Michał Wilkosz

Ant Systems – Report

1. Introduction

Presented solution was developed in Python language with external libraries such as Matplotlib(for drawing plots) and Numpy (for some calculations). In connection with the necessity to contain used libraries, .exe file size exceed the allowable capacity which is possible to send via mail. Therefore, we attach you .py file which is executable with Python - Anaconda interpreter.

The uploaded file is a solution for two tasks, the second one uses data set 3 for the following date of birth - 30.05.1996.

2. Source code

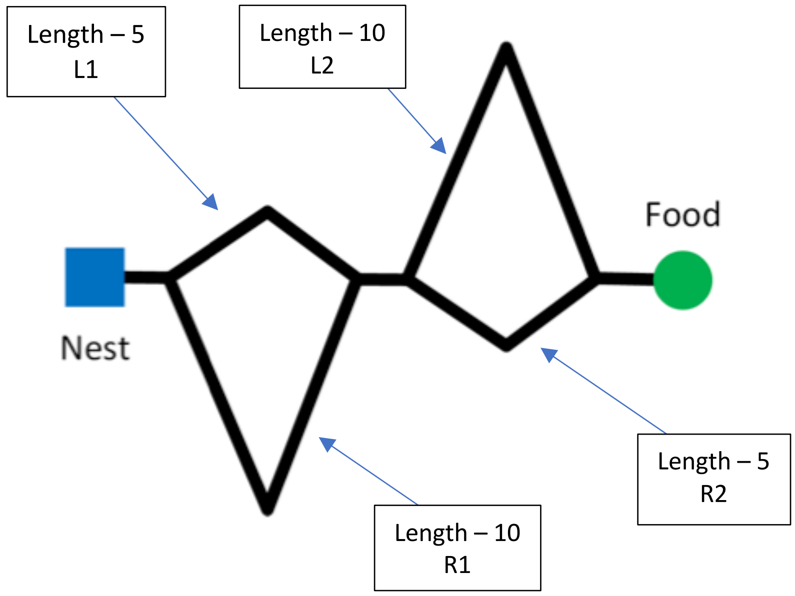
```
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
import matplotlib.pyplot as plt

def ant_colony_simultion():
    k = 20
    m = 1000
    n = 2

    d = [5,10,10,5]

    R1 = 0
    L1 = 0
    R2 = 0
    L2 = 0

    ants_routes = np.zeros((m,n))
    pheromone = np.ones((4,1))
    pheromone = 0.25*pheromone
    pheromone = pheromone.ravel()

    for i in range(m):

            decisions = np.random.choice(d,2,p = pheromone)
            #Each ant choses route by amount of pheromone.

            if decisions[0] == d[0]:
                pheromone[0] = pheromone[0] + k/(m*d[0])
                L1 = L1 + 1
            if decisions[0] == d[1]:
                pheromone[1] = pheromone[1] + k/(m*d[1])
                R1 = R1 + 1
            if decisions[1] == d[2]:
```

```
                pheromone[2] = pheromone[2] + k/(m*d[2])
                L2 = L2 + 1
            if decisions[1] == d[3]:
                pheromone[3] = pheromone[3] + k/(m*d[3])
                R2 = R2 + 1
            #Amount of pheromone left on each arc depends on the length of the route.

            pheromone = pheromone/pheromone.sum()
            #Calculation of probability for next iteration.

    print("Ants decisions in first intersection  L1: {}, R1: {}".format(L1,R1))
    print("Ants decisions in second intersection  L2: {}, R2: {}".format(L2,R2))
    print("Amount of pheromone on L1: {}, R1: {}, L2: {}, R2: 
{}".format(pheromone[0],pheromone[1],pheromone[2],pheromone[3]))

def ant_colony_tsp(m,n, Tmax, e, alpha, beta, d):

    pheromone = 0.3 * np.ones((m, n))
    #Before main loop starts small amount of pheromone is dposed on all arcs.
    routes = np.ones((m, n))
    #Routes for each of the ants.

    for t in range(Tmax):
        routes[:, 0] = 0
        #Each of ants start from the city 0.
        for i in range(m):
            nij = 1 / d
            nij[np.isinf(nij)] = 0
            for j in range(n-1):
            #N choices of each ant
                current_city = int(routes[i, j])
                #The city where the ant is currently located.
                nij[:, current_city] = 0
                #Current city is marked by 0.
                alpha_base = np.power(pheromone[current_city, :], alpha)
                beta_base = np.power(nij[current_city, :], beta)
                alpha_base = alpha_base[:, np.newaxis]
                beta_base = beta_base[:, np.newaxis]
                #Ant decision coefficients.
                aij = np.multiply(alpha_base, beta_base)
                sum_aij = np.sum(aij)
                probabilities = aij / sum_aij
                #Probabilities to chose next city.
                vector = probabilities.ravel()
                cities = np.arange(0,10)
                next_city = np.random.choice(list(cities),1,p = vector)
                #Chosing next city via roulette wheel selection to avoid falling into the 
local minimum.
                routes[i, j+1] = next_city

        ant_distances = np.zeros((m,1))
```

```
        for i in range(m):
            single_ant_distance = 0
            for j in range(n-1):
                single_ant_distance = single_ant_distance +
d[int(routes[i,j]),int(routes[i,j+1])]
            single_ant_distance = single_ant_distance + d[int(routes[i,9]),int(0)]
            ant_distances[i] = single_ant_distance
        #Calculation of the total distance traveled by each ant.

        min_distance_index = np.argmin(ant_distances)
        #Location of minimum distance route.
        minimum_distance = ant_distances[min_distance_index]
        #Minimum distance of route.
        best_route = routes[min_distance_index]

        if t == 0:
            starting_distance = minimum_distance

        pheromone = (1 - e) * pheromone
        #Pheromone evaporation.

        for i in range(m):
            for j in range(n - 1):
                pheromone_quantity = 1 / ant_distances[i]
                if (best_route[j], best_route[j + 1]) == (routes[i, j], routes[i, j +
1]):
                    pheromone[int(routes[i, j]), int(routes[i, j + 1])] =
pheromone[int(routes[i, j]), int(
                        routes[i, j + 1])] + pheromone_quantity
                else:
                    pheromone[int(routes[i, j]), int(routes[i, j + 1])] =
pheromone[int(routes[i, j]), int(
                        routes[i, j + 1])] + 0
        #Pheromonoe deposition part. The most visited cities get the largest amount of
pheromone.

    routes_with_return = np.append(routes,np.zeros([len(routes),1]),1)
    best_route = routes_with_return[min_distance_index,:]


return(routes_with_return,min_distance_index,best_route,minimum_distance,starting_distanc
e)



if __name__ == "__main__":

    print("ANT COLONY SIMULATION")
    ant_colony_simultion()
    print()
    print("ANT SYSTEM TSP")
```

```
    city_data = np.array([[0, 1], [3, 2], [6, 1], [7, 4.5], [15, -1],
                          [10, 2.5], [16, 11], [5, 6], [8, 9], [1.5, 12]])

    d = euclidean_distances(city_data, city_data)
    #Matrix of distances between cities.

    m = 10
    n = 10
    Tmax = 200
    e = 0.5
    alpha = 0.1
    beta = 1
    #Ant colony parameters.

    routes,min_distance_index,best_route,minimum_distance,starting_distance =
ant_colony_tsp(m,n, Tmax, e, alpha, beta,d)

    print('Routes chosen by each of the M ants:')
    print(routes)
    print('Shortest distance route:', best_route)
    print('Minimal total distance found:', (minimum_distance))

    fig = plt.figure()
    x = []
    y = []

    for i in range(len(best_route-1)):
        x.append(city_data[int(best_route[i]),0])
        y.append(city_data[int(best_route[i]),1])

    plt.plot(x,y,'r-p')
    for i, txt in enumerate(best_route):
        plt.annotate(txt, (x[i], y[i]))

    plt.title("Route of traveling salesman after Ant System",fontweight='bold')
    fig.text(0.5, 0.85, "Initial distance: {}".format(starting_distance), ha='center',
va='center',)
    fig.text(0.5, 0.8, "Shortest distance: {}".format(minimum_distance), ha='center',
va='center',)
    plt.text(0,0,"Start")
    plt.grid()
    plt.show()

    pass
```

3. Solution of first task

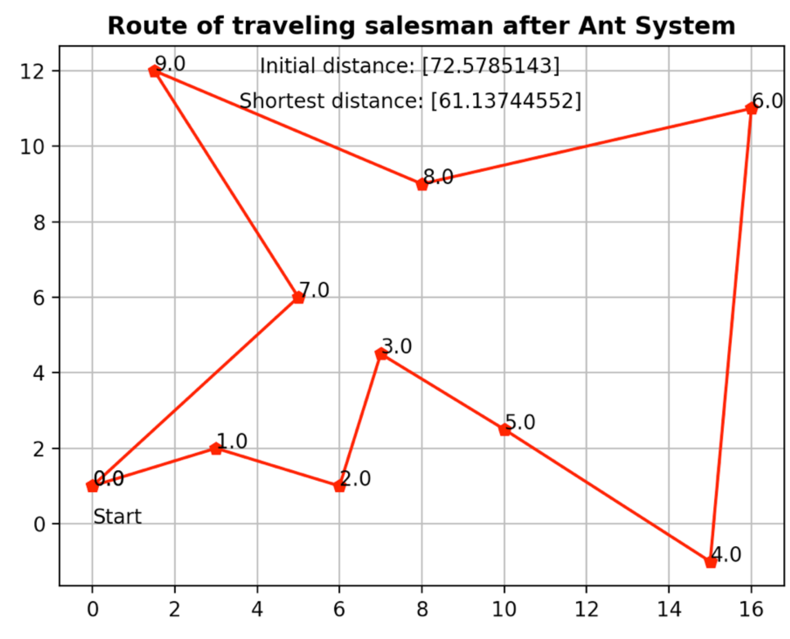


Commentary:

- In the first task it was assumed that the shorter arch is 5 and the longer one is 10. The ants have to make 2 decisions at two intersections. Arches are marked L1,R1,L2,R2 respectively.
- The following results were obtained after a thousand ants had passed the route from nest to food:

  Console output:
  ```
  Ants decisions in first intersection  L1: 846, R1: 154
  Ants decisions in second intersection  L2: 127, R2: 873
  Amount of pheromone on L1: 0.4913147232406854, R1:
  0.007605143566189608, L2: 0.005444877717993891, R2: 0.49563525601318614
  ```

4. Solution of second Task



Console output:

```
Routes chosen by each of the M ants:
[[0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
 [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]]
Shortest distance route: [0. 1. 2. 3. 5. 4. 6. 8. 9. 7. 0.]
Minimal total distance found: [61.13744552]
```

Commentary:

- In order to avoid the algorithm falling into the local minimum, the choice of roulette wheel selection was used.
- The solution uses elements from Max-Min Ant System. This allows pheromone to be concentrated in more efficient solutions
- Minimal total distance found by ant system is the same as in the genetic algorithm from laboratory task 1 – 61.137. Repeatedly obtaining the same result may indicate its correctness.
- With several algorithm runs the final results are sometimes slightly different, however final result is always much smaller than the initial one.