IE 517 Final Project

Emre Kara

June 28, 2021

1 Problem Definition

Addition to the Traveling Salesperson Problem (TSP), TSP with profits (TSPP) has profits for each customer. Also, it does not have a constraint of visiting all customers. Objective of the problem is to maximize net profit which equals to the profit earned by visiting customers less travel cost.

To solve 3 problems of TSPP with low and high profits (six instances in total), in this study, Ant Colony System (ACS) algorithm is adapted. The choice of the algorithm is based on the fact that ant colony algorithms were adapted fist to solve TSP problems. Also, its pheromone mechanic can be useful to design a robust algorithm that could handle dynamic cost values. Six problems that are solved in the study are:

- eil51: Low profits and high profits
- eil76: Low profits and high profits
- eil101: Low profits and high profits

2 Solution Approach

In the original paper, ACS algorithm is applied to solve TSP with fixed distances [1]. However, in this work, it is utilized to solve a TSP with dynamic distances which is also a transformed version of TSPP. Throughout the algorithm, permutation solution representation is used which stands for the order of the cities visited. All permutation lists starts with node 0 which is the depot and ends with node n+1 which is a duplicate of the depot.

Most of the parameters of the algorithm are given same values as the original paper. However, number of ants are kept at the same level as number of nodes to increase diversity. Also, algorithm is run for 500 iterations for each instance. Parameter values used in study are:

- $m = n$
- $\tau_0 = (n * L_{NN})^{-1}$
- $cl = 15$
- $q_0 = 0.9$
- $\alpha = 1$
- $\beta = 2$
- $\rho_0 = 0.1$

where m stands for number of ants,n for number of customers, cl for the candidate list size used when moving ants from one city to another and $q0$ for exploitation probability. Other three parameters are: α, exponentiation parameter of pheromone while calculating probabilities, β, same paramter for visibility values and finally ρ_0, evaporation parameter.

3 Problem Sets and Solutions

Execution times for eil51 instances were short, around 10 seconds. However, it took almost up to one minute for eil101 instances. These numbers indicate that although algorithm does not take so much time to run for these problems, it might perform slow for very large instances.

Number of customers visited are only one for eil51 and eil76 problems with low profits. This shows that it is highly possible that visiting customers are unprofitable for low profit instances. However, low profit instance of eil101 problem resulted in 97 customers which is close to 99 customers of high profit counterpart. Further details about results can also be seen from the table on the next page.

	Best Obj. Value (Net Profit)	No. of customers visited	Sequence of customers visited	CPU Time (s)
eil51-LP	-16.86	2	0, 45	13.25
eil51-HP	90.63	50	0, 31, 10, 37, 4, 48, 8, 49, 15, 1, 28, 20, 33, 29, 9, 38, 32, 44, 14, 43, 36, 16, 3, 46, 45, 50, 26, 47, 5, 13, 24, 17, 11, 41, 40, 18, 39, 12, 23, 22, 6, 25, 7, 30, 21, 27, 2, 34, 35, 19	12.406
eil76-LP	-23.84	2	0, 67	28.218
eil76-HP	307.67	76	0, 11, 39, 16, 50, 5, 67, 74, 75, 66, 33, 45, 7, 34, 6, 52, 13, 18, 53, 12, 26, 51, 44, 28, 47, 46, 20, 73, 27, 21, 61, 72, 32, 1, 29, 3, 25, 57, 71, 38, 8, 31, 43, 2, 15, 22, 55, 40, 41, 42, 62, 48, 23, 17, 49, 24, 54, 30, 9, 37, 64, 10, 65, 58, 56, 14, 4, 36, 19, 69, 59, 70, 35, 68, 60, 63	21.937
eil101-LP	88.83	98	0, 7, 44, 16, 83, 4, 59, 82, 17, 51, 88, 5, 93, 58, 98, 95, 94, 96, 91, 36, 97, 99, 90,84, 92, 60, 15, 43, 13, 41, 86, 1, 56, 14, 42, 37, 85, 12, 57, 39, 52, 100, 26, 27, 25, 11, 79, 67, 76, 2, 78, 32, 80, 8, 50, 19, 29, 69, 30, 87, 6, 81, 47, 46, 35, 48, 10, 18, 61, 9, 89, 62, 31, 65, 70, 34, 33, 77, 28, 23, 53, 54, 24, 38, 55, 74, 73, 71, 72, 40, 21, 20, 3, 22, 66, 75, 49, 68	48.015
eil101-HP	1518.28	100	0, 66, 38, 55, 74, 73, 71, 72, 20, 39, 57, 52, 100, 26, 27, 25, 11, 79, 67, 76, 2, 78, 32, 80, 8, 50, 19, 29, 69, 30, 87, 6, 47,81, 7, 44, 16, 83, 4, 59, 82, 17, 51, 88, 5, 93, 12,94, 96, 91, 58, 98, 95, 92, 84, 97, 36, 99, 90, 15, 60, 85, 43, 13, 41, 86, 1, 56, 14, 42, 40, 21, 22, 3,54, 24, 53, 23, 28, 77, 33, 34, 70, 65, 64, 31, 89, 62,10, 18, 46, 45, 35, 48, 63, 61, 9, 68, 49, 75	56.734

4 Results

As optimal objective values are not available, it is hard to evaluate results this way. However, it can be concluded that ACS can be used efficiently to obtain results for TSP with dynamic weights. To evaluate performance of the algorithm better, same problems can be solved with other algorithms and results can be compared in further studies.

5 Python Code

5.1 Read and Prepare Data

[1]:
```
import math
import re
import os
import random
import numpy as np
import pandas as pd
import time
from datetime import datetime
from matplotlib import pyplot as plt
```

[2]:
```
# Reading from txt file and saving it into rows variable as a list of rows
fdir = "dataset-TSPP.xls"
problems = ["eil51","eil76","eil101"]
data = list()
for i in range(3):
    data += [pd.read_excel(fdir,problems[i]).iloc[:,1:5]]
```

5.2 Algorithm Classes

[3]:
```
def dist(x,y = 0):
    return np.sqrt( np.sum((x - y)**2) )
```

[4]:
```
# Attributes: name, n, coords, profits, DistMat
# Methods: CreateDistMat()
class Problem():
    def __init__(self, Data, PrName, LH = 0):
        typ = "Low" if LH == 0 else "High"
        self.name = f"{PrName}{typ}"
        # Node 0 is duplicated as node n
        self.n = Data.shape[0] + 1
        self.coords = Data.iloc[:,0:2]
        self.coords = pd.concat([self.coords ,self.coords.loc[[0],:]])
        self.profits = Data[typ]
        self.profits = pd.concat([self.profits ,self.profits[0:1]])
        self.DistMat = self.CreateDistMat()
```

```
    def CreateDistMat(self):
        dist = np.sqrt(\
            np.power(
            np.diag (self.coords["x"] ) @ np.matrix( self.n*self.n *[1] ).
↪reshape(self.n,-1) -\
            np.matrix( self.n*self.n *[1] ).reshape(self.n,-1) @ np.diag(self.
↪coords["x"] ), 2) +\
            \
            np.power(\
            np.diag (self.coords["y"] ) @ np.matrix( self.n*self.n *[1] ).
↪reshape(self.n,-1) -\
            np.matrix( self.n*self.n *[1] ).reshape(self.n,-1) @ np.diag(self.
↪coords["y"] ), 2)\
                    )
        dist += np.diag(self.n * [float('inf')])
        dist[:,0] = float('inf')
        return np.around(dist,2)

    def __repr__(self):
        rep = f"{self.name} TSPP Problem"
        return rep

    def __str__(self):
        return self.__repr__()
```

[5]:
```
# Attributes: prob, T
# Methods: Solve()
class NNsolver():
    def __init__(self, prob):
        self.prob = prob
        self.T = Tour(self.prob)

    def Solve(self):
        while not self.T.isComp :
            options = pd.Series(self.T.UpdatedCostMat[self.T[-1],:])
            nxt = options.drop(self.T.nodes).idxmin()
            self.T.add(nxt)
        return self.T

    def __repr__(self):
        rep = f"NN Heuristic\nSol:{self.T}"
        return rep

    def __str__(self):
        return self.__repr__()
```

[6]:
```
# Attributes: prob, nodes, isComp, UpdatedCostMat
# Methods: add(), unvisited(), L(), z(), CandList(), LocPherUpd()
class Tour():
    def __init__(self, prob, nodes = [], isComp = False):
        self.prob = prob
        self.isComp = isComp

        if type(nodes) in [np.int64,np.int32,int]: nodes=[nodes]
        self.nodes = list(pd.unique(nodes))
        if len(self.nodes) < 1: self.nodes = [0] + self.nodes
        elif self.nodes[0] != 0: self.nodes = [0] + self.nodes
        if isComp and self.nodes[-1] != [self.prob.n-1]:
            self.nodes += [self.prob.n-1]

        self.UpdatedCostMat = self.prob.DistMat.copy()
        self.UpdateCostMatrix()

    def add(self, newNodes):
        if self.isComp:
            return
        if type(newNodes) in [np.int64,np.int32,int] : newNodes=[newNodes]
        newNodes = list(pd.unique(newNodes))
        newNodes = [node for node in newNodes if node not in self.nodes]
        if newNodes==[]:
            print("No Node can be added!")
            return

        self.nodes += newNodes
        if self.prob.n-1 in self.nodes:
            self.isComp = True
            self.__repr__()
        self.UpdateCostMatrix()

    def CandList(self, cl):
        i = self.nodes[-1]
        return [j for j in range(self.prob.n)\
         if self.UpdatedCostMat[i,j] in np.sort(self.UpdatedCostMat[i,:])[:cl]][:
↪cl]

    def L(self):
        if len(self.nodes)>1:
            res = np.sum(self.UpdatedCostMat[(self.nodes[:-1],self.nodes[1:])])
        else: res = 0
        return np.round(res,2)

    def z(self):
        res = self.L() - np.array(self.prob.profits).sum()
```

```
        return np.round(res,2)

    def UpdateCostMatrix(self):
        TotalProf = np.array(self.prob.profits).sum()
        EarnedProf = np.array(self.prob.profits)[self.nodes].sum()
        self.UpdatedCostMat[:,-1] = self.prob.DistMat[:,-1] + TotalProf -␣
↪EarnedProf

    def unvisited(self):
        allNodes = list(range(self.prob.n))
        unvisited = [n for n in allNodes if n not in self.nodes]
        return unvisited

    def __getitem__(self, key):
        if key < len(self.nodes):
            return self.nodes[key]
        else:
            print("key out of bounds!")

    def __repr__(self):
        rep = f"L={self.L()} z={self.z()}"
        if self.isComp:
            rep += f"\nComplete"
        else: rep += f"\nNOT complete"
        return rep

    def __str__(self):
        return self.__repr__()
```

[7]:
```
# Attributes: T
# Methods: Move(), LocalPhmoneUpd()
class Ant():
    def __init__(self, prob, InitNode):
        self.T = Tour(prob, InitNode)

    def Move(self, PhmoneMatrix, q0, alpha, beta, cl, rho, tau0):
        i = self.T[-1]
        opts = set(self.T.CandList(cl)).intersection(self.T.unvisited())
        if opts == set():
            nxt = pd.Series(self.T.UpdatedCostMat[i,:])[self.T.unvisited()].
↪idxmin()
        elif len(opts) == 1:
            nxt = list(opts)
        else:
            visibs = 1/pd.Series(self.T.UpdatedCostMat[i,:]); visibs =␣
↪visibs[list(opts)]
```

```
            phmones = pd.Series(np.array(PhmoneMatrix[i,:])[0]); phmones =␣
↪phmones[list(opts)]
            RawP = phmones.pow(alpha)*visibs.pow(beta)
            if np.random.random() <= q0:
                nxt = RawP.idxmax()
            else:
                p = RawP/RawP.sum()
                nxt = np.random.choice(list(opts), 1, replace = False,p = p)
        self.T.add(nxt)
        self.LocalPhmoneUpd(PhmoneMatrix, (i,nxt), rho, tau0)

    def LocalPhmoneUpd(self, PhmoneMatrix, arc, rho, tau0):
        PhmoneMatrix[arc] = PhmoneMatrix[arc]*(1-rho) + rho*tau0

    def __repr__(self):
        rep = f"Ant traversed {len(self.T.nodes)} nodes.\nL={self.T.L()}"
        return rep

    def __str__(self):
        return self.__repr__()
```

[8]:
```
# Attributes: prob, ants, phmones, [m, alpha, beta, rho, tau0, q0, cl, MaxIter],␣
↪Tplus, Lplus
# Members: Solve(), InitGen()
class ACSsolver():
    def __init__(self, prob, params):
        self.m, self.alpha, self.beta, self.rho,\
        self.tau0, self.q0, self.cl, self.MaxIter = params
        self.prob = prob

        self.phmones = np.matrix(self.prob.n*self.prob.n*[self.tau0]).
↪reshape(self.prob.n,-1)
        self.phmones[:,0] = 0
        self.phmones = self.phmones - np.diag(np.diag(self.phmones))

        self.ants = list()
        initNodes = np.random.choice(range(1,self.prob.n-1),self.m,replace=False)

        for i in range(self.m): self.ants += [Ant(self.prob, initNodes[i])]
        self.Tplus = list()
        self.Lplus = list()
        self.Tplus += [Tour(self.prob, np.random.choice(self.prob.n-1,1) ,␣
↪isComp=True)]
        self.Lplus += [self.Tplus[-1].L()]

    def Solve(self):
        itr = 0
```

```
        while itr < self.MaxIter:
            itr +=1
            self.MoveAnts()
            self.UpdateTplus()
            self.GlobalPhmoneUpdate()
        print(f"L+={self.Tplus[-1].L()}")
        print(f"z+={self.Tplus[-1].z()}")
        print(f"Visited {len(self.Tplus[-1].nodes)} nodes.")

    def MoveAnts(self):
        for ant in self.ants:
            for step in range(1,self.prob.n-1):
                if ant.T.isComp == False:
                    ant.Move(self.phmones, self.q0, self.alpha, self.beta, self.
↪cl, self.rho, self.tau0)

    def UpdateTplus(self):
        TplusNew = self.Tplus[-1]
        for ant in self.ants:
            if ant.T.L() < self.Lplus[-1]:
                TplusNew = ant.T
        self.Tplus += [TplusNew]
        self.Lplus += [TplusNew.L()]

    def GlobalPhmoneUpdate(self):
        delta = 1/self.Lplus[-1]
        for idx in range(len(self.Tplus[-1].nodes[:-1])):
            i=self.Tplus[-1].nodes[idx]
            j=self.Tplus[-1].nodes[idx+1]
            self.phmones[i,j] = self.phmones[i,j]*(1-self.rho) + delta*self.rho

    def __repr__(self):
        rep = f"ACS Solver\nzPlus={self.Tplus[-1].z()}"
        return rep

    def __str__(self):
        return self.__repr__()
```

5.3 Running Algorithm

[9]:
```
prList = list()
for i in range(3):
    prList += [Problem(data[i],problems[i],0)]
    prList += [Problem(data[i],problems[i],1)]
```

[10]:
```
LNN = list()
tau0 = list()
for p in prList:
    nnObj = NNsolver(p)
    nnObj.Solve()
    LNN += [np.round(nnObj.T.L(),2)]
    tau0 += [1/(p.n*LNN[-1])]
```

[11]:
```
acsObj = list()
```

5.3.1 eil51 Low Demand

[12]:
```
probNo = 0
```

[13]:
```
#params=[m, alpha, beta, rho, tau0, q0, cl, MaxIter]
params = [50,1,2,0.1,tau0[probNo],0.9,15,500]
```

[14]:
```
tstart = time.process_time_ns()
np.random.seed(13)
acsObj += [ACSsolver(prList[probNo],params)]
acsObj[-1].Solve()
tend = time.process_time_ns()
print(f"Executed in {1.e-9*(tend - tstart)} CPU*seconds.")
```

```
L+=407.86
z+=16.86
Visited 3 nodes.
Executed in 13.25 CPU*seconds.
```

5.3.2 eil51 High Demand

[15]:
```
probNo+=1
```

[16]:
```
#params=[m, alpha, beta, rho, tau0, q0, cl, MaxIter]
params = [50,1,2,0.1,tau0[probNo],0.9,15,500]
```

[17]:
```
tstart = time.process_time_ns()
np.random.seed(13)
acsObj += [ACSsolver(prList[probNo],params)]
acsObj[-1].Solve()
tend = time.process_time_ns()
print(f"Executed in {1.e-9*(tend - tstart)} CPU*seconds.")
```

```
L+=493.37
z+=-90.63
Visited 51 nodes.
Executed in 12.40625 CPU*seconds.
```

5.3.3 eil76 Low Demand

[18]:
```
probNo+=1
```

[19]:
```
#params=[m, alpha, beta, rho, tau0, q0, cl, MaxIter]
params = [75,1,2,0.1,tau0[probNo],0.9,15,500]
```

[20]:
```
tstart = time.process_time_ns()
np.random.seed(13)
acsObj += [ACSsolver(prList[probNo],params)]
acsObj[-1].Solve()
tend = time.process_time_ns()
print(f"Executed in {1.e-9*(tend - tstart)} CPU*seconds.")
```

```
L+=522.84
z+=23.84
Visited 3 nodes.
Executed in 28.21875 CPU*seconds.
```

5.3.4 eil76 High Demand

[21]:
```
probNo+=1
```

[22]:
```
#params=[m, alpha, beta, rho, tau0, q0, cl, MaxIter]
params = [75,1,2,0.1,tau0[3],0.9,15,500]
```

[23]:
```
tstart = time.process_time_ns()
np.random.seed(13)
acsObj += [ACSsolver(prList[probNo],params)]
acsObj[-1].Solve()
tend = time.process_time_ns()
print(f"Executed in {1.e-9*(tend - tstart)} CPU*seconds.")
```

```
L+=642.33
z+=-307.67
Visited 77 nodes.
Executed in 21.9375 CPU*seconds.
```

5.3.5 eil101 Low Demand

[24]:
```
probNo+=1
```

[25]:
```
#params=[m, alpha, beta, rho, tau0, q0, cl, MaxIter]
params = [100,1,2,0.1,tau0[0],0.9,15,500]
```

[26]:
```
tstart = time.process_time_ns()
np.random.seed(13)
```

```
acsObj += [ACSsolver(prList[probNo],params)]
acsObj[-1].Solve()
tend = time.process_time_ns()
print(f"Executed in {1.e-9*(tend - tstart)} CPU*seconds.")
```

```
L+=739.17
z+=-88.83
Visited 99 nodes.
Executed in 48.015625 CPU*seconds.
```

5.3.6 eil101 High Demand

[27]:
```
probNo+=1
```

[28]:
```
#params=[m, alpha, beta, rho, tau0, q0, cl, MaxIter]
params = [100,1,2,0.1,tau0[0],0.9,15,500]
```

[29]:
```
tstart = time.process_time_ns()
np.random.seed(13)
acsObj += [ACSsolver(prList[probNo],params)]
acsObj[-1].Solve()
tend = time.process_time_ns()
print(f"Executed in {1.e-9*(tend - tstart)} CPU*seconds.")
```

```
L+=760.72
z+=-1518.28
Visited 101 nodes.
Executed in 56.734375 CPU*seconds.
```

5.4 Solutions

[30]:
```
for solver in acsObj:
    print(solver.Tplus[-1].nodes)
```

```
[0, 45, 51]
[0, 31, 10, 37, 4, 48, 8, 49, 15, 1, 28, 20, 33, 29, 9, 38, 32, 44, 14, 43, 36,
16, 3, 46, 45, 50, 26, 47, 5, 13, 24, 17, 11, 41, 40, 18, 39, 12, 23, 22, 6, 25,
7, 30, 21, 27, 2, 34, 35, 19, 51]
[0, 67, 76]
[0, 11, 39, 16, 50, 5, 67, 74, 75, 66, 33, 45, 7, 34, 6, 52, 13, 18, 53, 12, 26,
51, 44, 28, 47, 46, 20, 73, 27, 21, 61, 72, 32, 1, 29, 3, 25, 57, 71, 38, 8, 31,
43, 2, 15, 22, 55, 40, 41, 42, 62, 48, 23, 17, 49, 24, 54, 30, 9, 37, 64, 10,
65, 58, 56, 14, 4, 36, 19, 69, 59, 70, 35, 68, 60, 63, 76]
[0, 7, 44, 16, 83, 4, 59, 82, 17, 51, 88, 5, 93, 58, 98, 95, 94, 96, 91, 36, 97,
99, 90, 84, 92, 60, 15, 43, 13, 41, 86, 1, 56, 14, 42, 37, 85, 12, 57, 39, 52,
100, 26, 27, 25, 11, 79, 67, 76, 2, 78, 32, 80, 8, 50, 19, 29, 69, 30, 87, 6,
81, 47, 46, 35, 48, 10, 18, 61, 9, 89, 62, 31, 65, 70, 34, 33, 77, 28, 23, 53,
```

```
54, 24, 38, 55, 74, 73, 71, 72, 40, 21, 20, 3, 22, 66, 75, 49, 68, 101]
[0, 66, 38, 55, 74, 73, 71, 72, 20, 39, 57, 52, 100, 26, 27, 25, 11, 79, 67, 76,
2, 78, 32, 80, 8, 50, 19, 29, 69, 30, 87, 6, 47, 81, 7, 44, 16, 83, 4, 59, 82,
17, 51, 88, 5, 93, 12, 94, 96, 91, 58, 98, 95, 92, 84, 97, 36, 99, 90, 15, 60,
85, 43, 13, 41, 86, 1, 56, 14, 42, 40, 21, 22, 3, 54, 24, 53, 23, 28, 77, 33,
34, 70, 65, 64, 31, 89, 62, 10, 18, 46, 45, 35, 48, 63, 61, 9, 68, 49, 75, 101]
```

References

[1] Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation* 1(1):53–66.