CSCI 5654 - Linear Programming - Project

Team Members

1. Ketan Ramesh

2. Shreyas Gopalakrishna

Vehicle Routing Problem

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance
import random
import pulp
import gurobipy as gp
from gurobipy import GRB
```

Vehicle Routing with capacity

Link for paper: https://arxiv.org/pdf/1606.01935.pdf (https://arxiv.org/pdf/1606.01935.pdf)

The below code solves the capacited vehicle routing problem by following the methodology mentioned in the above cited paper.

The formulation is as follows:

Capacity Vehicle Routing Problem - Model 1

Paper Link - A generalized formulation for vehicle routing problems (https://arxiv.org/pdf/1606.01935.pdf)

The code provided below creates a `class` to frame and solve the Capacity Vehicle Routing Problem (CVRP) using the two-index flow formulation as mentioned in the paper. The formulation of the problem as a Mixed ILP is as follows:

$$min \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} c_{ij} x_{ij}$$

$$min \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} c_{ij} x_{ij}$$

$$s.t. \sum_{j=1, j \neq i}^{n+1} x_{ij} = 1, i = 1, \ldots, n, -(1.1)$$

$$s.t. \sum_{j=1, j \neq i}^{n+1} x_{ij} = 1, i = 1, \ldots, n, -(1.1)$$

$$\sum_{i=0, i \neq h}^{n} x_{ih} - \sum_{j=1, j \neq h}^{n+1} x_{hj} = 0, h = 1, \ldots, n, -(1.2)$$

$$\sum_{i=0, i \neq h}^{n} x_{ih} - \sum_{j=1, j \neq h}^{n+1} x_{hj} = 0, h = 1, \ldots, n, -(1.2)$$

$$\sum_{j=1}^{n} x_{0j} \leq K - (1.3)$$

$$\sum_{j=1}^{n} x_{0j} \leq K - (1.3)$$

$$y_j \geq y_i + q_j x_{ij} - Q(1 - x_{ij}), i, j = 0, \ldots, n+1, -(1.4)$$

$$y_j \geq y_i + q_j x_{ij} - Q(1 - x_{ij}), i, j = 0, \ldots, n+1, -(1.4)$$

$$q_i \leq y_i \leq Q, i = 0, \ldots, n+1, -(1.5)$$

$$q_i \leq y_i \leq Q, i = 0, \ldots, n+1, -(1.5)$$

$$x_{ij} = \{0, 1\}, i, j = 0, \ldots, n + 1.$$

$$x_{ij} = \{0, 1\}, i, j = 0, \ldots, n + 1.$$

where,

N customers are denoted by 1, 2, ... n. The depot is denoted as 0 and n+1 respectively.
K = Number of vehicles.
Q = Capacity of the vehicle.
q_i = Demand of customer i.
c_{ij} = Cost incurred on traveling from i to j.

Decision Variables:

1. x_{ij} = {**1**, if edge (i, j) present in any route. **0**, otherwise}.
2. y_j = Cummulative demand on route that visits node j. A continuous non-negative variable.

The objective is to minimize the sum of the distances of all routes that satisfy the given constraints. The constraint (1.1) is to ensure that all customer nodes are visited exactly once (each node *i* visits only one other node *j*). The constraint (1.2) ensures that if a node is visited, it must depart from the same node (node *h* visited by only one other node *i* and in turn visits only one other node *j*). The constraint (1.3) is used to limit number of different routes/number of vehicles from the depot. The constraints (1.4) and (1.5) is used to eliminate sub-tours and force each route to pass through/include the depot in it.

In [8]:

```
# PuLP class for vehicle routing

class CVRP_GUROBI:
    def __init__(self, numberOfCustomers, numberOfVehicles, capacit
        self.numberOfCustomers = numberOfCustomers
        self.numberOfVehicles  = numberOfVehicles
        self.capacityOfVehicle = capacityOfVehicle
        self.demandOfCustomers = demandOfCustomers
        self.costMatrix        = costMatrix
        self.initializeLP()

    def initializeLP(self):
        self.cvrpLP = gp.Model("CVRP")
        objective = None
        x,y = [], []
        constraint1 = None

        # objective function and variables
        for i in range(len(costMatrix)): #adding depot
            xTemp1 = []
            for j in range(len(costMatrix)):
                if(i != j):
                    xTemp2 = self.cvrpLP.addVar(name='x('+str(i)+',
                    xTemp1.append(xTemp2)
                    objective += xTemp2 * costMatrix[i][j]
                else:
                    xTemp1.append(None)
            x.append(xTemp1)
        self.cvrpLP.setObjective(objective, GRB.MINIMIZE)

        for i in range(len(costMatrix)):
            y.append(self.cvrpLP.addVar(name='y'+str(i),vtype=GRB.C


        # constraints
        # ensure that all customers are visited exactly once
        for i in range(1, len(costMatrix)-1): #adding depot
            constraint1 = None
            for j in range(1, len(costMatrix)):
                if(i != j):
                    if(constraint1 == None):
                        constraint1 = x[i][j]
                    else:
                        constraint1 = constraint1 + x[i][j]
            self.cvrpLP.addConstr(constraint1 == 1)

        # limits the maximum number of routes to the number of vehi
        constraint2 = None
        for j in range(1, len(costMatrix)-1): #not include depot
            if(constraint2 == None):
                constraint2 = x[0][j]
            else:
                constraint2 = constraint2 + x[0][j]
        self.cvrpLP.addConstr(constraint2 <= self.numberOfVehicles)

        # limits the maximum number of routes to the number of vehi
        # constraint2 = None
        # for j in range(1, len(costMatrix) - 1): #not include depo
        #     constraint2 += x[j][-1]
        # self.cvrpLP += constraint2 <= self.numberOfVehicles
        # ensure together that the vehicle capacity is not exceeded
        for i in range(len(costMatrix)):
            constarint3a, constarint3b  = None, None
            constarint3a = self.demandOfCustomers[i] <= y[i]
            constarint3b = y[i] <= self.capacityOfVehicle
            self.cvrpLP.addConstr(constarint3a)
            self.cvrpLP.addConstr(constarint3b)

        # ensure together that the vehicle capacity is not exceeded
        for i in range(len(costMatrix)): #adding depot
            for j in range(len(costMatrix)):
                constraint4 = None
                if(i != j):
                    constraint4 = y[j] >= y[i] + self.demandOfCuston
                    self.cvrpLP.addConstr(constraint4)

        #guarantee the correct flow of vehicles through the arcs, by
        #then it must depart from this node
        for h in range(1, len(costMatrix)-1):
            constraint5a, constraint5b = None, None
            for i in range(0, len(costMatrix)-1):
                if(i != h):
                    if(constraint5a == None):
                        constraint5a = x[i][h]
                    else:
                        constraint5a = constraint5a + x[i][h]
            for j in range(1, len(costMatrix)):
                if(j != h):
                    if(constraint5b == None):
                        constraint5b = x[h][j]
                    else:
                        constraint5b = constraint5b + x[h][j]
            self.cvrpLP.addConstr(constraint5a - constraint5b == 0)

        # print(self.cvrpLP)

    def solve(self):
        status = self.cvrpLP.optimize()
        print(status)

    def getResult(self):
        print("Objective value: ", self.cvrpLP.ObjVal)
        for v in self.cvrpLP.getVars():
            print(v.varName, " = ", v.x)
        return self.cvrpLP
```

TESTING - RANDOM DATASET

In [9]:

```
numberOfCustomers = 5
capacityOfVehicle = 10
numberOfVehicles = 3
C = [i for i in range(1, numberOfCustomers+1)] #set of customers
V = [0] + C + [numberOfCustomers+1] #depot + customer nodes
demandOfCustomers = {1: 1, 2: 5, 3: 8, 4: 5, 5: 5, 0: 0, 6: 0}
# demandOfCustomers[0] = 0
# demandOfCustomers[numberOfCustomers+1] = 0

# Creating random coordinates
xCoordinates = [30, 20, 10, 10, 40, 50, 30]
yCoordinates = [30, 40, 45, 30, 10, 30, 30]

# Cost matrix
costMatrix = np.ndarray(shape=(len(V),len(V)))
for i in range(len(V)):
    for j in range(len(V)):
        if(i == 0 and j == len(V)-1):
            costMatrix[i][j] = 0
            continue

        if(j == 0 and i == len(V)-1):
            costMatrix[i][j] = 0
            continue

        if(i!=j):
            costMatrix[i][j] = int(distance.euclidean([xCoordinates
        else:
            costMatrix[i][j] = 0
print(costMatrix)
print(demandOfCustomers)
```

```
[[ 0. 14. 25. 20. 22. 20.  0.]
 [14.  0. 11. 14. 36. 31. 14.]
 [25. 11.  0. 15. 46. 42. 25.]
 [20. 14. 15.  0. 36. 40. 20.]
 [22. 36. 46. 36.  0. 22. 22.]
 [20. 31. 42. 40. 22.  0. 20.]
 [ 0. 14. 25. 20. 22. 20.  0.]]
{1: 1, 2: 5, 3: 8, 4: 5, 5: 5, 0: 0, 6: 0}
```

In [10]:

```
lp = CVRP_GUROBI(numberOfCustomers, numberOfVehicles, capacityOfVeh
lp.solve()
result = lp.getResult()
```

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 67 rows, 49 columns and 220 nonz
eros
Model fingerprint: 0x14c26846
Variable types: 7 continuous, 42 integer (42 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [1e+01, 5e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+01]
Found heuristic solution: objective 154.0000000
Presolve removed 48 rows and 30 columns
Presolve time: 0.00s
Presolved: 19 rows, 19 columns, 82 nonzeros
Variable types: 5 continuous, 14 integer (14 binary)

Root relaxation: cutoff, 7 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Boun
ds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestB
d   Gap | It/Node Time

     0     0     cutoff    0       154.00000  154.0000
0  0.00%     -    0s

Explored 0 nodes (7 simplex iterations) in 0.01 second
s
Thread count was 12 (of 12 available processors)

Solution count 1: 154
```

```
Optimal solution found (tolerance 1.00e-04)
Best objective 1.540000000000e+02, best bound 1.540000
000000e+02, gap 0.0000%
None
Objective value:  154.0
x(0,1)  =  1.0
x(0,2)  =  0.0
x(0,3)  =  1.0
x(0,4)  =  0.0
x(0,5)  =  1.0
x(0,6)  =  0.0
x(1,0)  =  0.0
x(1,2)  =  1.0
x(1,3)  =  0.0
x(1,4)  =  0.0
x(1,5)  =  0.0
x(1,6)  =  0.0
x(2,0)  =  0.0
x(2,1)  =  0.0
x(2,3)  =  0.0
x(2,4)  =  0.0
x(2,5)  =  0.0
x(2,6)  =  1.0
x(3,0)  =  0.0
x(3,1)  =  0.0
x(3,2)  =  0.0
x(3,4)  =  0.0
x(3,5)  =  0.0
x(3,6)  =  1.0
x(4,0)  =  0.0
x(4,1)  =  0.0
x(4,2)  =  0.0
x(4,3)  =  0.0
x(4,5)  =  0.0
x(4,6)  =  1.0
x(5,0)  =  0.0
x(5,1)  =  0.0
x(5,2)  =  0.0
x(5,3)  =  0.0
x(5,4)  =  1.0
x(5,6)  =  0.0
x(6,0)  =  0.0
x(6,1)  =  0.0
x(6,2)  =  0.0
```

```
x(6,3)  =  0.0
x(6,4)  =  0.0
x(6,5)  =  0.0
y0  =  0.0
y1  =  1.0
y2  =  6.0
y3  =  8.0
y4  =  10.0
y5  =  5.0
y6  =  10.0
```

In [13]:

```
# Use output to get path
# gets variables as (x,y) coordinates
variables = []
for v in result.getVars():
    if('x' in v.varName and v.x == 1):
        # print(v.name, " = ", v.varValue)
        temp = (v.varName.split('(')[1].split(')')[0].split(','))
        variables.append((int(temp[0]),int(temp[1])))
# print(variables)

# recursive calls for getting the path
def recursiveList(start, L, X):
    for item in L:
        if(item[0] == start):
            X.append(item)
            return recursiveList(item[1], L, X)
    return X

pathList = []
setList = []
start = 0
for v in variables:
    if(v[0] == start):
        path = recursiveList(v[1], variables, [v])
        print(path)
        pathList.append(path)
        set1 = []
        for i in path:
            set1.append(i[0])
            set1.append(i[1])
        setList.append(list(set(set1)))
```

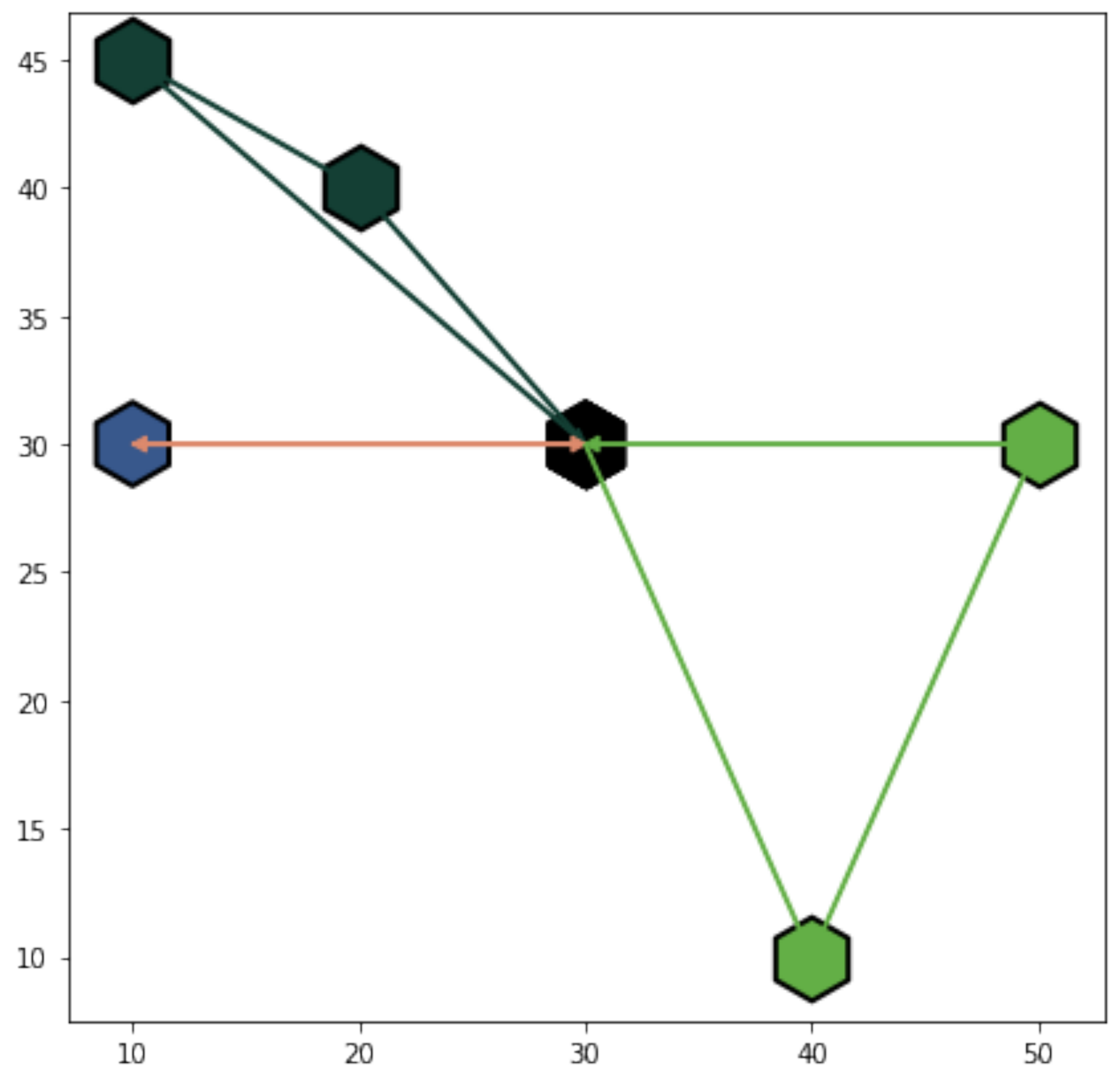
```
[(0, 1), (1, 2), (2, 6)]
[(0, 3), (3, 6)]
[(0, 5), (5, 4), (4, 6)]
```

In [14]:

```
# Visualization
plt.figure(figsize=(7,7))
plt.rc('xtick',labelsize=10)
plt.rc('ytick',labelsize=10)
plt.rc('ytick',labelsize=16)

coordinateList = []
for s in setList:
    coordinate = []
    for j in s:
        coordinate.append([float(xCoordinates[j]), float(yCoordinat
    coordinateList.append(coordinate)
# print(coordinateList)

def addToPlot(L):
    x_val = [x[0] for x in L]
    y_val = [x[1] for x in L]

    r = random.random()
    b = random.random()
    g = random.random()
    newColor = (r, g, b)

    plt.scatter(x_val,y_val, c=newColor, edgecolor='black', linewid
    ax = plt.axes()

    length = len(L)-1

    for i in range(length):
        ax.arrow(L[i][0],  #x1
                    L[i][1],  # y1
                    L[i+1][0]-L[i][0], # x2 - x1
                    L[i+1][1]-L[i][1], # y2 - y1
                    width=0.1, head_width=0.6, head_length=0.6, col
                    )

    # ax.arrow(L[-1][0],  #x1
    #                 L[-1][1],  # y1
    #                 L[0][0]-L[-1][0], # x2 - x1
    #                 L[0][1]-L[-1][1], # y2 - y1
    #                 width=0.1, head_width=0.6, head_length=0.6, c
    #                 )
for i in range(len(coordinateList)):
    addToPlot(coordinateList[i])
plt.scatter(coordinateList[0][0][0],coordinateList[0][0][1], c='bla
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
```

```
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.
/Users/shreyas/opt/anaconda3/lib/python3.7/site-packag
es/ipykernel_launcher.py:24: MatplotlibDeprecationWarn
ing: Adding an axes using the same arguments as a prev
ious axes currently reuses the earlier instance.  In a
future version, a new instance will always be created
and returned.  Meanwhile, this warning can be suppress
ed, and the future behavior ensured, by passing a uniq
ue label to each axes instance.
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.
```



TESTING WITH PRE_DEFINED DATA N=15

P-n16-k8.vrp - Augerat et al. Set - P

In [15]:

```
# Data Source:
numberOfCustomers = 15
capacityOfVehicle = 35
numberOfVehicles = 8
C = [i for i in range(1, numberOfCustomers+1)] #set of customers
V = [0] + C + [numberOfCustomers+1] #depot + customer nodes
demandOfCustomers = {0:0,1:19,2:30,3:16,4:23,5:11,6:31,7:15,8:28,9:
# demandOfCustomers[0] = 0
demandOfCustomers[numberOfCustomers+1] = 0

xCoordinates = [30,37,49,52,31,52,42,52,57,62,42,27,43,58,58,37,30]
yCoordinates = [40,52,49,64,62,33,41,41,58,42,57,68,67,48,27,69,40]

# Cost matrix
costMatrix = np.ndarray(shape=(len(V),len(V)))
for i in range(len(V)):
    for j in range(len(V)):
        if(i == 0 and j == len(V)-1):
            costMatrix[i][j] = 0
            continue

        if(j == 0 and i == len(V)-1):
            costMatrix[i][j] = 0
            continue

        if(i!=j):
            costMatrix[i][j] = int(distance.euclidean([xCoordinates
        else:
            costMatrix[i][j] = 0
costMatrix
```

Out[15]:

```
array([[ 0., 13., 21., 32., 22., 23., 12., 22., 32., 3
2., 20., 28., 29.,
         29., 30., 29.,  0.],
        [13.,  0., 12., 19., 11., 24., 12., 18., 20., 2
6.,  7., 18., 16.,
         21., 32., 17., 13.],
        [21., 12.,  0., 15., 22., 16., 10.,  8., 12., 1
4., 10., 29., 18.,
          9., 23., 23., 21.],
        [32., 19., 15.,  0., 21., 31., 25., 23.,  7., 2
4., 12., 25.,  9.,
         17., 37., 15., 32.],
        [22., 11., 22., 21.,  0., 35., 23., 29., 26., 3
6., 12.,  7., 13.,
         30., 44.,  9., 22.],
        [23., 24., 16., 31., 35.,  0., 12.,  8., 25., 1
3., 26., 43., 35.,
         16.,  8., 39., 23.],
        [12., 12., 10., 25., 23., 12.,  0., 10., 22., 2
0., 16., 30., 26.,
         17., 21., 28., 12.],
        [22., 18.,  8., 23., 29.,  8., 10.,  0., 17., 1
0., 18., 36., 27.,
          9., 15., 31., 22.],
        [32., 20., 12.,  7., 26., 25., 22., 17.,  0., 1
6., 15., 31., 16.,
         10., 31., 22., 32.],
        [32., 26., 14., 24., 36., 13., 20., 10., 16.,
0., 25., 43., 31.,
          7., 15., 36., 32.],
        [20.,  7., 10., 12., 12., 26., 16., 18., 15., 2
5.,  0., 18., 10.,
         18., 34., 13., 20.],
        [28., 18., 29., 25.,  7., 43., 30., 36., 31., 4
3., 18.,  0., 16.,
         36., 51., 10., 28.],
        [29., 16., 18.,  9., 13., 35., 26., 27., 16., 3
1., 10., 16.,  0.,
         24., 42.,  6., 29.],
        [29., 21.,  9., 17., 30., 16., 17.,  9., 10.,
7., 18., 36., 24.,
          0., 21., 29., 29.],
        [30., 32., 23., 37., 44.,  8., 21., 15., 31., 1
5., 34., 51., 42.,
         21.,  0., 46., 30.],
        [29., 17., 23., 15.,  9., 39., 28., 31., 22., 3
6., 13., 10.,  6.,
         29., 46.,  0., 29.],
        [ 0., 13., 21., 32., 22., 23., 12., 22., 32., 3
2., 20., 28., 29.,
         29., 30., 29.,  0.]])
```

In [17]:

```
lp = CVRP_GUROBI(numberOfCustomers, numberOfVehicles, capacityOfVeh
lp.solve()
result = lp.getResult()
```

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac6
4)
Optimize a model with 337 rows, 289 columns and 1540
nonzeros
Model fingerprint: 0x464e5be6
Variable types: 17 continuous, 272 integer (272 bina
ry)
Coefficient statistics:
  Matrix range     [1e+00, 7e+01]
  Objective range  [6e+00, 5e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 4e+01]
Presolve removed 186 rows and 130 columns
Presolve time: 0.00s
Presolved: 151 rows, 159 columns, 1316 nonzeros
Variable types: 13 continuous, 146 integer (146 bina
ry)

Root relaxation: objective 2.482569e+02, 103 iterati
```

In [18]:

```
# Use output to get path
# gets variables as (x,y) coordinates
variables = []
for v in result.getVars():
    if('x' in v.varName and v.x == 1):
        # print(v.name, " = ", v.varValue)
        temp = (v.varName.split('(')[1].split(')')[0].split(','))
        variables.append((int(temp[0]),int(temp[1])))
# print(variables)

# recursive calls for getting the path
```

```
def recursiveList(start, L, X):
    for item in L:
        if(item[0] == start):
            X.append(item)
            return recursiveList(item[1], L, X)
    return X

pathList = []
setList = []
start = 0
for v in variables:
    if(v[0] == start):
        path = recursiveList(v[1], variables, [v])
        print(path)
        pathList.append(path)
        set1 = []
        for i in path:
            set1.append(i[0])
            set1.append(i[1])
        a = list(set(set1))
        setList.append(sorted(a))
print(setList)
```

```
[(0, 1), (1, 16)]
[(0, 2), (2, 16)]
[(0, 3), (3, 9), (9, 5), (5, 16)]
[(0, 6), (6, 16)]
[(0, 10), (10, 12), (12, 15), (15, 16)]
[(0, 11), (11, 4), (4, 16)]
[(0, 13), (13, 8), (8, 16)]
[(0, 14), (14, 7), (7, 16)]
[[0, 1, 16], [0, 2, 16], [0, 3, 5, 9, 16], [0, 6, 16],
[0, 10, 12, 15, 16], [0, 4, 11, 16], [0, 8, 13, 16], [
0, 7, 14, 16]]
```

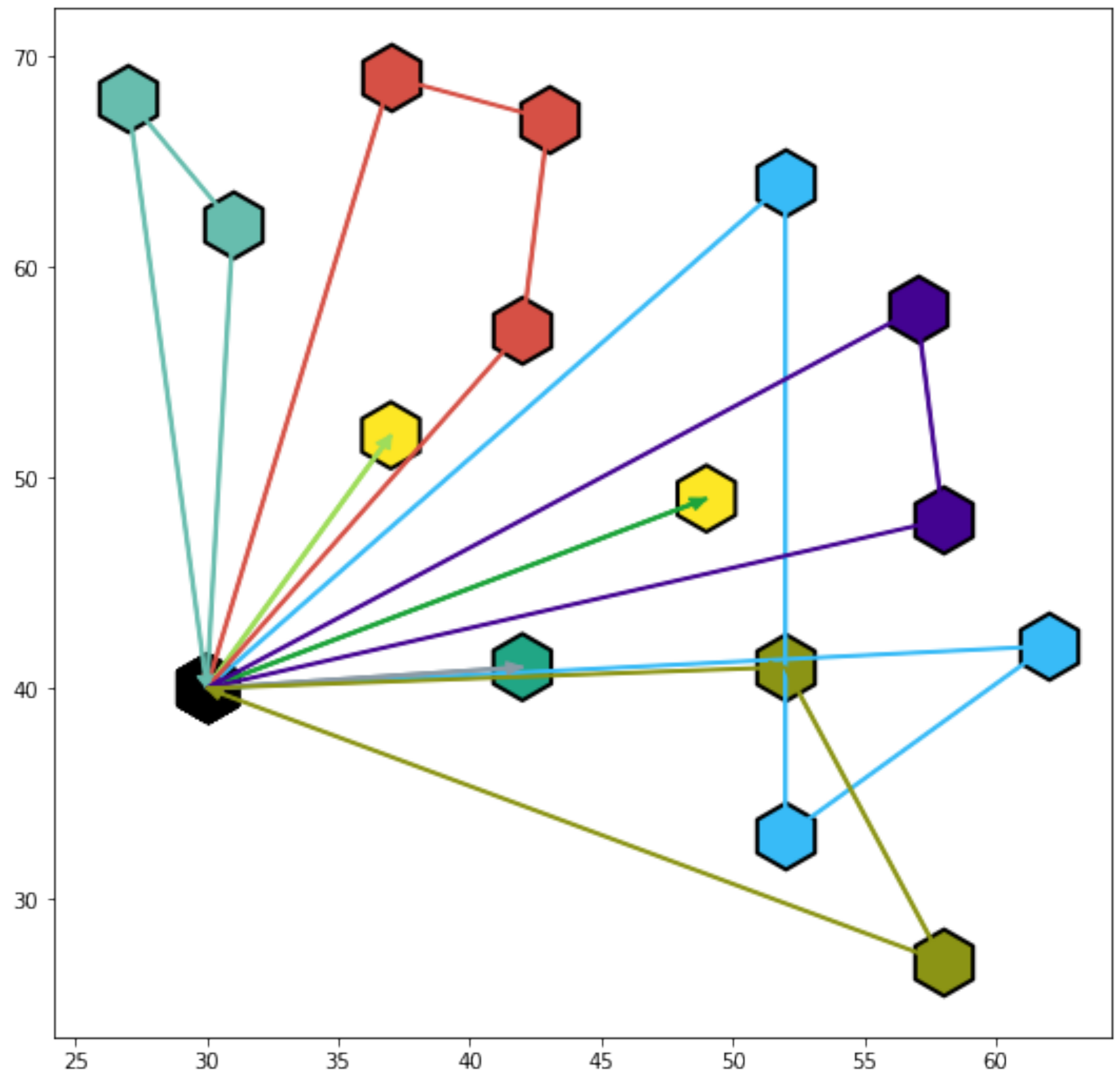
In [19]:

```
# Visualization
plt.figure(figsize=(9,9))
plt.rc('xtick',labelsize=10)
plt.rc('ytick',labelsize=10)

coordinateList = []
for s in setList:
    coordinate = []
    for j in s:
        coordinate.append([float(xCoordinates[j]), float(yCoordinat
    coordinateList.append(coordinate)
# print(coordinateList)

def addToPlot(L):
    x_val = [x[0] for x in L]
    y_val = [x[1] for x in L]

    r = random.random()
    b = random.random()
    g = random.random()
    newColor = (r, g, b)

    plt.scatter(x_val,y_val, c=newColor, edgecolor='black', linewid
    ax = plt.axes()

    length = len(L)-1

    for i in range(length):
        ax.arrow(L[i][0],  #x1
                    L[i][1],  # y1
                    L[i+1][0]-L[i][0], # x2 - x1
                    L[i+1][1]-L[i][1], # y2 - y1
                    width=0.1, head_width=0.6, head_length=0.6, col
                    )

    # ax.arrow(L[-1][0],  #x1
    #                 L[-1][1],  # y1
    #                 L[0][0]-L[-1][0], # x2 - x1
    #                 L[0][1]-L[-1][1], # y2 - y1
    #                 width=0.1, head_width=0.6, head_length=0.6, co
    #                 )
for i in range(len(coordinateList)):
    print(coordinateList[i])
    addToPlot(coordinateList[i])
plt.scatter(coordinateList[0][0][0],coordinateList[0][0][1], c='bla
plt.show()
```

```
/Users/shreyas/opt/anaconda3/lib/python3.7/site-packag
es/ipykernel_launcher.py:24: MatplotlibDeprecationWarn
ing: Adding an axes using the same arguments as a prev
ious axes currently reuses the earlier instance.  In a
```

```
future version, a new instance will always be created
and returned.  Meanwhile, this warning can be suppress
ed, and the future behavior ensured, by passing a uniq
ue label to each axes instance.
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.


[[30.0, 40.0], [37.0, 52.0], [30.0, 40.0]]
[[30.0, 40.0], [49.0, 49.0], [30.0, 40.0]]
[[30.0, 40.0], [52.0, 64.0], [52.0, 33.0], [62.0, 42.0
], [30.0, 40.0]]
[[30.0, 40.0], [42.0, 41.0], [30.0, 40.0]]
[[30.0, 40.0], [42.0, 57.0], [43.0, 67.0], [37.0, 69.0
], [30.0, 40.0]]
[[30.0, 40.0], [31.0, 62.0], [27.0, 68.0], [30.0, 40.0
]]
```

```
[[30.0, 40.0], [57.0, 58.0], [58.0, 48.0], [30.0, 40.0
]]
[[30.0, 40.0], [52.0, 41.0], [58.0, 27.0], [30.0, 40.0
]]
```



TESTING WITH PRE_DEFINED DATA N=20

P-n20-k2.vrp - Augerat et al. Set - P

In [23]:

```
numberOfCustomers = 19
capacityOfVehicle = 160
numberOfVehicles = 2
C = [i for i in range(1, numberOfCustomers+1)] #set of customers
```

```
C = [i for i in range(1, numberOfCustomers+1)] #set of customers
V = [0] + C + [numberOfCustomers+1] #depot + customer nodes
demandOfCustomers = [0, 19, 30, 16, 23, 11, 31, 15, 28, 8, 8, 7, 14
# demandOfCustomers[0] = 0
# demandOfCustomers[numberOfCustomers+1] = 0

xCoordinates = [30, 37, 49, 52, 31, 52, 42, 52, 57, 62, 42, 27, 43,
yCoordinates = [40, 52, 49, 64, 62, 33, 41, 41, 58, 42, 57, 68, 67,

# Cost matrix
costMatrix = np.ndarray(shape=(len(V),len(V)))
for i in range(len(V)):
    for j in range(len(V)):
        if(i == 0 and j == len(V)-1):
            costMatrix[i][j] = 0
            continue

        if(j == 0 and i == len(V)-1):
            costMatrix[i][j] = 0
            continue

        if(i!=j):
            costMatrix[i][j] = int(distance.euclidean([xCoordinates
        else:
            costMatrix[i][j] = 0
costMatrix
```

Out[23]:

```
array([[ 0., 13., 21., 32., 22., 23., 12., 22., 32., 3
2., 20., 28., 29.,
        29., 30., 29., 31., 39., 43., 15.,  0.],
       [13.,  0., 12., 19., 11., 24., 12., 18., 20., 2
6.,  7., 18., 16.,
        21., 32., 17., 30., 27., 31., 18., 13.],
       [21., 12.,  0., 15., 22., 16., 10.,  8., 12., 1
4., 10., 29., 18.,
         9., 23., 23., 20., 19., 24., 14., 21.],
       [32., 19., 15.,  0., 21., 31., 25., 23.,  7., 2
4., 12., 25.,  9.,
        17., 37., 15., 32., 10., 12., 29., 32.],
       [22., 11., 22., 21.,  0., 35., 23., 29., 26., 3
6., 12.,  7., 13.,
        30., 44.,  9., 41., 31., 32., 30., 22.],
       [23., 24., 16., 31., 35.,  0., 12.,  8., 25., 1
```

```
3., 26., 43., 35.,
         16.,  8., 39.,  9., 31., 37.,  7., 23.],
        [12., 12., 10., 25., 23., 12.,  0., 10., 22., 2
0., 16., 30., 26.,
         17., 21., 28., 20., 29., 35.,  6., 12.],
        [22., 18.,  8., 23., 29.,  8., 10.,  0., 17., 1
0., 18., 36., 27.,
          9., 15., 31., 12., 24., 30.,  9., 22.],
        [32., 20., 12.,  7., 26., 25., 22., 17.,  0., 1
6., 15., 31., 16.,
         10., 31., 22., 25.,  7., 12., 25., 32.],
        [32., 26., 14., 24., 36., 13., 20., 10., 16.,
0., 25., 43., 31.,
          7., 15., 36.,  9., 21., 27., 18., 32.],
        [20.,  7., 10., 12., 12., 26., 16., 18., 15., 2
5.,  0., 18., 10.,
         18., 34., 13., 30., 20., 24., 22., 20.],
        [28., 18., 29., 25.,  7., 43., 30., 36., 31., 4
3., 18.,  0., 16.,
         36., 51., 10., 48., 35., 36., 37., 28.],
        [29., 16., 18.,  9., 13., 35., 26., 27., 16., 3
1., 10., 16.,  0.,
         24., 42.,  6., 38., 19., 20., 32., 29.],
        [29., 21.,  9., 17., 30., 16., 17.,  9., 10.,
7., 18., 36., 24.,
          0., 21., 29., 15., 15., 21., 18., 29.],
        [30., 32., 23., 37., 44.,  8., 21., 15., 31., 1
5., 34., 51., 42.,
         21.,  0., 46.,  6., 36., 42., 15., 30.],
        [29., 17., 23., 15.,  9., 39., 28., 31., 22., 3
6., 13., 10.,  6.,
         29., 46.,  0., 43., 25., 26., 34., 29.],
        [31., 30., 20., 32., 41.,  9., 20., 12., 25.,
9., 30., 48., 38.,
         15.,  6., 43.,  0., 30., 36., 16., 31.],
        [39., 27., 19., 10., 31., 31., 29., 24.,  7., 2
1., 20., 35., 19.,
         15., 36., 25., 30.,  0.,  6., 32., 39.],
        [43., 31., 24., 12., 32., 37., 35., 30., 12., 2
7., 24., 36., 20.,
         21., 42., 26., 36.,  6.,  0., 38., 43.],
        [15., 18., 14., 29., 30.,  7.,  6.,  9., 25., 1
8., 22., 37., 32.,
         18., 15., 34., 16., 32., 38.,  0., 15.],
        [ 0., 13., 21., 32., 22., 23., 12., 22., 32., 3
```

```
2., 20., 28., 29.,
        29., 30., 29., 31., 39., 43., 15.,  0.]])
```

In [24]:

```
lp = CVRP_GUROBI(numberOfCustomers, numberOfVehicles, capacityOfVeh
lp.solve()
result = lp.getResult()
```

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac6
4)
Optimize a model with 501 rows, 441 columns and 2404
nonzeros
Model fingerprint: 0xa6550ad2
Variable types: 21 continuous, 420 integer (420 bina
ry)
Coefficient statistics:
  Matrix range     [1e+00, 2e+02]
  Objective range  [6e+00, 5e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+02]
Presolve removed 120 rows and 42 columns
Presolve time: 0.01s
Presolved: 381 rows, 399 columns, 1767 nonzeros
Variable types: 19 continuous, 380 integer (380 bina
ry)

Root relaxation: objective 1.342688e+02, 69 iteratio
```

In [25]:

```python
# Use output to get path
# gets variables as (x,y) coordinates
variables = []
for v in result.getVars():
    if('x' in v.varName and v.x == 1):
        # print(v.name, " = ", v.varValue)
        temp = (v.varName.split('(')[1].split(')')[0].split(','))
        variables.append((int(temp[0]),int(temp[1])))
# print(variables)

# recursive calls for getting the path
def recursiveList(start, L, X):
    for item in L:
        if(item[0] == start):
            X.append(item)
            return recursiveList(item[1], L, X)
    return X

pathList = []
setList = []
start = 0
for v in variables:
    if(v[0] == start):
        path = recursiveList(v[1], variables, [v])
        print(path)
        pathList.append(path)
        set1 = []
        for i in path:
            set1.append(i[0])
            set1.append(i[1])
        a = list(set(set1))
        setList.append(sorted(a))
print(setList)
```

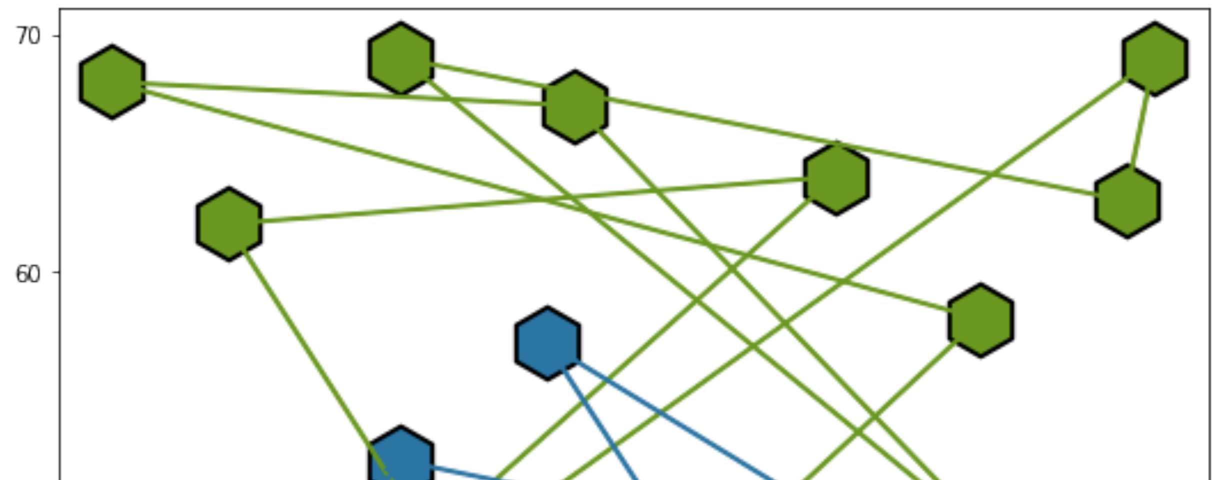
```
[(0, 6), (6, 13), (13, 8), (8, 17), (17, 18), (18, 3),
(3, 12), (12, 15), (15, 11), (11, 4), (4, 20)]
[(0, 19), (19, 5), (5, 14), (14, 16), (16, 9), (9, 7),
(7, 2), (2, 10), (10, 1), (1, 20)]
[[0, 3, 4, 6, 8, 11, 12, 13, 15, 17, 18, 20], [0, 1, 2
, 5, 7, 9, 10, 14, 16, 19, 20]]
```

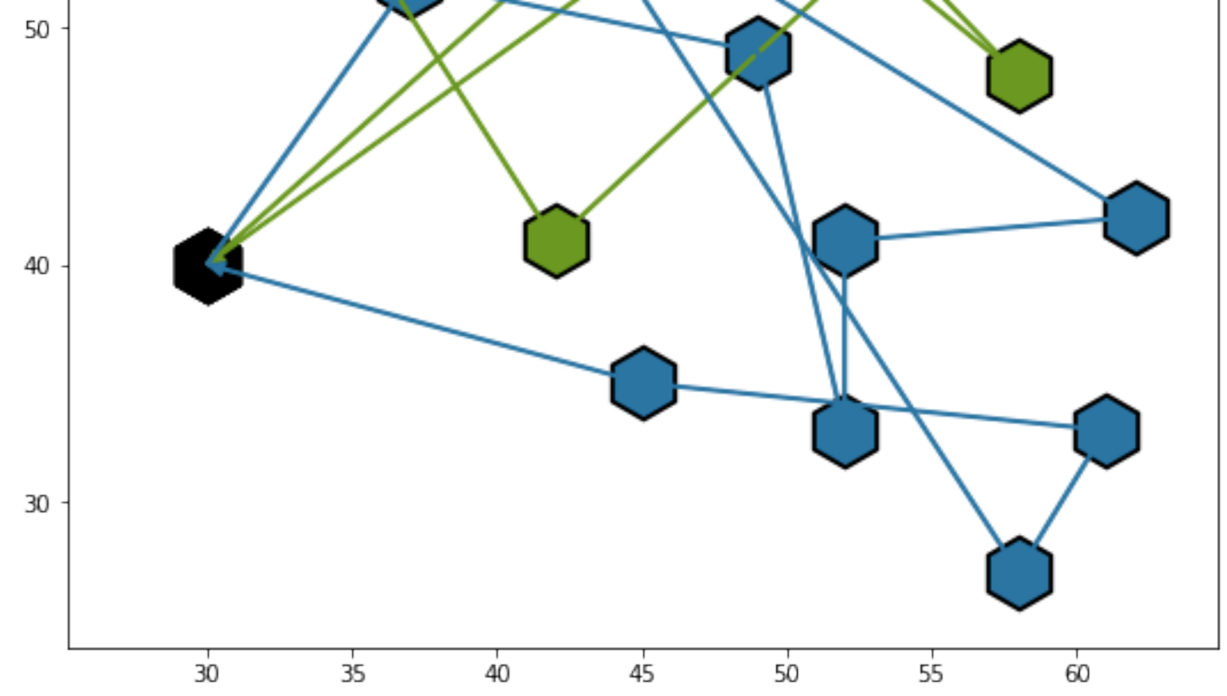
In [26]:

In [26]:

```
# Visualization
plt.figure(figsize=(9,9))
plt.rc('xtick',labelsize=10)
plt.rc('ytick',labelsize=10)

coordinateList = []
for s in setList:
    coordinate = []
    for j in s:
        coordinate.append([float(xCoordinates[j]), float(yCoordinat
    coordinateList.append(coordinate)
# print(coordinateList)

def addToPlot(L):
    x_val = [x[0] for x in L]
    y_val = [x[1] for x in L]

    r = random.random()
    b = random.random()
    g = random.random()
    newColor = (r, g, b)

    plt.scatter(x_val,y_val, c=newColor, edgecolor='black', linewid
    ax = plt.axes()

    length = len(L)-1

    for i in range(length):
        ax.arrow(L[i][0],  #x1
                    L[i][1],  # y1
                    L[i+1][0]-L[i][0], # x2 - x1
                    L[i+1][1]-L[i][1], # y2 - y1
                    width=0.1, head_width=0.6, head_length=0.6, col
                    )

    # ax.arrow(L[-1][0],  #x1
    #                 L[-1][1],  # y1
    #                 L[0][0]-L[-1][0], # x2 - x1
    #                 L[0][1]-L[-1][1], # y2 - y1
    #                 width=0.1, head_width=0.6, head_length=0.6, c
    #                 )
for i in range(len(coordinateList)):
    print(coordinateList[i])
    addToPlot(coordinateList[i])
    addPlot(coordinateList[1])
plt.scatter(coordinateList[0][0][0],coordinateList[0][0][1], c='bla
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.
/Users/shreyas/opt/anaconda3/lib/python3.7/site-packag
es/ipykernel_launcher.py:24: MatplotlibDeprecationWarn
ing: Adding an axes using the same arguments as a prev
ious axes currently reuses the earlier instance.  In a
future version, a new instance will always be created
and returned.  Meanwhile, this warning can be suppress
ed, and the future behavior ensured, by passing a uniq
ue label to each axes instance.
'c' argument looks like a single numeric RGB or RGBA s
equence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' &
'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for
all points.


[[30.0, 40.0], [52.0, 64.0], [31.0, 62.0], [42.0, 41.0
], [57.0, 58.0], [27.0, 68.0], [43.0, 67.0], [58.0, 48
.0], [37.0, 69.0], [62.0, 63.0], [63.0, 69.0], [30.0,
40.0]]
[[30.0, 40.0], [37.0, 52.0], [49.0, 49.0], [52.0, 33.0
], [52.0, 41.0], [62.0, 42.0], [42.0, 57.0], [58.0, 27
.0], [61.0, 33.0], [45.0, 35.0], [30.0, 40.0]]
```





TESTING WITH PRE_DEFINED DATA N=32

A-n32-k5.vrp - Augerat et al. Set - A

```
In [20]:
```

```python
#N-31 CVRP:
numberOfCustomers = 31
capacityOfVehicle = 100
numberOfVehicles = 5
C = [i for i in range(1, numberOfCustomers+1)] #set of customers
V = [0] + C + [numberOfCustomers+1] #depot + customer nodes
demandOfCustomers = {0:0,1:19,2:21,3:6,4:19,5:7,6:12,7:16,8:6,9:16,
# demandOfCustomers[0] = 0
demandOfCustomers[numberOfCustomers+1] = 0



xCoordinates = [82,96,50,49,13,29,58,84,14,2,3,5,98,84,61,1,88,91,1
yCoordinates = [76,44,5,8,7,89,30,39,24,39,82,10,52,25,59,65,51,2,3
# Cost matrix
costMatrix = np.ndarray(shape=(len(V),len(V)))
for i in range(len(V)):
    for j in range(len(V)):
        if(i == 0 and j == len(V)-1):
            costMatrix[i][j] = 0
            continue
        if(j == 0 and i == len(V)-1):
            costMatrix[i][j] = 0
            continue
        if(i!=j):
            costMatrix[i][j] = int(distance.euclidean([xCoordinates
        else:
            costMatrix[i][j] = 0
costMatrix
```

```
Out[20]:

array([[ 0., 34., 77., ..., 16., 72.,  0.],
       [34.,  0., 60., ..., 19., 39., 34.],
       [77., 60.,  0., ..., 65., 48., 77.],
       ...,
       [16., 19., 65., ...,  0., 56., 16.],
       [72., 39., 48., ..., 56.,  0., 72.],
       [ 0., 34., 77., ..., 16., 72.,  0.]])
```

In [21]:

```
lp = CVRP_GUROBI(numberOfCustomers, numberOfVehicles, capacityOfVeh
lp.solve()
result = lp.getResult()
```

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac6
4)
Optimize a model with 1185 rows, 1089 columns and 61
48 nonzeros
Model fingerprint: 0xf3e5f92b
Variable types: 33 continuous, 1056 integer (1056 bi
nary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [2e+00, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+02]
Presolve removed 192 rows and 66 columns
Presolve time: 0.01s
Presolved: 993 rows, 1023 columns, 4743 nonzeros
Variable types: 31 continuous, 992 integer (992 bina
ry)

Root relaxation: objective 3.528700e+02, 117 iterati
```

In [ ]:

```
# Use output to get path
# gets variables as (x,y) coordinates
variables = []
for v in result.getVars():
    if('x' in v.varName and v.x == 1):
        # print(v.name, " = ", v.varValue)
        temp = (v.varName.split('(')[1].split(')')[0].split(','))
        variables.append((int(temp[0]),int(temp[1])))
# print(variables)

# recursive calls for getting the path
def recursiveList(start, L, X):
    for item in L:
        if(item[0] == start):
            X.append(item)
            return recursiveList(item[1], L, X)
    return X

pathList = []
setList = []
start = 0
for v in variables:
    if(v[0] == start):
        path = recursiveList(v[1], variables, [v])
        print(path)
        pathList.append(path)
        set1 = []
        for i in path:
            set1.append(i[0])
            set1.append(i[1])
        a = list(set(set1))
        setList.append(sorted(a))
print(setList)
```

In [ ]:

```
# Visualization
plt.figure(figsize=(9,9))
plt.rc('xtick',labelsize=10)
plt.rc('ytick',labelsize=10)

coordinateList = []
```

```python
for s in setList:
    coordinate = []
    for j in s:
        coordinate.append([float(xCoordinates[j]), float(yCoordinat
    coordinateList.append(coordinate)
# print(coordinateList)

def addToPlot(L):
    x_val = [x[0] for x in L]
    y_val = [x[1] for x in L]

    r = random.random()
    b = random.random()
    g = random.random()
    newColor = (r, g, b)

    plt.scatter(x_val,y_val, c=newColor, edgecolor='black', linewid
    ax = plt.axes()

    length = len(L)-1

    for i in range(length):
        ax.arrow(L[i][0],  #x1
                    L[i][1],  # y1
                    L[i+1][0]-L[i][0], # x2 - x1
                    L[i+1][1]-L[i][1], # y2 - y1
                    width=0.1, head_width=0.6, head_length=0.6, col
                    )

    # ax.arrow(L[-1][0],  #x1
    #                 L[-1][1],  # y1
    #                 L[0][0]-L[-1][0], # x2 - x1
    #                 L[0][1]-L[-1][1], # y2 - y1
    #                 width=0.1, head_width=0.6, head_length=0.6, c
    #                 )
for i in range(len(coordinateList)):
    print(coordinateList[i])
    addToPlot(coordinateList[i])
plt.scatter(coordinateList[0][0][0],coordinateList[0][0][1], c='bla
plt.show()
```

CSCI 5654 - Linear Programming - Project

Team Members

1. Ketan Ramesh

2. Shreyas Gopalakrishna

Vehicle Routing Problem

In [1]:

```

```

Vehicle Routing with capacity

Link for paper: https://arxiv.org/pdf/1606.01935.pdf (https://arxiv.org/pdf/1606.01935.pdf)

The below code solves the capacited vehicle routing problem by following the methodology mentioned in the above cited paper.

The formulation is as follows:

Capacity Vehicle Routing Problem - Model 1

Paper Link - A generalized formulation for vehicle routing problems (https://arxiv.org/pdf/1606.01935.pdf)
The code provided below creates a `class` to frame and solve the Capacity Vehicle Routing Problem (CVRP) using the two-index flow formulation as mentioned in the paper. The formulation of the problem as a Mixed ILP is as follows:

$$min \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} c_{ij} x_{ij}$$

$$s.t. \sum_{j=1, j \neq i}^{n+1} x_{ij} = 1, i = 1, \dots, n, \ - (1.1)$$

$$\sum_{i=0, i \neq h}^{n} x_{ih} - \sum_{j=1, j \neq h}^{n+1} x_{hj} = 0, h = 1, \dots, n, \ - (1.2)$$

$$\sum_{j=1}^{n} x_{0j} \leq K - (1.3)$$

$$y_j \geq y_i + q_j x_{ij} - Q(1 - x_{ij}), i, j = 0, \dots, n+1, \ - (1.4)$$

$$q_i \leq y_i \leq Q, i = 0, \dots, n+1, \ - (1.5)$$

$$x_{ij} = \{0, 1\}, i, j = 0, \dots, n+1.$$

where,

N customers are denoted by 1, 2, ... n. The depot is denoted as 0 and n+1 respectively.
K = Number of vehicles.
Q = Capacity of the vehicle.
q_i = Demand of customer i.
c_{ij} = Cost incurred on traveling from i to j.

Decision Variables:

1. x_{ij} = {**1**, if edge (i, j) present in any route. **0**, otherwise}.
2. y_j = Cummulative demand on route that visits node j. A continuous non-negative variable.

The objective is to minimize the sum of the distances of all routes that satisfy the given constraints. The constraint (1.1) is to ensure that all customer nodes are visited exactly once (each node *i* visits only one other node *j*). The constraint (1.2) ensures that if a node is visited, it must depart from the same node (node *h* visited by only one other node *i* and in turn visits only one other node *j*). The constraint (1.3) is used to limit number of different routes/number of vehicles from the depot. The constraints (1.4) and (1.5) is used to eliminate sub-tours and force each route to pass through/include the depot in it.

In [8]:

TESTING - RANDOM DATASET

In [9]:

```
[[ 0. 14. 25. 20. 22. 20.  0.]
 [14.  0. 11. 14. 36. 31. 14.]
 [25. 11.  0. 15. 46. 42. 25.]
 [20. 14. 15.  0. 36. 40. 20.]
 [22. 36. 46. 36.  0. 22. 22.]
 [20. 31. 42. 40. 22.  0. 20.]
 [ 0. 14. 25. 20. 22. 20.  0.]]
{1: 1, 2: 5, 3: 8, 4: 5, 5: 5, 0: 0, 6: 0}
```

In [10]:

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac6
4)
Optimize a model with 67 rows, 49 columns and 220 no
nzeros
Model fingerprint: 0x14c26846
Variable types: 7 continuous, 42 integer (42 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [1e+01, 5e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+01]
Found heuristic solution: objective 154.0000000
Presolve removed 48 rows and 30 columns
Presolve time: 0.00s
Presolved: 19 rows, 19 columns, 82 nonzeros
Variable types: 5 continuous, 14 integer (14 binary)

Root relaxation: cutoff, 7 iterations, 0.00 seconds
```

In [13]:

```
[(0, 1), (1, 2), (2, 6)]
[(0, 3), (3, 6)]
[(0, 5), (5, 4), (4, 6)]
```

In [14]:

```
'c' argument looks like a single numeric RGB or RGBA
sequence, which should be avoided as value-mapping w
ill have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA v
alue for all points.
/Users/shreyas/opt/anaconda3/lib/python3.7/site-pack
ages/ipykernel_launcher.py:24: MatplotlibDeprecation
Warning: Adding an axes using the same arguments as
a previous axes currently reuses the earlier instanc
e.  In a future version, a new instance will always
be created and returned.  Meanwhile, this warning ca
n be suppressed, and the future behavior ensured, by
passing a unique label to each axes instance.
'c' argument looks like a single numeric RGB or RGBA
sequence, which should be avoided as value-mapping w
ill have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA v
```

TESTING WITH PRE_DEFINED DATA N=15

P-n16-k8.vrp - Augerat et al. Set - P

In [15]:

Out[15]:

```
array([[ 0., 13., 21., 32., 22., 23., 12., 22., 32.,
32., 20., 28., 29.,
         29., 30., 29.,  0.],
        [13.,  0., 12., 19., 11., 24., 12., 18., 20.,
26.,  7., 18., 16.,
         21., 32., 17., 13.],
        [21., 12.,  0., 15., 22., 16., 10.,  8., 12.,
14., 10., 29., 18.,
          9., 23., 23., 21.],
        [32., 19., 15.,  0., 21., 31., 25., 23.,  7.,
24., 12., 25.,  9.,
         17., 37., 15., 32.],
        [22., 11., 22., 21.,  0., 35., 23., 29., 26.,
36., 12.,  7., 13.,
         30., 44.,  9., 22.],
        [23., 24., 16., 31., 35.,  0., 12.,  8., 25.,
13., 26., 43., 35.,
         16.,  8., 39., 23.],
```

In [17]:

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac6
4)
Optimize a model with 337 rows, 289 columns and 1540
nonzeros
Model fingerprint: 0x464e5be6
Variable types: 17 continuous, 272 integer (272 bina
ry)
Coefficient statistics:
  Matrix range     [1e+00, 7e+01]
  Objective range  [6e+00, 5e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 4e+01]
Presolve removed 186 rows and 130 columns
Presolve time: 0.00s
Presolved: 151 rows, 159 columns, 1316 nonzeros
Variable types: 13 continuous, 146 integer (146 bina
ry)

Root relaxation: objective 2.482569e+02, 103 iterati
```

In [18]:

```
[(0, 1), (1, 16)]
[(0, 2), (2, 16)]
[(0, 3), (3, 9), (9, 5), (5, 16)]
[(0, 6), (6, 16)]
[(0, 10), (10, 12), (12, 15), (15, 16)]
[(0, 11), (11, 4), (4, 16)]
[(0, 13), (13, 8), (8, 16)]
[(0, 14), (14, 7), (7, 16)]
[[0, 1, 16], [0, 2, 16], [0, 3, 5, 9, 16], [0, 6, 16],
[0, 10, 12, 15, 16], [0, 4, 11, 16], [0, 8, 13, 16], [
0, 7, 14, 16]]
```

In [19]:

```
/Users/shreyas/opt/anaconda3/lib/python3.7/site-pack
ages/ipykernel_launcher.py:24: MatplotlibDeprecation
Warning: Adding an axes using the same arguments as
a previous axes currently reuses the earlier instanc
e.  In a future version, a new instance will always
be created and returned.  Meanwhile, this warning ca
n be suppressed, and the future behavior ensured, by
passing a unique label to each axes instance.
'c' argument looks like a single numeric RGB or RGBA
sequence, which should be avoided as value-mapping w
ill have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA v
alue for all points.
'c' argument looks like a single numeric RGB or RGBA
sequence, which should be avoided as value-mapping w
ill have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA v
```

TESTING WITH PRE_DEFINED DATA N=20

P-n20-k2.vrp - Augerat et al. Set - P

In [23]:

Out[23]:

```
array([[ 0., 13., 21., 32., 22., 23., 12., 22., 32.,
32., 20., 28., 29.,
         29., 30., 29., 31., 39., 43., 15.,  0.],
        [13.,  0., 12., 19., 11., 24., 12., 18., 20.,
26.,  7., 18., 16.,
         21., 32., 17., 30., 27., 31., 18., 13.],
        [21., 12.,  0., 15., 22., 16., 10.,  8., 12.,
14., 10., 29., 18.,
          9., 23., 23., 20., 19., 24., 14., 21.],
        [32., 19., 15.,  0., 21., 31., 25., 23.,  7.,
24., 12., 25.,  9.,
         17., 37., 15., 32., 10., 12., 29., 32.],
        [22., 11., 22., 21.,  0., 35., 23., 29., 26.,
36., 12.,  7., 13.,
         30., 44.,  9., 41., 31., 32., 30., 22.],
        [23., 24., 16., 31., 35.,  0., 12.,  8., 25.,
13., 26., 43., 35.,
         16.,  8., 39.,  9., 31., 37.,  7., 23.],
```

In [24]:

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac6
4)
Optimize a model with 501 rows, 441 columns and 2404
nonzeros
Model fingerprint: 0xa6550ad2
Variable types: 21 continuous, 420 integer (420 bina
ry)
Coefficient statistics:
  Matrix range     [1e+00, 2e+02]
  Objective range  [6e+00, 5e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+02]
Presolve removed 120 rows and 42 columns
Presolve time: 0.01s
Presolved: 381 rows, 399 columns, 1767 nonzeros
Variable types: 19 continuous, 380 integer (380 bina
ry)

Root relaxation: objective 1.342688e+02, 69 iteratio
ns, 0.00 seconds
```

In [25]:

```
[(0, 6), (6, 13), (13, 8), (8, 17), (17, 18), (18, 3),
(3, 12), (12, 15), (15, 11), (11, 4), (4, 20)]
[(0, 19), (19, 5), (5, 14), (14, 16), (16, 9), (9, 7),
(7, 2), (2, 10), (10, 1), (1, 20)]
[[0, 3, 4, 6, 8, 11, 12, 13, 15, 17, 18, 20], [0, 1, 2
, 5, 7, 9, 10, 14, 16, 19, 20]]
```

In [26]:

```
'c' argument looks like a single numeric RGB or RGBA
sequence, which should be avoided as value-mapping w
ill have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA v
alue for all points.
/Users/shreyas/opt/anaconda3/lib/python3.7/site-pack
ages/ipykernel_launcher.py:24: MatplotlibDeprecation
Warning: Adding an axes using the same arguments as
a previous axes currently reuses the earlier instanc
e.  In a future version, a new instance will always
be created and returned.  Meanwhile, this warning ca
n be suppressed, and the future behavior ensured, by
passing a unique label to each axes instance.
'c' argument looks like a single numeric RGB or RGBA
sequence, which should be avoided as value-mapping w
ill have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA v
```

TESTING WITH PRE_DEFINED DATA N=32

A-n32-k5.vrp - Augerat et al. Set - A

```
In [20]:
```

```
Out[20]:

array([[ 0., 34., 77., ..., 16., 72.,  0.],
       [34.,  0., 60., ..., 19., 39., 34.],
       [77., 60.,  0., ..., 65., 48., 77.],
       ...,
       [16., 19., 65., ...,  0., 56., 16.],
       [72., 39., 48., ..., 56.,  0., 72.],
       [ 0., 34., 77., ..., 16., 72.,  0.]])
```

In [21]:

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac6
4)
Optimize a model with 1185 rows, 1089 columns and 61
48 nonzeros
Model fingerprint: 0xf3e5f92b
Variable types: 33 continuous, 1056 integer (1056 bi
nary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [2e+00, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+02]
Presolve removed 192 rows and 66 columns
Presolve time: 0.01s
Presolved: 993 rows, 1023 columns, 4743 nonzeros
Variable types: 31 continuous, 992 integer (992 bina
ry)

Root relaxation: objective 3.528700e+02, 117 iterati
```

In [ ]:

In [ ]: