sophisticated algorithms and optimization techniques. Researchers and practitioners use various approaches, including mathematical programming, heuristics, metaheuristics, and artificial intelligence techniques to address different routing challenges. We can solve it by **mathematical** method and **computational** method.

Type of mathematical model used to solve Routing problem:

We have many type of mathematical models used to solve routing problem but we will talk about **three** type of mathematical model.

1. Traveling Salesman Problem (TSP):
   It is a problem whose goal is to figure out the shortest possible route that can be assigned to a salesman. By following this route they can visit a set of

cities and return to their starting location, covering each stop once. As the number of cities increases, finding the optimal solution becomes exponentially difficult. Given a collection of cities and the distance of travel between each pair of them, the traveling salesman problem is to find the shortest way of visiting all the cities and returning to the starting point. Though the statement is simple to state, it is more difficult to solve. Traveling Salesman Problem [1][2] is an optimization problem and has a vast search space and is said to be NP-hard, which means it cannot be solve din polynomial time. It is one of the most fundamental problems in the field of computer science in today's time. The Traveling salesman problem is being applied in many fields nowadays. Some of its applications are vehicle routing, manufacturing of microchips, packet routing in GSM, drilling in printed circuit boards etc. In simpler words, say we have a set of n number of cities, and then we can obtain (n- 1)! alternative routes for covering all the n cities. Traveling salesman problem is to procure the route which has the least distance.
So, this problem is an NP-hard problem. Manufacturing, logistics, and transportation are some of the real-world applications of TSP. The linear formulation for TSP involves binary decision variables x_{i,j}, which are equal to 1 if the salesperson travels directly from city i to city j, and 0 otherwise. The objective function minimizes the sum of the distances between all pairs of cities, subject to constraints that ensure each city is visited exactly once and that the route is a closed loop.

mathematical formulation for the Traveling Salesman Problem:
Objective function: Minimize the total distance traveled by the salesman.
Decision variables: xij = 1 if the salesman travels from city i to city j, 0 otherwise
Constraints:
Each city is visited exactly once: Σi xij = 1 for all j, $j \neq 1$ Σj xij = 1 for all i, $i \neq 1$
Subtour elimination: $\Sigma i \in S$ $\Sigma j \in S$ xij $\leq |S| - 1$ for all proper subsets S of {2, 3, ..., n}

The first set of constraints ensures that each city is visited exactly once, except for the starting city (usually labeled as city 1). The second set of constraints is the subtour elimination constraint, which prevents the formation of subtours in the solution by ensuring that the sum of xij across any subset of cities S is less than or equal to |S|- 1.

2. Capacitated Vehicle Routing Problem (CVRP):
   The CVRP is a variant of the VRP in which each vehicle has a limited capacity, and the objective is to minimize the number of vehicles used to serve all customers. The goal of CVRP is to figure out an optimal set of routes to reach each customer on time, considering the capacity of each delivery vehicle. The CVRP can be formulated as an integer programming problem with decision variables for the routes and the number of vehicles used. The capacitated vehicle routing problem (CVRP) is a classical combinatorial optimization problem, which has received much attention due to its main challenges as distribution, logistics, and transportation. This proposed attempts to find the vehicle routes with minimizing traveling distance, in which the excellent solution delivers a set of customers in one visit by capacitated vehicle. For solving the CVRP problem, a cooperative hybrid firefly algorithm (CVRP-CHFA) is proposed in this paper with multiple firefly algorithm (FA) populations. Each FA is hybridized with two types of local search (i.e., Improved 2-opt as a local search and 2-h-opt as a mutation operator) and genetic operators. The proposed algorithms (FAs) communicate from time to time for exchanging some solutions (fireflies). The main aim of the hybridization and communication strategies is to maintain the diversity of populations to prevent the proposed algorithm from falling into local optima and overcome the drawbacks of a single swarm FA. The experiments are conducted on 108 instances from eight standard benchmarks. The results revealed that the proposed CVRP-CHFA got promising results compared to other well-known methods. Moreover, the proposed CVRP-CHFA significantly outperformed the recent three hybrid firefly algorithms.

   a mathematical formulation for the Capacitated Vehicle Routing Problem (CVRP):

   Objective function: Minimize the total distance traveled by all vehicles

Decision variables: $x_{ij} = 1$ if vehicle travels from customer i to customer j, 0 otherwise $y_i = 1$ if customer i is visited by any vehicle, 0 otherwise
u_i = cumulative demand of customer i on its route.

Constraints:

1. Each customer is visited exactly once: $\Sigma_i x_{ij} = 1$ for all j, $j \neq 0$ (where $j = 0$ is the depot)
2. Each vehicle can only leave and return to the depot: $\Sigma_j x_{0j} = \Sigma_i x_{i0} = m$ (number of vehicles)
3. Vehicle capacity cannot be exceeded: $\Sigma_i u_i y_i \leq Q$ for all vehicles.
4. Subtour elimination: $\Sigma_i x_{ij} = \Sigma_i x_{ji}$ for all j, $i \neq j$ and for all $S \subseteq \{1,2, ..., n\}$, $S \neq \emptyset$, $\Sigma_{i \in S} \Sigma_{j \in S} x_{ij} \leq |S| - 1$

The first constraint ensures that each customer is visited exactly once. The second constraint ensures that each vehicle leaves and returns to the depot. The third constraint ensures that the total demand of customers on each vehicle does not exceed the vehicle capacity. The fourth constraint is the subtour elimination constraint, which prevents the formation of subtours in the solution by ensuring that the sum of x_{ij} across any subset of customers S is less than or equal to $|S| - 1$.

3. **Pickup and Delivery Problem (PDP):**

The PDP involves finding the optimal set of routes for a fleet of vehicles to pick up goods from a set of locations and deliver them to other locations, subject to constraints on vehicle capacity and route length. The PDP can be formulated as an integer programming problem with decision variables for the routes and the amount of goods transported on each route.
Just like other VRPs, PDP is also formulated as MILP and is known to be NP-hard. PDVRP has many real-life applications, and one of them is transporting goods from factories to warehouses. In most static applications, the typical objective functions to be minimized are, in first place, the fleet size needed to satisfy the demand with a determined level of service, and next, the total distance traveled by all vehicles, provided that the fleet size is fixed (Desrosiers et al., 1995). In the case of passenger movements, instead of the distance attribute, a more reasonable objective function should include both the total waiting and travel time of all passengers, combined with some measure of the operational cost for running the system, weighted differently.

The static PDP constraints can be classified as visiting constraints (each pickup and delivery have to be visited exactly once).

a mathematical formulation for the Pickup and Delivery Problem (PDP):

Objective function: Minimize the total distance traveled by all vehicles.
Decision variables: $x_{ij} = 1$ if vehicle travels from location i to location j, 0 otherwise $y_i = 1$ if location i is visited by any vehicle, 0 otherwise u_i = cumulative load picked up at location i on its route.
Constraints:

1. Each location is visited exactly once: $\Sigma_i x_{ij} = 1$ for all j, $j \neq 0$ and $j \neq n+1$ $\Sigma_j x_{ij} = 1$ for all i, $i \neq 0$ and $i \neq n+1$
2. Each vehicle can only leave and return to the depot: $\Sigma_j x_{0j} = \Sigma_i x_{in+1} = m$ (number of vehicles)
3. Vehicle capacity cannot be exceeded: $\Sigma_i u_i y_i \leq Q$ for all vehicles.
4. Consistency of loads: $\Sigma_i x_{ij} u_i = \Sigma_i x_{ji} u_i$ for all j, $i \neq j$
5. Pickup and delivery: $u_i + q_i \leq u_j$ for all i, j such that x_i is a pickup and x_j is a delivery.

The first set of constraints ensures that each location is visited exactly once, except for the depot and the final destination. The second constraint ensures that each vehicle leaves and returns to the depot. The third constraint ensures that the total load picked up at each location does not exceed the vehicle capacity. The fourth constraint is the consistency of loads constraint, which ensures that the cumulative load at any location is the same for all routes that pass through that location. The fifth constraint is the pickup and delivery constraint, which ensures that the load picked up at any pickup location is delivered to the corresponding delivery location.

The mathematical model we chose to solve routing problem.

We choose **Traveling Salesman Problem (TSP)** to solve example of routing problem. We need to formulate the problem we choose to solve.
The problem is we have the 4 cities we need to visit all city with bus only once time and the start city and the finish city is city 1, we need to solve the problem to get the shortest bath.

	A	B	C	D
A	0	10	15	20
B	10	0	35	25
C	15	35	0	30
D	20	25	30	0

Objective function:

Minimize f(x)=10x[1][2] + 15x[1][3] + 20x[1][4] + 10x[2][1] + 35x[2][3] + 25x[2][4] + 15x[3][1] + 35x[3][2] + 30x[3][4] + 20x[4][1] + 25x[4][2] + 30x[4][3]

Constraint:

Subject to:

- x[1][2] + x[1][3] + x[1][4] = 1
- x[2][1] + x[2][3] + x[2][4] = 1
- x[3][1] + x[3][2] + x[3][4] = 1
- x[4][1] + x[4][2] + x[4][3] = 1
- x[1][2] + x[2][1] + x[3][1] + x[1][3] + x[3][2] + x[2][4] + x[4][3] + x[3][4] + x[4][1] ≤ 3
- x[i][j] ∈ {0,1} for all i,j=1 to 4.

Anther example for 6 cities:

Objective function:

Minimize: 2x12 + 3x13 + 5x14 + 6x15 + 7x16 + 2x21 + 4x23 + 8x24 + 3x25 + 4x26 + 3x31 + 4x32 + 6x34 + 7x35 + 8x36 + 5x41 + 8x42 + 6x43 + 2x45 + 3x46 + 6x51 + 3x52 + 7x53 + 2x54 + 4x56 + 7x61 + 4x62 + 8x63 + 3x64 + 4x65

Constraint:

Subject to:

$x12 + x13 + x14 + x15 + x16 = 1$

$x21 + x23 + x24 + x25 + x26 = 1$

$x31 + x32 + x34 + x35 + x36 = 1$

$x41 + x42 + x43 + x45 + x46 = 1$

$x51 + x52 + x53 + x54 + x56 = 1$

$x61 + x62 + x63 + x64 + x65 = 1$

$x12 + x21 + x31 + x41 + x51 + x61 = 1$

$x13 + x23 + x32 + x42 + x52 + x62 = 1$

$x14 + x24 + x34 + x43 + x53 + x63 = 1$

$x15 + x25 + x35 + x45 + x54 + x64 = 1$

$x16 + x26 + x36 + x46 + x56 + x65 = 1$

$x[i][j] \in \{0,1\}$ for all i,j=1 to 4.

Pso algorithm used to solve routing problem:

To solve a routing problem using the PSO algorithm, the first step is to formulate the problem mathematically, defining the set of nodes (locations) and arcs (possible routes between nodes) along with associated costs or distances. Each potential solution is represented as a particle, where a particle represents a potential route or path. It consists of a sequence of nodes that define the order in which the entities (e.g., vehicles) visit the locations.

The algorithm starts by initializing a population of particles randomly. Each particle's position represents a candidate solution (a route), and its velocity represents the direction and magnitude of the particle's movement in the search space. An objective function is defined based on the specific routing problem, which evaluates the quality of a particle's position or route by considering factors such as total distance traveled, time taken, resource utilization, or any other relevant criteria.

In each iteration of the PSO algorithm, particles move through the search space to find optimal solutions. The movement is guided by adjusting their velocities based on their previous velocity, their best-known position (personal best), and the best-known position among the entire population (global best). The velocity update formula includes inertia, cognitive component, and social component, which control the balance between exploration (global search) and exploitation (local search) during the optimization process.

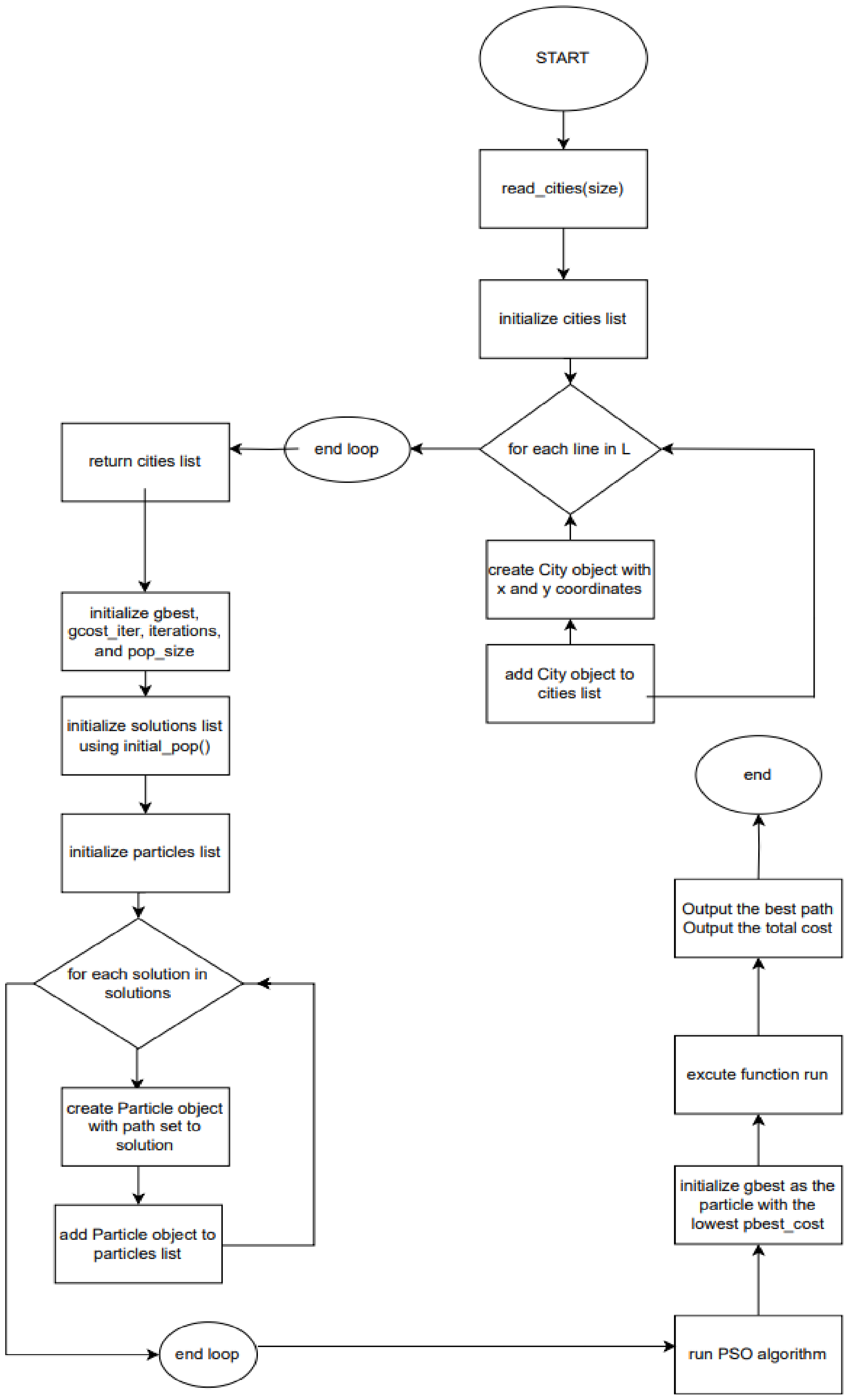
After updating the velocities, the particles' positions are updated, moving them in the search space according to their new velocities. This movement represents the exploration or exploitation of new routes based on the information available to each particle. The iteration process continues for a specified number of iterations or until a termination criterion is met.

At the end of the iterations, the best-known position (route) among all the particles is considered the solution to the routing problem. It represents the optimal or near-optimal route that satisfies the problem's constraints and optimizes the objective function. However, depending on the specific routing problem, additional post-processing steps may be required to refine the solution or convert it into a practical plan. This can involve considering real-world factors, such as traffic conditions, vehicle capacity, time windows, or any other specific constraints.

By applying the PSO algorithm to the routing problem, it efficiently searches the solution space, balancing exploration and exploitation to find high-quality routes that minimize costs or optimize the desired objectives. The algorithm's ability to leverage collective intelligence and learn from the best positions discovered by particles helps in finding better solutions. Ultimately, the PSO algorithm provides a powerful tool for solving routing problems in various domains, such as transportation, logistics, and telecommunications.

code explanation for solving routing problems by pso algorithm

flow chart


START
read_cities(size)
initialize cities list
for each line in L
create City object with x and y coordinates
add City object to cities list
end loop
return cities list
initialize gbest, gcost_iter, iterations, and pop_size
initialize solutions list using initial_pop()
initialize particles list
for each solution in solutions
create Particle object with path set to solution
add Particle object to particles list
end loop
run PSO algorithm
initialize gbest as the particle with the lowest pbest_cost
excute function run
Output the best path
Output the total cost
end

The flow chart explanation

our code starts from the main function at it we call function read cities and pass to it the specific size the function receive specific file dependent on the size that pass the file that contain the coordinate value x and y for each city the function read cities spilt the coordinates then add each city's coordinates to the cities list then return the list cities.

then we create object from the class PSO and the call the function run by these object At the run function we start by take the min p best for all particles to by the Global best Then we iterate by the number of iterations At the start to any iteration, we update the G best by take the min P best for all particles

1. Two empty lists, (x_list) and (y_list), are initialized to store the x and y coordinates of cities.

2. A loop is performed over each city in self.gbest.pbest (representing the personal best route of the current particle).

3. Within the loop, the x-coordinate of each city is extracted using city.x and appended to the x_list, while the y-coordinate is extracted using city.y and appended to the y_list. This process effectively collects the x and y coordinates of each city in the personal best route.

4. Additionally, the x-coordinate of the first city in pso.gbest.pbest (representing the global best route of the entire swarm) is appended to x_list, and the y-coordinate is appended to y_list. This ensures that the coordinates of the first city in the global best route are included in the lists.

5. Finally, self.gcost_iter, which stores the global best costs per iteration, appends the current global best route's cost (self.gbest.pbest_cost) Then we iterates over each particle in the self.particles list:

1-The particle's velocity is cleared using particle.clear_velocity().

2- then Two empty lists, Temp velocity and gbest, are created.

3- A copy of the particle's route is made into new route.

4- The code generates swap operations based on the particle's personal best route It uses a list comprehension to iterate over the range of city indices and checks if the city in new route at index i is different from the city in the particle's personal best at the same index If they differ, a swap operation (i, particle.pbest.index(new_route[i]), self.pbest_probability) is created.

5- The code generates swap operations based on the global best route.

6-then we update the Velocity.

7-the particle update to go on new route .

8- Finally, the particle's costs and personal best are updated using.

<u>parameters, operators, and constraint handling technique in code:</u>

Class (city) include. __init__ (), distance(), and __repr__().

The distance () method calculates the distance between two Cities objects. It takes.

another city as a coordinates to calculate the distance based on the differences in

x and y coordinates between the two cities.

read_cities() this function reads a text file containing city coordinates, creates

City objects for each coordinate pair, and returns a list of these City objects

representing the cities.

path_cost()this function calculates the total cost or distance of a given route by

summing up the distances between consecutive cities in the route.

Class (particle) include __init__(),

clear_velocity(),update_costs_and_pbest(),path_cost(), and __repr__().

The current_cost instance variable is set based on the cost parameter if it is

provided. Otherwise, it is set by calling the path_cost() method, which calculates

the cost of the current path.

Similarly, the pbest_cost instance variable is set based on the cost parameter if provided, otherwise by calling the path_cost() method.

The clear_velocity() method clears the velocity information of the particle. It empties the velocity list.

The update_costs_and_pbest() method updates the current cost and personal best (pbest) information of the particle. It first calculates the current cost by calling the path_cost() method. If the current cost is lower than the current pbest

cost, it updates both the self.pbest and self.pbest_cost to reflect the new best path and cost. The path_cost() method calculates and returns the cost of the current path by calling the path_cost() function.

Class(pso) include __init__(), initial_pop, random_route , greedy_route, and __repr__().

iterations: The number of iterations or generations the algorithm will run.

pop_size: The size of the population, i.e., the number of particles.

gbest_probability: The probability of selecting the global best path for a particle.

pbest_probability: The probability of selecting the personal best path for a particle.

cities: The list of City objects representing the cities in the optimization problem.

The random_route() method generates a random route (path) by shuffling the list

of cities. It returns the shuffled route.

The initial_pop() method generates the initial population of solutions (paths) for the particles. It creates a random_population list by repeatedly calling the random_route() method and appending the generated random routes to the list.

It also generates a single greedy route using the greedy_route() method starting

from the first city.

The random_population and the greedy route are combined and returned as the

initial population. The greedy_route() method generates a greedy route (path)

starting from a specified start_node index.

The run method is a part of a larger class that encompasses a Particle Swarm

Optimization (PSO) algorithm.

Present the results of your implementation:

5 cities -> (1,1) , (6,4) , (5,7) , (-1,2) , (3 , 4)

Result -> The total cost: 19.306984762977038

best route: [(3.0, 4.0), (5.0, 7.0), (6.0, 4.0), (1.0, 1.0), (-1.0, 2.0)]

10 cities -> (4 ,7) , (2,3) , (6, 6), (5, 2), (8 ,4) , (6 ,1) , (0 , 2) , (9 ,12) , (1 ,8), (3 ,1)

Result -> The total cost: 37.52317449587724

best route: [(9.0, 12.0), (6.0, 6.0), (8.0, 4.0), (6.0, 1.0), (5.0, 2.0), (3.0, 1.0), (0.0, 2.0), (2.0, 3.0),(1.0, 8.0), (4.0, 7.0)]

15 cities -> (4 ,7) , (2,3) , (6, 6), (5, 2), (8 ,4) , (6 ,1) , (0 , 2) , (9 ,12) , (1 ,8), (3 ,1) , (1,1) , (6,4) , (5,7) , (-1,2) , (3 , 4)

Result -> The total cost: 43.14822683948876

best route: [(4.0, 7.0), (5.0, 7.0), (6.0, 6.0), (6.0, 4.0), (8.0, 4.0), (6.0, 1.0), (5.0, 2.0), (3.0, 1.0), (1.0, 1.0), (0.0, 2.0), (-1.0, 2.0), (2.0, 3.0), (3.0, 4.0), (1.0, 8.0), (9.0, 12.0)]

20 cities -> (4 ,7) , (2,3) , (6, 6), (5, 2), (8 ,4) , (6 ,1) , (0 , 2) , (9 ,12) , (1 ,8), (3 ,1) , (1,1) , (6,4) , (5,7) , (-1,2) , (0 ,0) , (2 ,12) , (5 ,10) , (3, 3) , (8 ,7) , (10 ,10)

Result -> The total cost: 54.80192233114787

best route: [(4.0, 7.0), (5.0, 7.0), (6.0, 6.0), (6.0, 4.0), (8.0, 4.0), (8.0, 7.0), (10.0, 10.0), (9.0, 12.0), (5.0, 10.0), (2.0, 12.0), (1.0, 8.0), (2.0, 3.0), (3.0, 3.0), (3.0, 1.0), (1.0, 1.0), (0.0, 2.0), (-1.0, 2.0), (0.0, 0.0), (6.0, 1.0), (5.0, 2.0)] references

1. https://www.researchgate.net/publication/341371861_Literature_Review_on_Travelling_Salesman_Problem
2. https://www.sciencedirect.com/science/article/abs/pii/S1568494621003264
3. https://www.sciencedirect.com/science/article/abs/pii/S0377221709000356
4. https://link.springer.com/referenceworkentry/10.1007/978-3-319-07124-4_42
5. https://arxiv.org/pdf/1606.01935.pdf
6. https://math.stackexchange.com/questions/1957211/linear-programming-formulation-of-traveling-salesman-tsp-in-wikipedia
7. https://www.upperinc.com/blog/vehicle-routing-problem/