Ant Colony System: Application to traveling salesman problem

Optimization

Nieves Montes Gómez (1393150)

February 4, 2020

1 Introduction

The purpose of this assignment is to apply the Ant Colony System (ACS) algorithm to tackle the benchmark traveling salesman problem. The pseudocode for the solution is displayed in Algorithm 1. A copy of the complete Python code is attached at the end of this document. It can also be found here, where the necessary csv files for running the program are also included.

Algorithm 1: Ant Colony System algorithm for traveling salesman problem

```
Input: Distances and coordinates of European cities.
Output: A cyclic path joining all cities, going only once through each one of them.
1  DepositInitialPheromones($\tau_0$)
2  InitializeRandomAntColony(colsize)
3  BestSoFar ← best solution in colony
4  while BestSoFar not updated for stop iterations do
5      for each ant in the colony do
6          FindNewPath for the ant, according to pseudorandom proportional rule (with
             probability $q_0$) and transition probabilities, with LocalPheromoneUpdate($\phi$)
7      end
8      BestInColony ← best solution in colony
9      GlobalPheromoneUpdate($\rho$) with best only offline update, best is considered to be the
         best ant in the current iteration (BestInColony)
10     if path of BestInColony $<$ path of BestSoFar then
11         BestSoFar ← BestInColony
12         stop$+=$max_iter
13     end
14 end
```

The transition probabilities are defined as the probability that one ant, while building its new path and currently sitting in city i, will add the edge $i \leftrightarrow j$ to its path. Given the requirements of the problem, city j must not be already visited in the previous steps of the path. The transition probability is given by the following equation:

$$P(i \leftrightarrow j) = \frac{(\tau_{i\leftrightarrow j})^\alpha \cdot (\eta_{i\leftrightarrow j})^\beta}{\sum_{\text{allowed } k} (\tau_{i\leftrightarrow k})^\alpha \cdot (\eta_{i\leftrightarrow k})^\beta} \tag{1}$$

where $\tau_{i \leftrightarrow j}$ is the pheromone concentration on edge $i \leftrightarrow j$, and $\eta_{i \leftrightarrow j}$ is the inverse of the length of edge $i \leftrightarrow j$. Note that the double-pointed arrow notation indicates that the edges of the graph representing Europe are undirected.

There are many parameters involved in the algorithm. Finding suitable values requires some experimenting by trial and error. Following is a list of them, a brief explanation for each and the value they have been instantiated to:

- `tau_0` is the initial pheromone concentration that is deposited on all edges of the graph. It is set to 0.01.
- `colsize` is the number of ants in the colony, *i.e.* the number of candidate solution simultaneously "alive". It is set to 20.
- `stop` is the initial number of iterations the algorithm is allowed to run before halting. It is set to 50.
- `max_iter` is the number of iterations the algorithm is allowed to run after the best solution has been updated. It is set to 20.
- `q0` is the parameter governing pseudorandom proportional selection rule. It is set to 0.8.
- `alpha` controls the influence of pheromone concentration on the transition probabilities. It is set to 1.
- `beta` controls the influence of the edge cost (the inverse of distance) on the transition probabilities. It is set to 2.
- `phi` is the pheromone decay coefficient. It characterizes the convex combination of pheromone concentrations applied to selected edges during the local pheromone update. It is set to 0.04.
- `rho` is the pheromone evaporation rate. It characterizes the convex combination of pheromone concentrations applied to all edges during the best-only offline pheromone update. It is set to 0.02.

2 Results

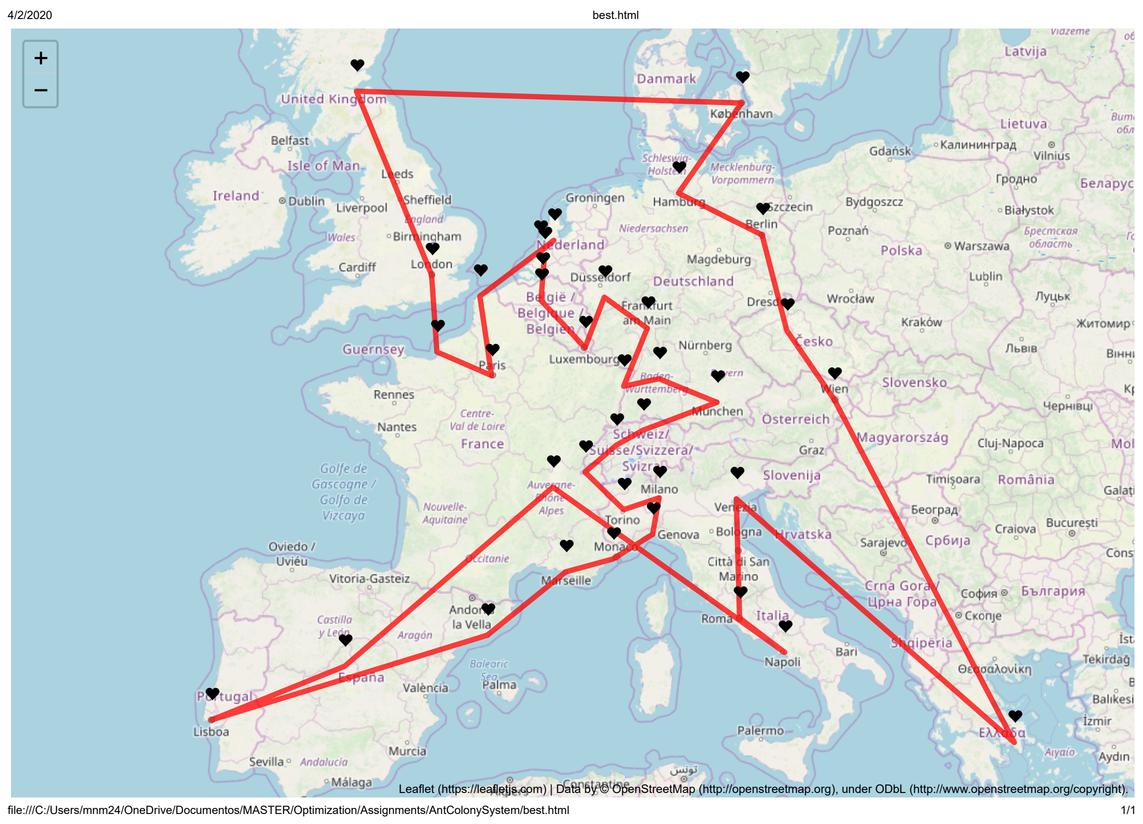
Examples of some outputs are:

```
Best solution found during iteration #119.
This clever ant visits all cities in a 18370.0 km tour.
Her adventorous journey is: Barcelona-Marseille-Nice-Genoa-Rome-Naples-Venice-
    Milan-Turin-Geneva-Bern-Zurich-Munich-Stuttgart-Strasbourg-Luxembourg-
    Cologne-Frankfurt-Copenhagen-Hamburg-Berlin-Prague-Vienna-Athens-Edinburgh-
    London-Calais-Brussels-Antwerp-Amsterdam-The Hague-Rotterdam-Paris-Le Havre-
    Lyon-Madrid-Lisbon-Barcelona

Best solution found during iteration #115.
This clever ant visits all cities in a 17827.0 km tour.
Her adventorous journey is: Barcelona-Marseille-Nice-Genoa-Milan-Turin-Geneva-
    Bern-Zurich-Munich-Stuttgart-Strasbourg-Frankfurt-Cologne-Luxembourg-
    Brussels-Antwerp-Rotterdam-The Hague-Amsterdam-Calais-Paris-Le Havre-London-
    Edinburgh-Copenhagen-Hamburg-Berlin-Prague-Vienna-Athens-Venice-Rome-Naples-
    Lyon-Madrid-Lisbon-Barcelona
```

```
Best solution found during iteration #47.
This clever ant visits all cities in a 19038.0 km tour.
Her adventorous journey is: Barcelona-Marseille-Turin-Milan-Genoa-Nice-Lyon-
    Geneva-Bern-Zurich-Strasbourg-Luxembourg-Cologne-Frankfurt-Stuttgart-Munich-
    Prague-Vienna-Venice-Rome-Naples-Athens-Berlin-Hamburg-Copenhagen-Rotterdam-
    The Hague-Amsterdam-Antwerp-Brussels-Calais-London-Le Havre-Paris-Edinburgh-
    Madrid-Lisbon-Barcelona
```

Please note the program also produced an html file with an interactive map of the solution route.



4/2/2020
best.html
Leaflet (https://leafletjs.com) | Data by © OpenStreetMap (http://openstreetmap.org), under ODbL (http://www.openstreetmap.org/copyright).
file:///C:/Users/mnm24/OneDrive/Documentos/MASTER/Optimization/Assignments/AntColonySystem/best.html
1/1

Figure 1: Map of the route for the best solution found (third example output, 17827.0 km tour). The interactive version of the map is included in the delivery.

3 Conclusions

The Ant Colony System algorithm has been succesfully employed to solve the traveling salesman problem. Suitable parameters to solve the problem are proposed, however they are possibly far from optimal as they have been found by trial and error. The stopping criteria for the search (not updating the best solution for a maximum number of iterations) leads to find acceptable routes, however not necessarily the best ones. After many runs, the favourite solution has been found to be 17827 km long.

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Feb  3 09:29:41 2020

@author: Nieves Montes Gómez

@description: Ant Colony System algorithm to solve the Traveling Salesman problem.
"""

import pandas as pd
import numpy as np
import os
import folium


"""Import data on distances between European cities."""
file = pd.read_csv("distancesEurope.csv", sep=';', index_col = 0)
europe = {}
cities = list(file.columns)
n = len(cities)
for city in cities[:-1]:
  neighbors = cities[cities.index(city)+1:]
  for neighbor in neighbors:
    europe.update({(city, neighbor):file[city][neighbor]})
del city, file, neighbors, neighbor

coordinates = pd.read_csv("coordinates.csv", sep=';', index_col = 0)


"""ACS parameters:"""
alpha = 1. # influence of pheromone values (exploitation)
beta = 2. # influence of edges costs (exploration)
q0 = 0.8 # pseudorandom proportionate rule parameter
phi = 0.04 # pheromone decay coefficient (local pheromone update)
rho = 0.02 # evaporation rate (global pheromone update)
tau0 = 0.01 # initial value of pheromone
max_iter = 20


"""Initialize pheromone values."""
pheromones = {}
for pair in list(europe.keys()):
  pheromones.update({pair: tau0})
del pair


class Ant:
  def __init__(self):
    """Class that represents an ant that follows a path with a certain length."""
    self.path = []
    self.path_length = 0.

  def buildNewPath(self):
    """An ant builds a new path according to pseudorandom proportional rule.
    Pheromone values are updated locally. The total distance of the path is
    computed."""
    self.path = [] # erase previous path
    self.path_length = 0.
    self.path.append('Barcelona') # start from BCN
    for _ in range(1,n):
      # select possible next stops
      poss_next = cities[:]
      # delete already visited cities
      for visited in self.path:
        poss_next.remove(visited)
      probs = {}
      # get tau and eta for each next possible option
      for opt in poss_next:
        if (self.path[-1], opt) in europe.keys():
          probs.update({opt : pheromones[(self.path[-1], opt)]**alpha /
                                         europe[(self.path[-1], opt)]**beta})
        elif (opt, self.path[-1]) in europe.keys():
          probs.update({opt : pheromones[(opt, self.path[-1])]**alpha /
                                         europe[(opt, self.path[-1])]**beta})
        else:
          raise IndexError('Something went wrong.')
      total = sum(list(probs.values()))
      for k in probs:
        probs[k] = probs[k] / total
      # select next destination: pseudorandom proportional rule
      q = os.urandom(1)[0]/255.
      if q < q0:
        next_index = np.argmax(list(probs.values()))
        self.path.append(list(probs.keys())[next_index])
      else:
        next_city = np.random.choice(list(probs.keys()),
                                     p = list(probs.values()))
        self.path.append(next_city)
      # local pheromone update and add edge cost to total distance
      if (self.path[-1], self.path[-2]) in pheromones.keys():
        pheromones[(self.path[-1], self.path[-2])] = (1-phi) * pheromones[
            (self.path[-1], self.path[-2])] + phi*tau0
        self.path_length += europe[(self.path[-1], self.path[-2])]
      elif (self.path[-2], self.path[-1]) in pheromones.keys():
        pheromones[(self.path[-2], self.path[-1])] = (1-phi) * pheromones[
            (self.path[-2], self.path[-1])] + phi*tau0
        self.path_length += europe[(self.path[-2], self.path[-1])]
      else:
        raise IndexError('Something went wrong.')
    # after exit the loop: local pheromone and distance update on edge closing the cycle
    if (self.path[-1], 'Barcelona') in pheromones.keys():
      pheromones[(self.path[-1], 'Barcelona')] = (1-phi) * pheromones[
          (self.path[-1], 'Barcelona')] + phi*tau0
      self.path_length += europe[(self.path[-1], 'Barcelona')]
    elif ('Barcelona', self.path[-1]) in pheromones.keys():
      pheromones[('Barcelona', self.path[-1])] = (1-phi) * pheromones[
          ('Barcelona', self.path[-1])] + phi*tau0
      self.path_length += europe[('Barcelona', self.path[-1])]
    else:
      raise IndexError('Something went wrong.')

  def mapPath(self):
    """Plot the ant's path on the Europe map."""
    m = folium.Map(location=(48, 10), zoom_start = 5, min_zoom = 4, max_zoom = 10)
    points = []
    for city in self.path:
      points.append((coordinates.loc[city, 'lat'], coordinates.loc[city, 'lon']))
      folium.Marker((coordinates.loc[city, 'lat'], coordinates.loc[city, 'lon']),
                    popup = folium.Popup(city),
                    icon = folium.Icon(color='red', icon='heart')).add_to(m)
    points.append((coordinates.loc[self.path[0], 'lat'],
                   coordinates.loc[self.path[0], 'lon']))
    folium.PolyLine(points, color = 'red', weight = 5, opacity = 0.75).add_to(m)
    return m

  def __str__(self):
    return ('Hi, I\'m a curious ant exploring Europe. I can visit all cities in a ' +
            str(self.path_length) + ' km tour.')


class Colony:
  def __init__(self, colsize=20):
    """Class that represents a colony of ants. colsize is the number of ants.
    Ants are randomly initialized."""
    self.size = colsize
    self.members = [Ant() for _ in range(colsize)]
    self.best = self.members[0]
    for member in self.members:
      member.buildNewPath()
    self.findBest()

  def findBest(self):
    """Find the ant in the current iteration that has the shortest path."""
    for member in self.members:
      if member.path_length < self.best.path_length:
        self.best = member

  def newPaths(self):
    """Send all ants in the colony in search of a new path."""
    for member in self.members:
      member.buildNewPath()

  def globalPheromoneUpdate(self):
    """Global pheromone update. Must be called after best ant has already been found."""
    DeltaBest = 1/self.best.path_length
    # common update to all edges
    for edge in pheromones.keys():
      pheromones[edge] *= (1-rho)
    # update to edges in best path
    for i in range(len(self.best.path) - 1):
      if (self.best.path[i], self.best.path[i+1]) in pheromones.keys():
        pheromones[(self.best.path[i], self.best.path[i+1])] += rho*DeltaBest
      elif (self.best.path[i+1], self.best.path[i]) in pheromones.keys():
        pheromones[(self.best.path[i+1], self.best.path[i])] += rho*DeltaBest
      else:
        raise IndexError('Something went wrong.')
    # edges that closes the cycle of the best path
    if (self.best.path[0], self.best.path[-1]) in pheromones.keys():
      pheromones[(self.best.path[0], self.best.path[-1])] += rho*DeltaBest
    elif (self.best.path[-1], self.best.path[0]) in pheromones.keys():
      pheromones[(self.best.path[-1], self.best.path[0])] += rho*DeltaBest
    else:
      raise IndexError('Something went wrong.')


"""Ant Colony System"""
import copy
col = Colony()
bestSoFar = copy.deepcopy(col.best)
iters = 1
stop = 50
bestIter = 0
```

```
while iters <= stop:
  col.newPaths()
  col.findBest()
  col.globalPheromoneUpdate()
  if col.best.path_length < bestSoFar.path_length:
    bestSoFar = copy.deepcopy(col.best)
    stop += max_iter
    bestIter = iters
  iters += 1

# output result
print('Best solution found during iteration #' + str(bestIter) + '.')
print('This clever ant visits all cities in a ' + str(bestSoFar.path_length) +
      ' km tour.')
print('Her adventorous journey is:', end = ' ')
for city in bestSoFar.path:
  print(city, end = '-')
print(bestSoFar.path[0])

# export map of route as html
m = bestSoFar.mapPath()
m.save('tour.html')
```