CS512: Travelling Salesman Problem - Fall 2018

Vedant Choudhary
Rutgers University
Piscataway, NJ, USA
Email: vc389@rutgers.edu

Zhengjuan Fan
Rutgers University
Piscataway, NJ, USA
Email: zf67@rutgers.edu

Sanjana Kumar
Rutgers University
Piscataway, NJ, USA
Email: sk2053@rutgers.edu

***Abstract*—** Travelling salesman problem (TSP) is a widely known NP-hard problem. It asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?". In this project, we try to provide a solution to TSP through a meta-heuristic technique called Ant Colony Optimization (ACO).

I. Project Description

Travelling Salesman Problem (TSP) is a NP-hard problem so there is no known algorithm that will solve it in polynomial time. But, at the same time there are many real world applications of TSP and hence, a good optimized solution can be useful. We aim at arriving at an optimized solution through a meta-heuristic technique called Ant Colony Optimization (ACO). The input for the algorithm is a graph with three parameters - V: vertices, E: edges, W: weight of the edge (which can be distance, time, cost etc.)

The project has four stages: Gathering, Design, Infrastructure Implementation, and User Interface.

A. Stage1 - The Requirement Gathering Stage.

- We intend to design a system which is able to implement an optimized solution to the Travelling Salesman Problem through Ant Colony Optimization. The system will support the ability to implement TSP by using a pre-built data or if the user wants to customize the graph structure, we intend to provide a functionality for that too. The aim of the system is to reduce the amount of time spent in planning, scheduling, logistics and packing problems which involve paths and additional constraints which may be imposed such as limited resources or time windows
- The three types of users (grouped by their data access/update rights): There are three types of users: A trip planner, A PCB Designer, and An owner of a fuel distribution company
- A trip planner's interaction modes: For a trip planner, he/she is interested in covering as many locations in shortest time possible to have the most out of that trip. On getting to know the number of cities to be travelled (based on distance or time), the system will find out the shortest distance or time path between the cities. It is assumed that each city is traversed only once.
- The real world scenarios:
  - Scenario1 description: Bob is planning to take a road trip across the U.S by car. The trip must make only one stop at a popular city in some given states (ex: 48) of The U.S. He could use our algorithm to compute the shortest route for his trip.
  - System Data Input for Scenario1: 48 popular cities can be from TripAdvisor-rated best city to visit in U.S. The distance between each pair of cities by car.
  - Input Data Types for Scenario1: A Graph = (V, E, W), V is a set of 48 popular cities, E(i,j) is edge connecting two cities, and W(i,j) gives the distance from city i to city j.
  - System Data Output for Scenario1: A shortest route of 48 U.S. cities
  - Output Data Types for Scenario1: A Linked List or an Array
  - Scenario2 description: Bob is planning to take a trip across the U.S by flight. He could use our algorithm to compute the fastest route for his trip.
  - System Data Input for Scenario2: 48 U.S. airports, the flight duration between each pair of airports
  - Input Data Types for Scenario2: A Graph = (V, E, W), V is a set of 48 U.S. airports, E(i,j) is edge connecting two airports, and W(i, j) gives the flight duration from airport i to airport j.
  - System Data Output for Scenario2: A fastest route of 48 U.S. airports
  - Output Data Types for Scenario2: A Linked List or an Array
- A PCB designer's interaction modes: Printed Circuit Board (PCB) designing and manufacturing depends on the holes drilling time, which is a function of the number of holes and the order in which they are drilled. A typical PCB may have hundreds of holes and optimizing the time to complete the drilling plays a role in the production rate. Even a small increase in efficiency of drilling holes leads to huge time saving in mass production of those PCBs.
- The real world scenarios:
  - Scenario1 description: Mike is a PCB designer and has been asked to code how the drilling machine should move around the PCB so that there is a significant reduction in production time. He knows that the last time, he randomly used different drill sizes for holes. But, this time, he has an added knowledge of our algorithm and can use it to compute the fastest way drilling job can be done.

  - System Data Input for Scenario1: Number of conductive layers and their interlinked paths, along with the sizes of drills to be made on those layers
  - Input Data Types for Scenario1: A Graph = (V, E, W). V is a set of no. of holes to be drilled, E is edges V = (i,j) connecting one conductive layer to another, and W is the size of drilling required.
  - System Data Output for Scenario1: A drilling sequence of all the conductive layers
  - Output Data Types for Scenario1: A Linked List or an Array
  - Scenario2 description: Mike has now been asked to implement a new design of the PCB. Now, he knows the idea of our algorithm and can use it for assigning optimized paths between various components of a PCB.
  - System Data Input for Scenario2: Number of components of a PCB, their interlinked paths and the weights of those paths
  - Input Data Types for Scenario2: A Graph = (V, E, W). V is a no. of components in PCB, E is edges V = (i,j) connecting the components of PCB, and W is the path distance between components of PCB.
  - System Data Output for Scenario2: An optimized routing/sequence of connected components
  - Output Data Types for Scenario2: A Linked List or an Array

- An owner of a fuel distribution company's interaction modes: Joe is the owner of a fuel oil distribution company in New York. He uses a fleet of 10 trucks to serve his 50 customers from its main depot to the customers' stores. He could use our algorithm to minimizing the total travel time. The customers are selected on a cost minimization criterion and routes are constructed by matching capacity and time constraints. Constructive methods used are two-phase methods and can be divided into two classes: cluster first, route-second methods and route-first, cluster-second methods.
- The real world scenarios:
  - Scenario1 description: In cluster first, route-second method, customers are first organized into feasible clusters, and a vehicle route is constructed for each of them. The idea is to insert the nearest neighbour of the last inserted customer in the route. The first inserted customer on the route can be selected randomly or with some arbitrary criteria like the farthest distance customer from the depot. From this seed route, every other customer is inserted by the criteria of the nearest neighbour from the last inserted customer until the capacity of the vehicle is exhausted. This method is derived from TSP approach.
  - System Data Input for Scenario1: 50 stores, the distance between each pair of store.
  - Input Data Types for Scenario1: Input Data Types for Scenario 1: A Graph = (V, E, W), V is a set of 50 stores in U.S. E is edges V = (i,j) connecting two stores and W(i, j) gives the distance from store i to store j.
  - System Data Output for Scenario1: System Data Output for Scenario 1: Minimizing the total travel time of the trucks.
  - Output Data Types for Scenario1: A Linked List or an Array
  - Scenario2 description: In the route-first, cluster-second method, a tour is first built on all customers and then segmented into feasible vehicle routes.
  - System Data Input for Scenario2: 50 stores, the best route between each pair of store.
  - Input Data Types for Scenario2: A Graph = (V, E, W), V is a set of 50 stores in U.S with the best route. E is edges V = (i,j) connecting two stores and W(i, j) gives the distance from store i to store j.
  - System Data Output for Scenario2: Minimal costs of the travelled routes.
  - Output Data Types for Scenario2: A Linked List or an Array
- Project Time line and Division of Labor.
- Project Time line
  - 10/23 - 10/27 : Conceptualizing the problem statement and gathering knowledge for Stage 1
  - 10/25 - 11/01 : Data collection and generation
  - 11/02 - 11/09 : Framing the problem statement and designing the system flow diagram of ACO
  - 11/10 - 11/18 : Developing ACO network for TSP
  - 11/19 - 11/24 : Visualization of the solution
- Division of Labor Time line
  - Vedant Choudhary : Conceptualization, Documentation, System Flow Diagram, Implementation of ACO algorithm, User-interface, Visualization
  - Zhengjuan Fan : Conceptualization, Documentation, System Flow Diagram, Data collection, Implementation of ACO algorithm, Analysis of TSP
  - Sanjana Kumar : Conceptualization, Documentation, Data collection, System Flow Diagram, Implementation of ACO algorithm, Visualization

B. Stage2 - The Design Stage.

- Project Description.
  - In this system, we aim at arriving at an optimized solution through Ant Colony Optimization. We start by gathering a dataset which contains coordinates of cities that are to be visited
  - Next, we initialize the ACO parameters and ask the user to input the first city (start of travel) to the last city (end of travel)
  - The ants in ACO algorithm then use euclidean distance to find the desirability and probability of visiting other nodes
  - Once the exit conditions/optimal solution is met, the program exits, outputting the best possible path

- Flow Diagram.

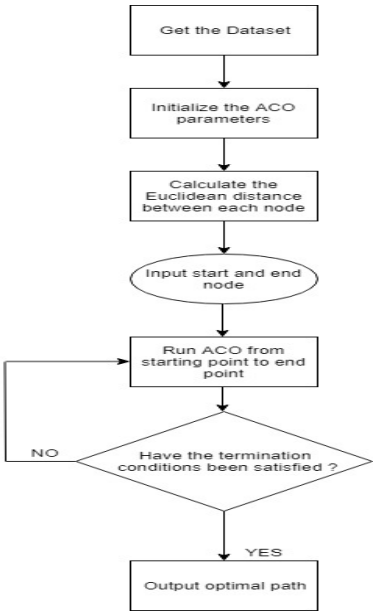

Fig. 1. A general ACO algorithm flow

- High Level Pseudo Code System Description.
  - Ant Colony Optimization Technique
    1) Populate a random starting node (ex- city, drilling point, address) with colony of ants
    2) For each ant
       a) Send the ant to all cities such that they visit each city once
       b) Initially, visiting cities is random
       c) After few runs, the next city selection gets influenced by the amount of phermone left on the trail according to Eq. (1)
    3) Once all ants have completed a tour of the world, deposit pheromone on each city path. The amount of pheromone deposited by each ant is proportional to the length of the path traveled by the given ant.

    At each city, a probability distribution for next city selection from a list of possibilities is generated according to the following:

$$p_{xy}^{k} = \frac{\Gamma_{xy}^{\alpha} + \eta_{xy}^{\beta}}{\sum_{z \epsilon Y_i} \Gamma_{xz}^{\alpha} + \eta_{xz}^{\beta}} \qquad (1)$$

    Where:
    * x is the current node (city)
    * y is an unvisited node (city)
    * Γ is the amount of phermones deposited on the trail(path) between x and y nodes
    * η is the desirability of node y with respect to node x
    * α is a weighting factor for phermones
    * β is a weighting factor for desirability

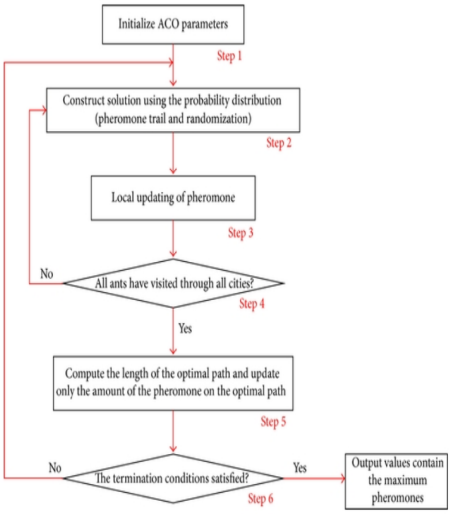

Fig. 2. A general ACO algorithm flow

    * z is a list of unexplored cities from x
  - Euclidean Distance Technique
    1) Given - the coordinates of every node
    2) Distance between them will be calculated by the formula:

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \qquad (2)$$

    Where:
    * d_{ij} is distance between i and j node(city)
    * x and y are x-y coordinates for nodes
- Algorithms and Data Structures.
  - Ant Colony Optimization

    Ant colony optimization is a meta-heuristic technique which has been applied to solve many combinatorial optimization problems. It is known that ant colony optimization gives near-optimal solution to the problem of our project - Travelling Salesman Problem. The algorithm is based on a set of ants. At each stage in the algorithm, the ant chooses to move from one city to another according to some properties mentioned below:
    1) It must visit each city exactly once
    2) A city farther away from ant's position has less chance of being chosen (less desirability)
    3) More intense pheromone trail on an edge between two cities, means greater the probability of that edge being chosen

    A graph G = (V, E, W) has to be an input for the algorithm, along with a starting node (or a random starting node)
  - Euclidean Distance

    Euclidean Distance is a popular distance calculation metric used in mathematics and computer science. It works on a simple principle.

Given, you have a pair of nodes, along with their coordinates, between which you need to calculate the distance, the euclidean distance formula can be applied according to Eq. (2).
Since the input for the algorithm is just a pair of nodes, any simple data structure can be used. For example - a linked list pointing from city u to city v, the value of city v will be the distance to reach it.

- Time Complexity.
  - The exact solution for TSP is achieved by brute force search by trying all ordered permutations and finding the cheapest one. But the running time is $O(n!)$. ACO provides the approximate solution for TSP. Convergence results show that as the time goes to infinity, the probability of find the optimum tends to 1.
  - To analyze the time complexity of ACO for TSP, we referred two papers from [1] and [2]. The finite-time dynamics of ACO is analyzed using asymptotic bounds on the expected time:
    1) ACO starts with an equal amount of pheromone on all edges. All values of Γ_{xy} sum up to 1. In other words, the probability of choosing an edge depends on both the pheromones and the heuristic information η.
    2) The next iteration state typically depends on many previous iterations. Because pheromone traces evaporate slowly and their impact on pheromones last for a much longer period of time.
    3) The TSP in our project is modeled as a complete graph. Thus, the expected run time is $O(n^6 + (1/\rho)n \ln n)$ $(\alpha = 1, \beta = 0)$; $O(n^5 + (1/\rho)n \ln n)$ $(\alpha = 1, \beta = 1)$. ρ denotes the pheromone evaporation ratio
- Space Complexity. att48.tsp (downloaded from TSPLIB) contains 48 capitals of the US: the first column is the index of cities, the rest two columns represent city's x-y coordinates. For simplicity, a systemic TSP problem will be solved in this project. In addition, Γ_{xy}, η_{xy} need to be stored in $n \times n$ matrices. The shortest path is stored in a list of length O(n). Thus, the overall space requirement is O(n^2).
- Constraints.
  - The dataset should at least have three columns: nodes (can be cities, addresses, drilling points etc.), their x and y coordinates
  - Without coordinates, the euclidean distance algorithm fails to return a distance between nodes, emphasizing the above point.
  - Since each ant traverses the graph, there is an upper limit of ants used for this parameter. User should not exceed this number.
  - Since ACO is an optimization technique, a very large input of nodes will lead to a large time in converging the model, therefore, if the user generates a dataset, it is advised to keep nodes less than 100

C. *Stage3 - The Implementation Stage.*

- Language and Programming Environment: The codebase uses Python language along with some of its libraries such as numpy, matplotlib for visualization. The programming environment is linux terminal with code written in Atom Text Editor, which provides some IDE capabilities.
- Sample small data snippet: There are 3 columns in the dataset, city index, its x coordinate, and its y coordinate

```
1 6734 1453
2 2233 10
3 5530 1424
4 401 841
5 3082 1644
6 7608 4458
7 7573 3716
8 7265 1268
9 6898 1885
10 1112 2049
```

Fig. 3. Data Input

- Sample small output: The output represent the shortest route of visiting all the 48 cities exactly once from a random starting city.

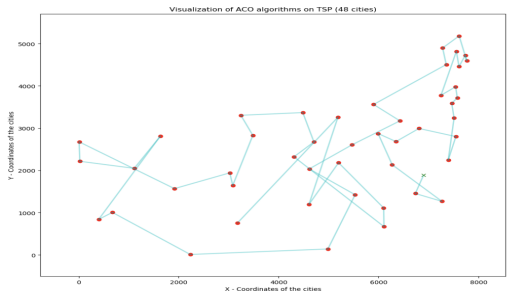

Fig. 4. Visualization

- Working code

```
# Parameters used -
# index - stores the city number
# x - x coordinate for the city i
# y - y coordinate for the city i
class City:
    def __init__(self,i=0, x_cord=0, y_cord=0):
        self.index = i
        self.x = x_cord
        self.y = y_cord

# Parameters used -
# num_cities - number of cities for travel
# initial_pheromone - initial value of pheromone deposited
# alpha - weight parameter for pheromones
# beta - weight parameter for desirability (attractiveness), which is inverse of distance
# pheromone_deposit - amount of pheromones which can be deposited
# evaporation_constant - amount of pheromone which will evaporate after a cycle
class ACO:
    def __init__(self, num_cities, initial_pheromone=1, alpha=1, beta=3,
                 pheromone_deposit=1, evaporation_constant=0.6):
        self.cities = []
        self.shortest_paths = []
        self.shortest_paths_lens = []
        self.shortest_path_len = -1
        self.evaporationConst = evaporation_constant
        self.pheromone_deposit = pheromone_deposit
        self.pheromone = numpy.full((num_cities,num_cities),initial_pheromone)
        self.alpha = alpha
        self.beta = beta
        self.attractiveness = numpy.zeros((num_cities,num_cities))
        self.routing_table = numpy.full((num_cities,num_cities),(1.00/(num_cities-1)))
```

Fig. 5. Code Snippet 1

Fig. 6. Code Snippet 2

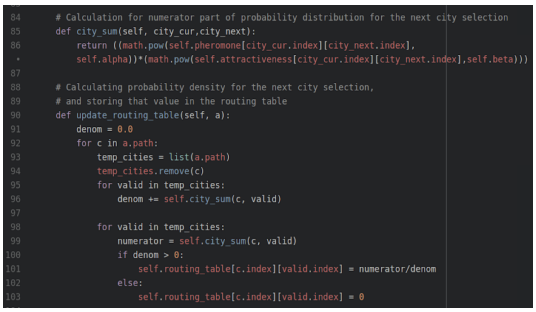

Fig. 7. Code Snippet 3

- Demo and sample findings
  - Data Size The code, when running take up 53 percent of RAM, where the RAM in our personal laptop is 8GB. RAM consumption varies on the values of parameters assigned in the code. Since, we use a small .txt file for data input, the disk resident space used is under 5 KB. Further, since no use of internet is done in our project, there is no streaming used.
  - In this project, we have found a method of implementing Traveling Salesman Problem algorithm in an optimized way by using Ant Colony Optimization algorithm. The Ant Colony optimization has reduced the time complexity of the TSP algorithm from O(n!) to polynomial time. This is a very useful application as TSP can be extended or modified in several ways. It can be used in all vehicle routing problems and for other situations where the shortest path between the nodes has to be found such as DNA fragments, soldering points etc. TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources. Further, we were able to replicate a real life travelling mechanism followed by ants to our code, using parameters to mimic how ants search for food in real life.

D. Stage4 - User Interface.

Please insert your deliverables for Stage4 as follows:

- The initial statement to activate your application with the corresponding initial UI screenshot: To activate the application, the user has to go to the folder where all the source code is kept, and then has to initialize the run statement for the program on terminal, i.e. paco_gui.py.

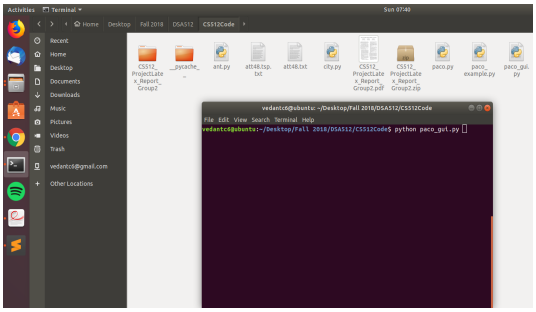

Fig. 8. Acivating the application

After that, user has to input the number of ants and the number of iterations i.e. the number of times the user wants the code to run in order to generate the shortest possible path that visits all the 48 cities exactly once.

- Two different sample navigation user paths through the data exemplifying the different modes of interaction and the corresponding screenshots: The user interacts with textual queries. We have tried to keep it as simple for the user as possible. The user has to input textual data (integers) in two textboxes - number of ants and number of iterations. After entering the values, user can see the iterations of the program in terminal itself, and the output is visualized in another box.

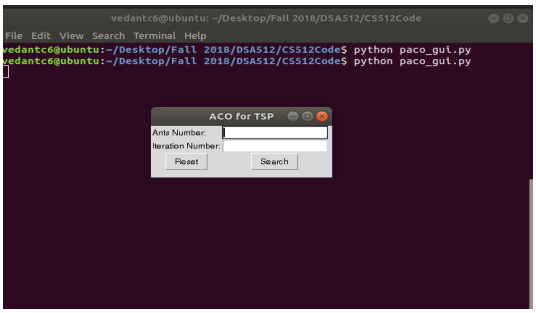

Fig. 9. UI Screen

- The error messages popping-up when users access and/or updates are denied (along with explanations and examples):
  - The error message: If the user enters text of data type other than integer or leaves the textbook blank then an error message is displayed asking the user to enter an integer value.
  - The error message explanation (upon which violation it takes place): Values other than integers for ants and interations are not possible/does not make sense, hence, the user is advised not to write numbers like 2.5 ants or iterations to avoid error messages.
  - The error message example according to user(s) scenario(s):
- The information messages or results that pop-up in response to user interface events.

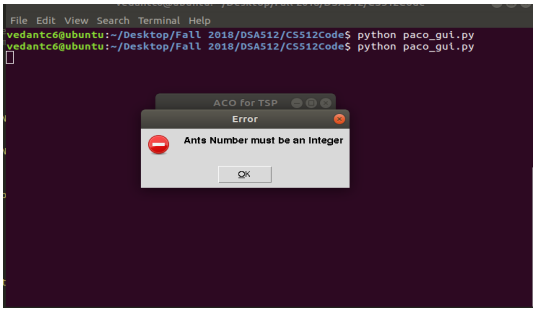

Fig. 10. Error 1

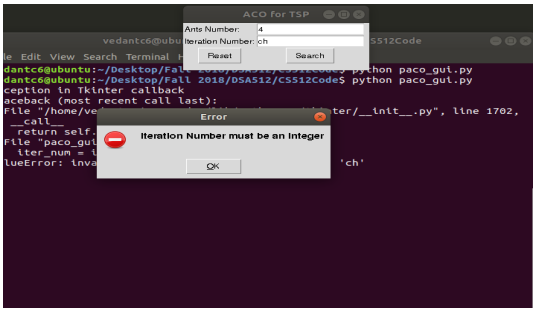

Fig. 11. Error 2

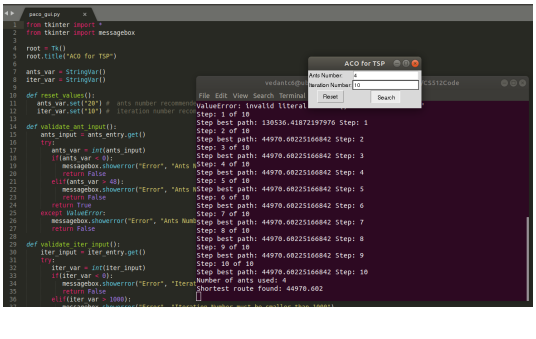

Fig. 12. When user violates no constraints

- The information message explanation and the corresponding event trigger A right set of numbers triggers the code paco_gui.py successfully, resulting in showing the output and visualization. A wrong set of numbers leads to error messages by the program.
- The error message example in response to data range constraints and the coresponding user's scenario: The user is asked to input a positive value for ants, but less than 48. Similarly, a capacity constraint is put on number of iterations such that the program runs in good time and does not use whole of computing power. The user is reminded to put a valid number whenever range constraint or data type constraint is violated. The user can also reset the variables to 20 and 10, if needed (will not violate any constraint).

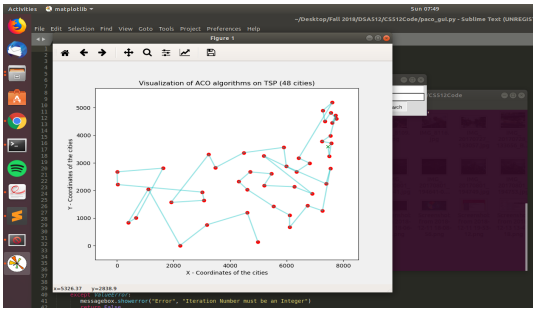

Fig. 13. Output

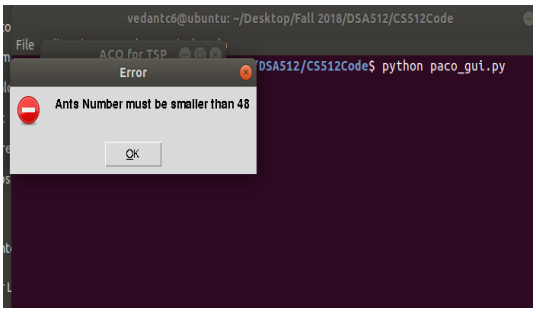

Fig. 14. Range constraint caution

- The interface mechanisms that activate different views.
  - The interface mechanism: We have tried to keep the program as simple for the user as possible, the user has to enter the numbers according to program specified above, failing which will lead to error dialog boxes, which can be closed to again input a valid set of numbers. The user does not have to interact further than entering those values, the program will be able to run based on those two values only.

REFERENCES

[1] Y. Zhou, "Runtime analysis of an ant colony optimization algorithm for tsp instances," *IEEE Transactions on Evolutionary Computation*, vol. 13, pp. 1083–1092, 2009.

[2] T. KÃ¶tzing, F. Neumann, H. RÃ¶glin, and C. Witt, "Theoretical analysis of two aco approaches for the traveling salesman problem," *Swarm Intelligence*, vol. 6, pp. 1–21, 03 2012.