Laboratory of Evolutionary Algorithms

Laboratory 3:
Ant Systems

Author:
Agata Raczyńska

1. Task 1

A map of environment with separated nest and food by a double asymmetric bridge was created with following coordinates:
x = [0,1,2,2,3,4,5,5,6,7]
y = [2,2,0,3,2,2,4,1,2,2]

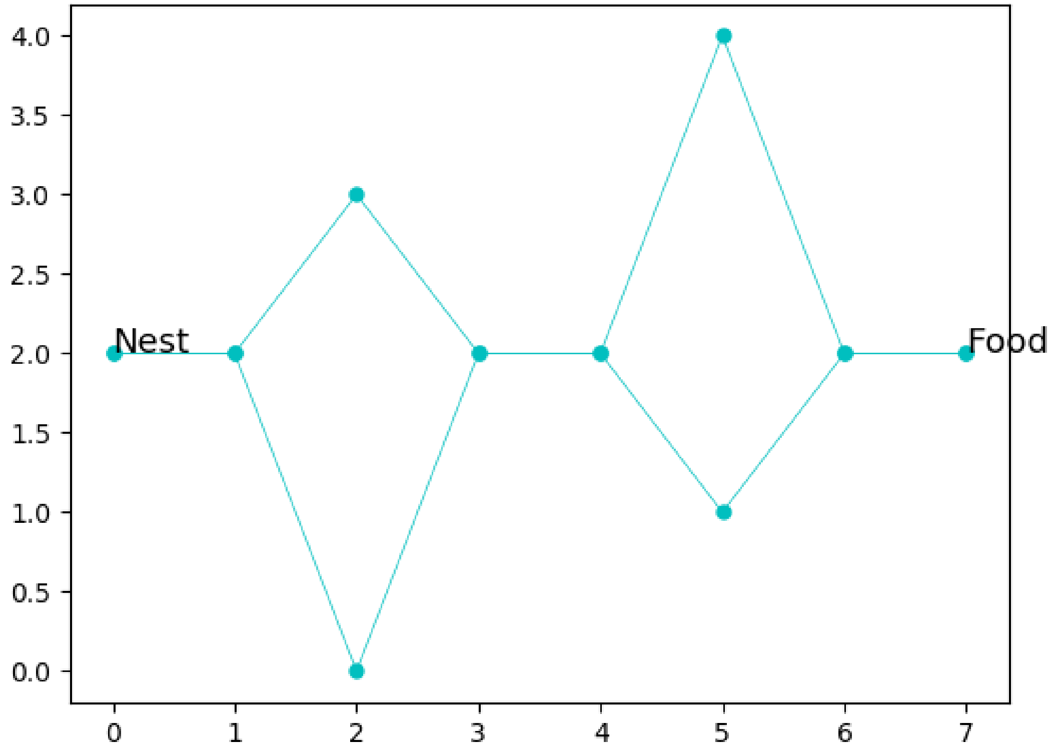

Fig.1.1. Map of an environment with a double asymmetric bridge.

An ant system was applied in order to simulate the ants colony behavior. In the model, 1000 ants were used.
The probability of choosing the upper branch (pR) and lower branch (pL) was given by an equation:

$$P_R(m) = \frac{(R_m + k)^d}{(R_m + k)^d + (L_m + k)^d}.$$

$$P_L(m) = 1 - P_R(m)$$

A random number r = [0,1] was chosen in order to decide, which branch should be chosen by the ant according to the rule:

$$\begin{cases} R_{m+1} = R_m + 1, L_{m+1} = L_m, & r \le P_R(m), \\ R_{m+1} = R_m, L_{m+1} = L_m + 1, & \text{otherwise}, \end{cases}$$

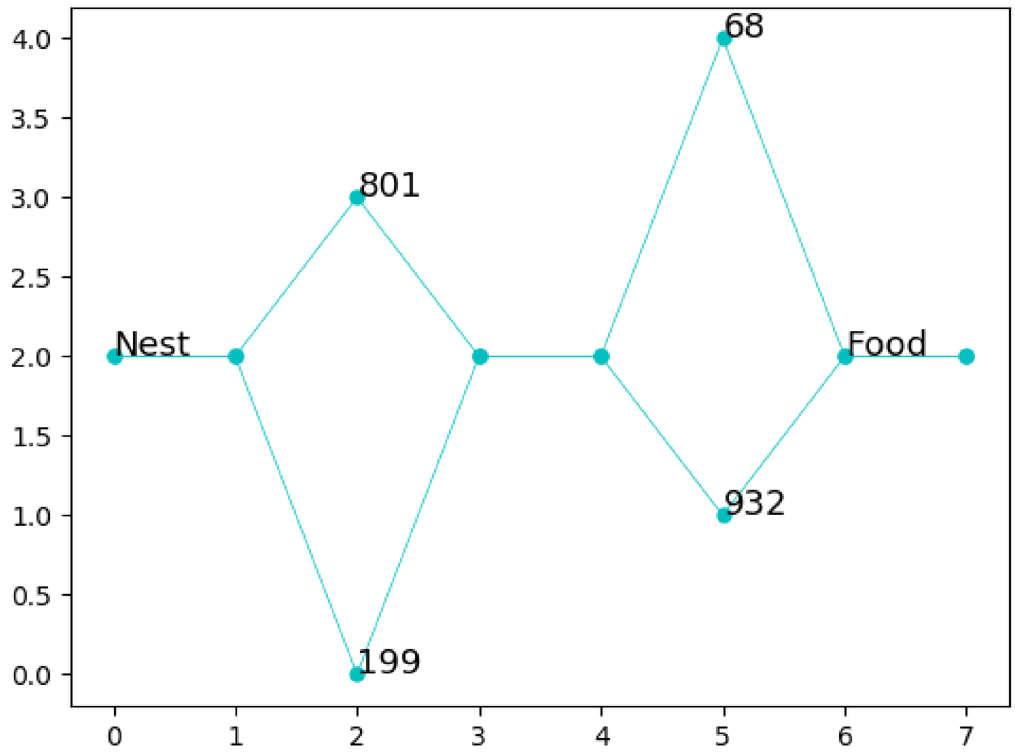

Fig.1.2. Map of an environment with a double asymmetric bridge.

Results of the ant colony simulation show, that much more ants (80.1% for the 1st bridge and 93.2% for the 2nd bridge) chose the shorter path through the bridges.

Conclusions

Simulation of ants system that choose the path where more ants have passed (and therefore more pheromone has been disposed) allowed to model behavior of an ants colony that eventually find the shortest path to the food.

2. Task 2

The set of N = 10 cities used was set 1 provided by tutor at laboratory 1- Genetic Algorithm for Traveling Salesman Problem. In this way, a set with known solution was used in order to compare results and measure ants system performance.

x = [0, 3, 6, 7, 15, 12, 14, 9, 7, 0]
y = [1, 4, 5, 3, 0, 4, 10, 6, 9, 10]

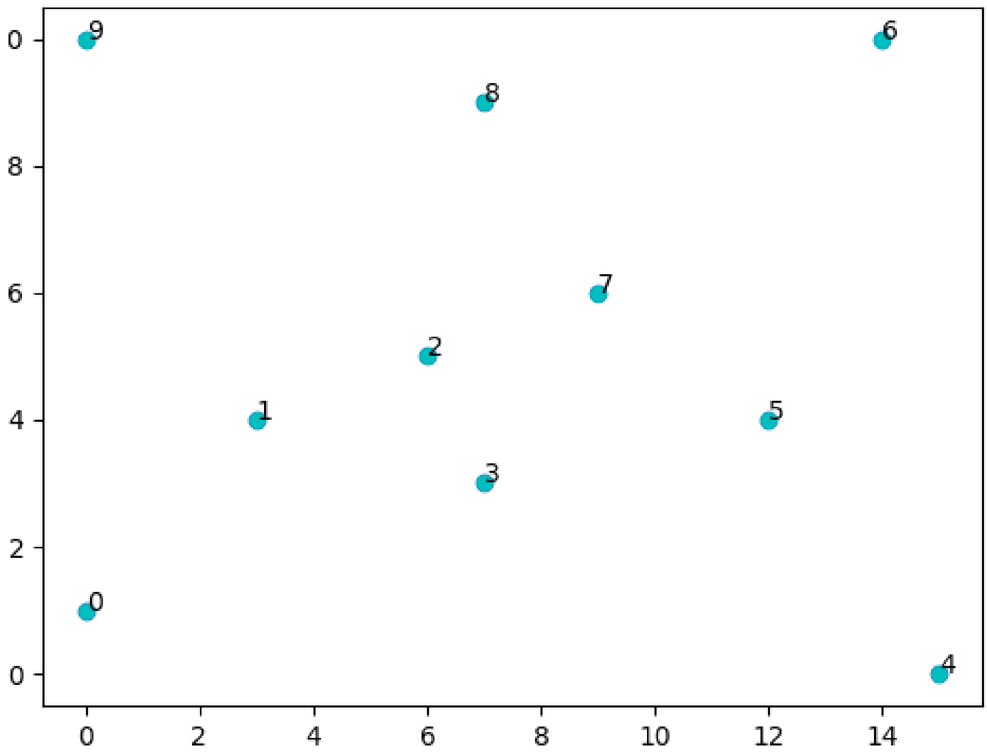

Fig.2.1. Map of cities.

An ant system was implemented in order to solve the travelling salesman problem.

Minimal total distance traveled: **55.044**

Sequence of cities to be visited ensuring the minimal total distance traveled:
[0, 1, 2, 3, 7, 5, 4, 6, 8, 9, 0]
And its perturbations.

Amount of pheromones at given routes:

	City 0	City 1	City 2	City 3	City 4	City 5	City 6	City 7	City 8	City 9
City 0	0,00	4,71	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
City 1	0,00	0,00	6,32	0,00	0,00	0,00	0,00	0,00	0,00	0,00
City 2	0,00	0,00	0,00	8,94	0,00	0,00	0,00	0,00	0,00	0,00
City 3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	5,55	0,00	0,00
City 4	0,00	0,00	0,00	0,00	0,00	0,00	1,99	0,00	0,00	0,00
City 5	0,00	0,00	0,00	0,00	4,00	0,00	0,00	0,00	0,00	0,00
City 6	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	2,83	0,00
City 7	0,00	0,00	0,00	0,00	0,00	5,55	0,00	0,00	0,00	0,00
City 8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	2,83
City 9	2,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Fig. 2.2. Table of pheromones disposed on given routes. Colorful cells are the chosen solution.

It can be seen, that at optimal routes, the amount of pheromones was much higher. In all other cells the amount was ~0. The chosen routes as the one with highest pheromone levels correspond to the optiml sequence of cities chosen.

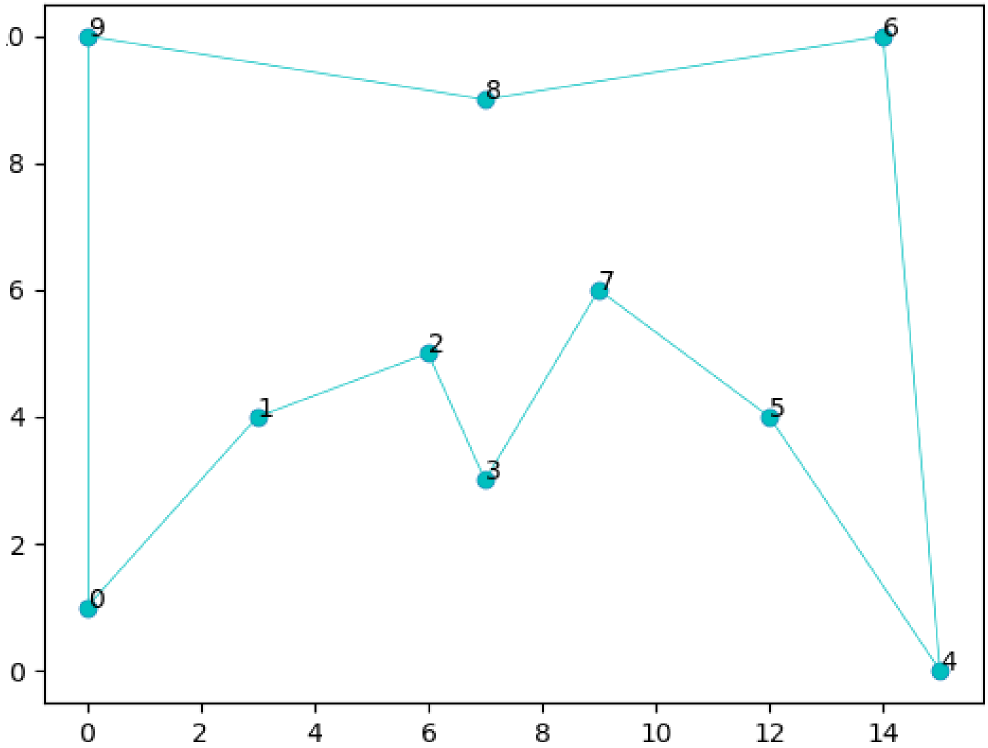

Fig.2.3. Map of cities with the applied optimal solution
– results representation in a graphic form.

Conclusions

Application of ants systems allowed to quickly find the best solution of the travelling salesman problem. The found solution corresponds to the one found with Genetic Algorithm during laboratory 1 for the given set of cities.

3. Implemented code

```
import numpy as np
import matplotlib.pyplot as plt
import math
import random

def euclidean_distance(x,y):
    d = np.zeros((len(x),len(y)), dtype=float, order='C')
    for i in range(len(x)):
        for j in range(len(y)):
            d[i,j] = math.sqrt( (x[i]-x[j])**2 + (y[i]-y[j])**2 )
    return d

def unique(list1):
    unique_list = []
    for x in list1:
        # check if exists in unique_list or not
        if x not in unique_list:
            unique_list.append(x)
    return unique_list
```

```python
def pR(U, L, k, d):
    p_r = ((U + k) ** d) / ((U + k) ** d + (L + k) ** d)
    return p_r

def A(sp, sd):
    a = (sp ** alpha * (1 / sd) ** beta) / (sum(sp ** alpha * (1 / sd) **
beta))
    return a

# Nest - food enviroment
print("Nest - food enviroment")
print(" ")

x = [0,1,2,2,3,4,5,5,6,7]
y = [2,2,0,3,2,2,4,1,2,2]

#Plot map
x1 = [0,1,2,3,4,5,6,7]
y1 = [2,2,0,2,2,4,2,2]
x2 = [0,1,2,3,4,5,6,7]
y2 = [2,2,3,2,2,1,2,2]
fig, ax = plt.subplots()
ax.plot(x1, y1,'co-', linewidth=0.5, markersize = 5)
ax.plot(x2, y2,'co-', linewidth=0.5, markersize = 5)
l = ['Nest','','','','','','','','','Food'] #labels
for i, txt in enumerate(l):
    ax.annotate(txt, (x[i], y[i]), fontsize=13)
plt.show()

#Initial values
num_ants = 1000
Upper_bridge = [0,0]
Lower_bridge = [0,0]
k = 20
d = 2

distances = euclidean_distance(x,y)
unique_moves = unique(x)

for i in range(num_ants):
    for j in range(len(unique_moves)):
        r = random.uniform(0, 1)
        if j == 2:
            U = Upper_bridge[0]
            L = Lower_bridge[0]
            p_R = pR(U,L,k,d)
            p_L = 1 - p_R
            if r <= p_R:
                Upper_bridge[0] = Upper_bridge[0] +1
            else:
                Lower_bridge[0] = Lower_bridge[0] + 1
        elif j == 5:
            U = Upper_bridge[1]
            L = Lower_bridge[1]
            p_R = pR(U,L,k,d)
            p_L = 1 - p_R
            if r <= p_R:
                Upper_bridge[1] = Upper_bridge[1] + 1
            else:
                Lower_bridge[1] = Lower_bridge[1] + 1
```

```
fig, ax = plt.subplots()
ax.plot(x1, y1,'co-', linewidth=0.5, markersize = 5)
ax.plot(x2, y2,'co-', linewidth=0.5, markersize = 5)
l = 
['Nest','',Upper_bridge[0],Lower_bridge[0],'','',Lower_bridge[1],Upper_brid
ge[1],'Food'] #labels
for i, txt in enumerate(l):
    ax.annotate(txt, (x[i], y[i]), fontsize=13)
plt.show()

# Travelling salesman problem
print(" ")
print("Travelling salesman problem")
print(" ")

#cities 1
x = [0, 3, 6, 7, 15, 12, 14, 9, 7, 0]
y = [1, 4, 5, 3, 0,  4,  10, 6, 9, 10]

#Initial values
Tmax = 200
alpha = 1
beta = 5
ro = 0.5
ants = 10
T = 0
N = 10

start = 0
best = []
best_ph = []
a = np.zeros((len(x), len(y)))
pheromones = np.ones((len(x), len(y)))
distances = euclidean_distance(x,y)

while T < Tmax:
    ph_canals = np.zeros(distances.shape + (ants,)) #10 canals of 
pheromones matrix for each ant
    routes = []

    for ant in range (ants):
        unvisited_cities = list(range(len(x))) #list of unvisited cities
        proposed_route = []
        proposed_route.append(start) #starting point of journey
        current_city = unvisited_cities.pop(start) #take out city 0 from 
unvisited cities
        quantity = np.zeros(distances.shape)

        for i in range(N-1):
            teta = pheromones[current_city, unvisited_cities]
            eta = distances[current_city, unvisited_cities]
            a = A(teta,eta)
            p = a/sum(a)
            next_city = np.random.choice(unvisited_cities, p = p) #a random 
city is chosen
            unvisited_cities.remove(next_city)
            quantity[current_city, next_city] = 1/distances[current_city, 
next_city]
            current_city = next_city
            proposed_route.append(next_city) #
```

```
        quantity[current_city, start] = 1 / distances[current_city, start]
        proposed_route.append(start) #go back to 1st city
        ph_canals[:,:,ant] = quantity #each ant saves pheromone memory
        routes.append(proposed_route)

    ph_sum = np.sum(np.sum(ph_canals, axis = 1), axis = 0) #sum pheromones of each ant
    index_best = np.argmax(ph_sum) #choose best ant
    best.append(routes[index_best])
    best_ph.append(ph_sum[index_best])
    delta_ph = np.sum(ph_canals, axis = 2) #sum pheromones of all ants (from all canals)
    pheromones = (1-ro) * pheromones + delta_ph #update deposition and evaporation of pheromone on all routes

    T = T+1

np.savetxt('pheromones.csv', pheromones, delimiter=',', fmt='%s')

best_idx = best_ph.index(max(best_ph))
best_route = best[best_idx]
print("Travelling salesman problem - best route: ", best_route)
dist = 0
for i in range (1, len(best_route)):
    c = best_route[i-1]
    n = best_route[i]
    dist = dist + distances[c,n]
print("Travelling salesman problem - total distance traveled: ",dist)

x2 = []
y2 = []
for i in range(N):
    x2.append(x[best_route[i]])
    y2.append(y[best_route[i]])
x2.append(x[best_route[0]])
y2.append(y[best_route[0]])

l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] #labels
fig, ax = plt.subplots()
ax.plot(x2[0:N+1], y2[0:N+1], 'co-', linewidth=0.5, markersize = 5)
for i, txt in enumerate(l):
    ax.annotate(txt, (x[i], y[i]), fontsize=13)
plt.show()
```