TSP using Ant Colony Optimization

Group 2
Vedant Choudhary - Rutgers University, New Brunswick
Zhengjuan Fan - Rutgers University, New Brunswick
Sanjana Kumar - Rutgers University, New Brunswick

Project Goal

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).

Overview

- Travelling Salesman Problem
- Ant Colony Optimization
- Flow Diagram
- Code Snippets (commented)
- User Interface
- Types of errors
- Output
- Future Work
- Acknowledgements
- Reference/Sources

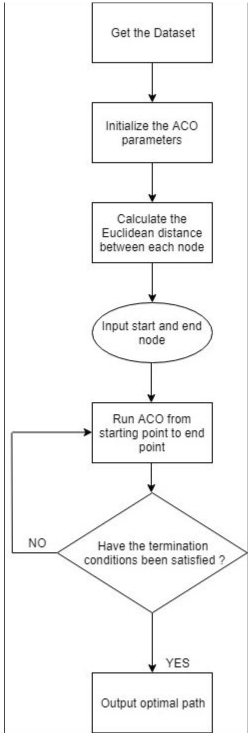
Travelling Salesman Problem

- Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city?
- NP-hard problem
- No solution which can solve it in polynomial time

Ant Colony Optimization

- Probabilistic technique for solving problems which can be reduced to finding good paths through graphs.
- Optimization technique
- Inspired by behavior of real ants

Flow Diagram



Code Snippets

```
# 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)))
```

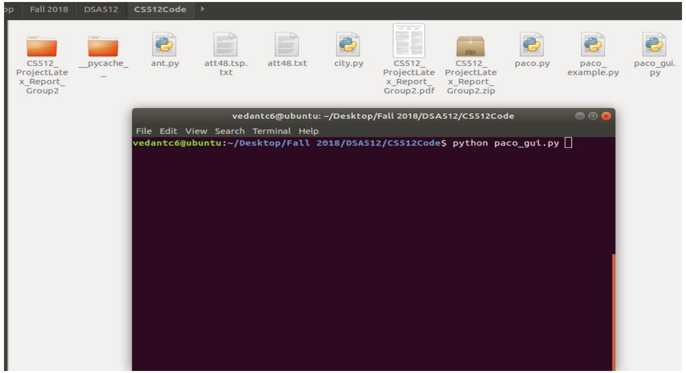
Code Snippets

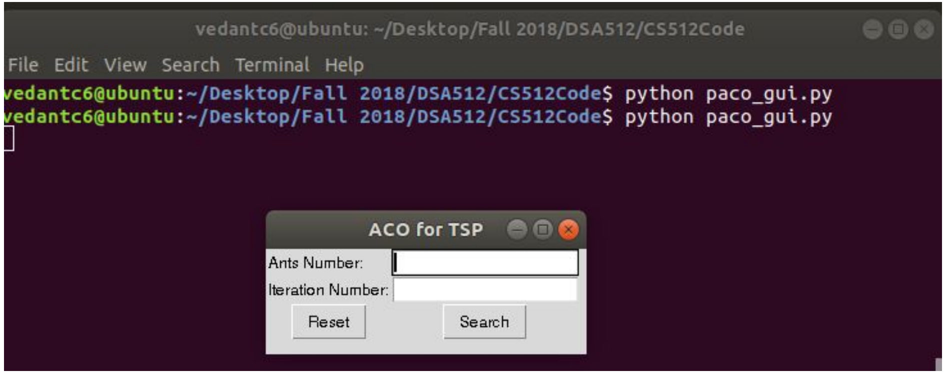
```python
    # Use euclidean formula to calculate distance between two cities.
    # Cities class has x and y coordinates
    def euclidean_distance(self,city1,city2):
        return  (math.sqrt(math.pow((city1.x-city2.x),2)+math.pow((city1.y-city2.y),2)))

    # Attractiveness is reciprocal of distance. This function calculates
    # attractiveness between cities.
    def calc_attraction(self):
        city_list=self.cities
        for i in range(len(city_list)):
            for j in range(len(city_list)):
                distance = self.euclidean_distance(city_list[i], city_list[j])
                if distance > 0:
                    self.attractiveness[i][j] = 1/distance
                else:
                    self.attractiveness[i][j] = 0.00

    # a - a single ant
    # pheromone deposit on each path is updated by the ant visiting that path
    # we have deposited the phermone uniformly across the path
    # additionally, evaporation part is removed from the pheromone
    def update_pheromone(self,a):
        for i in range(0,len(a.path)-1):
            try:
                curr_pher = self.pheromone[a.path[i].index][a.path[i+1].index]
                self.pheromone[a.path[i].index][a.path[i+1].index] = curr_pher + self.pheromone_deposit/a.path_length
            except:
                break
        self.pheromone = self.pheromone*(1-self.evaporationConst)

    # get the pheromone value for the path between city i and j
    def get_pheromone(self,i,j):
        return self.pheromone[i][j]
```

Code Snippets

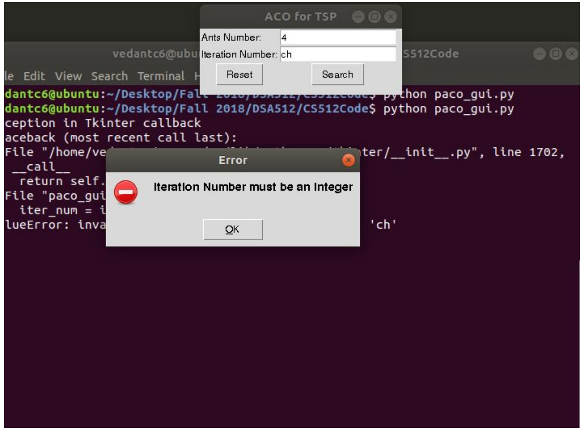
```
    # Calculation for numerator part of probability distribution for the next city selection
    def city_sum(self, city_cur,city_next):
        return ((math.pow(self.pheromone[city_cur.index][city_next.index],
        self.alpha))*(math.pow(self.attractiveness[city_cur.index][city_next.index],self.beta)))

    # Calculating probability density for the next city selection,
    # and storing that value in the routing table
    def update_routing_table(self, a):
        denom = 0.0
        for c in a.path:
            temp_cities = list(a.path)
            temp_cities.remove(c)
            for valid in temp_cities:
                denom += self.city_sum(c, valid)

            for valid in temp_cities:
                numerator = self.city_sum(c, valid)
                if denom > 0:
                    self.routing_table[c.index][valid.index] = numerator/denom
                else:
                    self.routing_table[c.index][valid.index] = 0
```

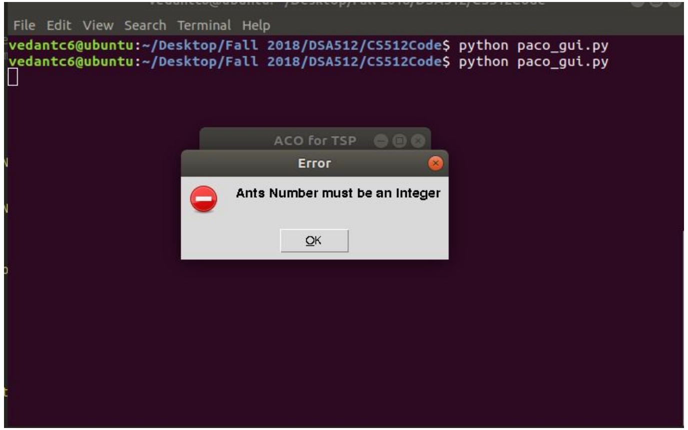
User Interface

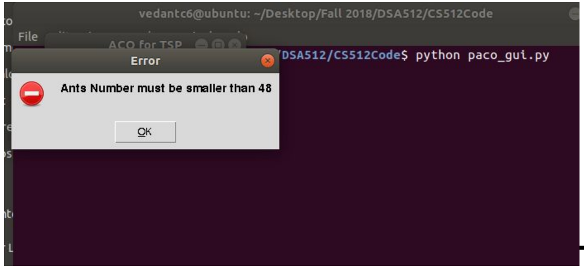




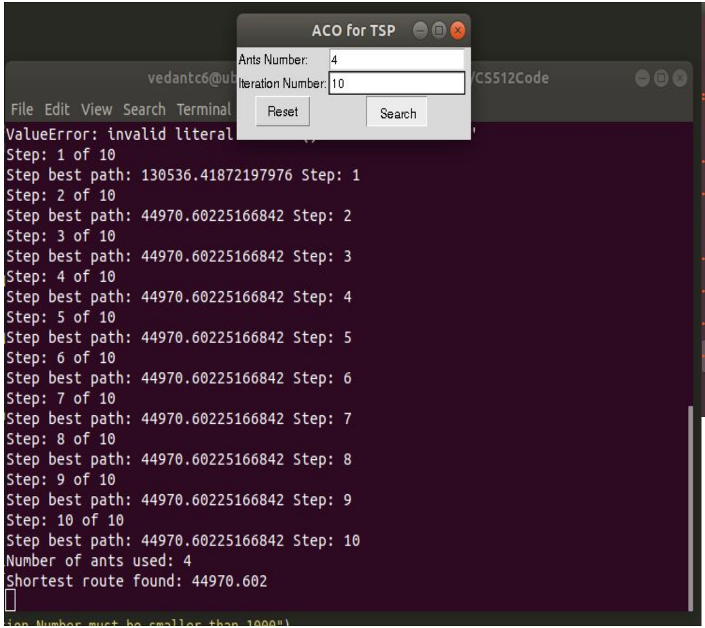
Types of Errors







Output


ACO for TSP
Ants Number: 4
Iteration Number: 10
Reset
Search
File Edit View Search Terminal
ValueError: invalid literal
Step: 1 of 10
Step best path: 130536.41872197976 Step: 1
Step: 2 of 10
Step best path: 44970.60225166842 Step: 2
Step: 3 of 10
Step best path: 44970.60225166842 Step: 3
Step: 4 of 10
Step best path: 44970.60225166842 Step: 4
Step: 5 of 10
Step best path: 44970.60225166842 Step: 5
Step: 6 of 10
Step best path: 44970.60225166842 Step: 6
Step: 7 of 10
Step best path: 44970.60225166842 Step: 7
Step: 8 of 10
Step best path: 44970.60225166842 Step: 8
Step: 9 of 10
Step best path: 44970.60225166842 Step: 9
Step: 10 of 10
Step best path: 44970.60225166842 Step: 10
Number of ants used: 4
Shortest route found: 44970.602

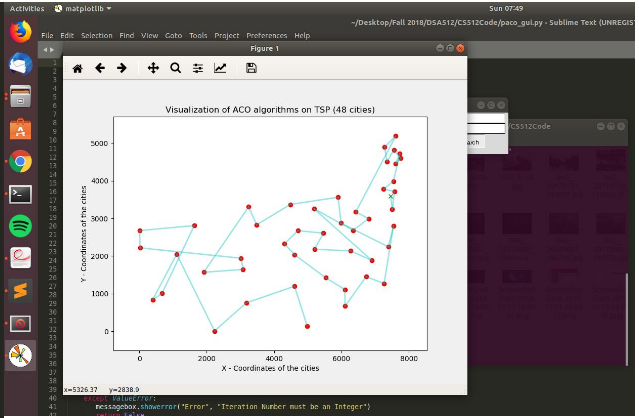


Figure 1
Visualization of ACO algorithms on TSP (48 cities)
Y - Coordinates of the cities
X - Coordinates of the cities
0
2000
4000
6000
8000
1000
3000
5000
x=5326.37 y=2838.9
messagebox.showerror("Error", "Iteration Number must be an Integer")

Future Work

Try to make UI more attractive by tkinter.ttk

Try to ask the user the number of cities the user wants to see, so number of cities also a parameter in UI

Try to allow the user to load tsp instance with the pre-predefined format

Acknowledgements

We would like to thank Professor Antonio Miranda Garcia , Harsha Srimath Tirumala for supporting us and giving useful insights throughout the development of this project.

Reference

[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. Kotzing, F. Neumann, H. Roglin, and C. Witt, “Theoretical analysis of two aco approaches for the traveling salesman problem,” Swarm Intelligence , vol. 6, pp. 1–21, 03 2012.

https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

Resource

Python3.6

Tkinter

http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/