

CSCI 5654 - Linear Programming - Project

Team Members

1. Ketan Ramesh

2. Shreyas Gopalakrishna

Vehicle Routing Problem

In [1]:

```
!pip3 install pulp
```

Collecting pulp

Downloading <https://files.pythonhosted.org/packages/41/34/757c88c320f80ce602199603afe63aed1e0bc11180b9a9fb6018fb2ce7ef/PuLP-2.1-py3-none-any.whl>
(<https://files.pythonhosted.org/packages/41/34/757c88c320f80ce602199603afe63aed1e0bc11180b9a9fb6018fb2ce7ef/PuLP-2.1-py3-none-any.whl>) (40.6MB)

|██| 40.6MB 71kB/s

Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from pulp) (2.4.7)

Installing collected packages: pulp

Successfully installed pulp-2.1

In [7]:

```
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
```

Capacity Vehicle Routing Problem - Model 2

Paper Link - [Integer linear programming formulation for a vehicle routing problem](https://doi.org/10.1016/0377-2217(91)90338-V)
([https://doi.org/10.1016/0377-2217\(91\)90338-V](https://doi.org/10.1016/0377-2217(91)90338-V))

The code provided below creates a class to frame and solve the Capacity Vehicle Routing Problem (CVRP) using a Vehicle Path variant of the **Kulkarni-Bhave formulation** as mentioned in the paper. The problem as a Mixed ILP is as follows:

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\sum_{i=1, i \neq j}^N x_{ij} = 1, j = 1, \dots, N-1, -(2.1)$$

$$\sum_{i=1, i \neq j}^N x_{ij} = 1, j = 1, \dots, N-1, -(2.1)$$

$$\sum_{j=1, j \neq i}^{N-1} x_{ij} + x_{iN'} = 1, j = 1, \dots, N-1, -(2.2)$$

$$\sum_{j=1, j \neq i}^{N-1} x_{ij} + x_{iN'} = 1, j = 1, \dots, N-1, -(2.2)$$

$$\sum_{j=1}^{N-1} x_{Nj} = V, -(2.3)$$

$$\sum_{j=1}^{N-1} x_{Nj} = V, -(2.3)$$

$$\sum_{i=1}^{N-1} x_{iN'} = V, -(2.4)$$

$$\sum_{i=1}^{N-1} x_{iN'} = V, -(2.4)$$

$$y_i - y_j + (L + 1)x_{ij} \leq L, \forall (i, j) \in A, -(2.5)$$

$$y_i - y_j + (L + 1)x_{ij} \leq L, \forall (i, j) \in A, -(2.5)$$

$$y_{N'} - y_N \leq L, \forall (i, j) \in A, -(2.6)$$

$$y_{N'} - y_N \leq L, \forall (i, j) \in A, -(2.6)$$

$$u_i - u_j + Wx_{ij} \leq W - Q_j, \forall (i, j) \in A, -(2.7)$$

$$u_i - u_j + Wx_{ij} \leq W - Q_j, \forall (i, j) \in A, -(2.7)$$

$$u_{N'} - u_N = W, -(2.8)$$

$$u_{N'} - u_N = W, -(2.8)$$

$$v_i - v_j + (c_{ij} + T)x_{ij} \leq T, \forall (i, j) \in A, -(2.9)$$

$$v_i - v_j + (c_{ij} + T)x_{ij} \leq T, \forall (i, j) \in A, -(2.9)$$

$$v_{N'} - v_N = T, -(2.10)$$

$$v_{N'} - v_N = T, -(2.10)$$

$$x_{ij} = \{0, 1\}, \forall (i, j) \in A$$

$$x_{ij} = \{0, 1\}, \forall (i, j) \in A$$

where,

$N - 1$ customer locations are denoted as $1, 2, \dots, N - 1$. The location of depot is denoted by N .

Q_j = Demand of customer at location j .

V = Number of delivery vehicles.

L = Number of customers that a vehicle can service in a route (preset).

T = Maximum distance that any route can reach (preset).

W = Capacity of the vehicle.

c_{ij} = Cost incurred on traveling from i to j .

Depot = N, N'

$$A = \{(N, i), (i, j), (i, N') : 1 \leq i \neq j \leq N - 1\},$$

$$A = \{(N, i), (i, j), (i, N') : 1 \leq i \neq j \leq N - 1\},$$

Decision Variables:

1. $x_{ij} = \{1, \text{ if edge } (i, j) \text{ present in any route. } 0, \text{ otherwise}\}.$
2. y_{ij}, u_{ij} and $v_{ij} = \text{A set of continuous non-negative variables.}$

The objective is to minimize the sum of the distances of all routes that satisfy the given constraints. The constraint (2, 1) ensures that all customer nodes are visited by exactly once while the constraint (2, 2) ensures that the number of paths arising from a node is exactly one (prevents multiple visits to the same node). The constraints (2, 3). and (2, 4) add the requirement that all vehicles be used for routing. The constraints (2, 5) and (2, 6) are used to eliminate sub tours in the problem. The constraints (2, 7) and (2, 8) ensure that the cumulative load on the route doesn't exceed the vehicle capacity. The constraints (2, 9) and (2, 10) ensure that each vehicle route doesn't exceed a set distance.

In [1]:

```
class CVRP2_GUROBI:
    def __init__(self, numberOfCustomers, numberOfVehicles, capacityOfVehicle, demandOfCustomers, costMatrix, A, L, T, N):
        self.numberOfCustomers = numberOfCustomers
        self.numberOfVehicles = numberOfVehicles
        self.capacityOfVehicle = capacityOfVehicle
        self.demandOfCustomers = demandOfCustomers
        self.costMatrix = costMatrix
        self.A = A
        self.L = L
        self.T = T
        self.N = len(self.costMatrix)
```

```
self.initializeLP()
```

```
def initializeLP(self):
    self.cvrpLP = gp.Model("CVRP_2")
    objective = None
    x, y = [], []
    u, v = [], []

    # Decision Variables
    for i in range(self.N):
        xRow = []
        y.append(self.cvrpLP.addVar(name='y('+str(i)+'')
        u.append(self.cvrpLP.addVar(name='u('+str(i)+'')
        v.append(self.cvrpLP.addVar(name='v('+str(i)+'')
        for j in range(self.N):
            xRow.append(self.cvrpLP.addVar(name='x('+str
        x.append(xRow)

    # Adding objective
    for tup in self.A:
        objective += costMatrix[tup[0]][tup[1]] * x[tup
    self.cvrpLP.setObjective(objective, GRB.MINIMIZE)

    # Adding constraint 1
    for j in range(1, self.N - 1):
        constraint1 = None
        for i in range(1, self.N):
            if (i != j):
                if (constraint1 == None):
                    constraint1 = x[i][j]
                else:
                    constraint1 = constraint1 + x[i][j]
        self.cvrpLP.addConstr(constraint1 == 1)

    # Adding constraint 2
    for i in range(1, self.N - 1):
        constraint2 = None
        for j in range(1, self.N - 1):
            if (i != j):
                if (constraint2 == None):
                    constraint2 = x[i][j]
                else:
                    constraint2 = constraint2 + x[i][j]
        constraint2 = constraint2 + x[i][self.N - 1]
        self.cvrpLP.addConstr(constraint2 == 1)
```

```

# Adding constraint 3
constraint3 = None
for i in range(1, self.N - 1):
    if(constraint3 == None):
        constraint3 = x[0][i]
    else:
        constraint3 = constraint3 + x[0][i]
self.cvrpLP.addConstr(constraint3 == self.numberOfVehicles)

# Adding constraint 4
constraint4 = None
for i in range(1, self.N - 1):
    if(constraint4 == None):
        constraint4 = x[i][self.N - 1]
    else:
        constraint4 = constraint4 + x[i][self.N - 1]
self.cvrpLP.addConstr(constraint4 == self.numberOfVehicles)

# Adding constraint 5
for tup in self.A:
    self.cvrpLP.addConstr(y[tup[0]] - y[tup[1]] + (self.costMatrix[tup[0]][tup[1]] - self.costMatrix[tup[1]][tup[0]]) * x[tup[0]][tup[1]] == 0)

# Adding constraint 6
self.cvrpLP.addConstr(y[-1] - y[0] <= self.L + 1)

# Adding constraint 7
for i, j in self.A:
    self.cvrpLP.addConstr(u[i] - u[j] + self.capacity * x[i][j] == 0)

# Adding constraint 8
self.cvrpLP.addConstr(u[-1] - u[0] == self.capacity)

# Adding constraint 9
for i, j in self.A:
    self.cvrpLP.addConstr(v[i] - v[j] + (self.costMatrix[i][j] - self.costMatrix[j][i]) * x[i][j] == 0)

# Adding constraint 10
self.cvrpLP.addConstr(v[-1] - v[0] == self.T)

# Adding flow constraint
for h in range(1, len(costMatrix)-1):
    constraint5a, constraint5b = None, None
    for i in range(0, len(costMatrix)-1):
        if(i != h):
            constraint5a = constraint5a + x[i][h]
            constraint5b = constraint5b + x[h][i]
    self.cvrpLP.addConstr(constraint5a == constraint5b)

```

```

        if(constraint5a == None):
            constraint5a = x[i][h]
        else:
            constraint5a = constraint5a
    for j in range(1, len(costMatrix)):
        if(j != h):
            if(constraint5b == None):
                constraint5b = x[h][j]
            else:
                constraint5b = constraint5b
    self.cvrpLP.addConstr(constraint5a - constraint5b)

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

```

Random Testing

In [4]:

```
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, 6: 0, 0: 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
costMatrix
```

Out [4]:

```
array([[ 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.]])
```


In [5]:

```
A = []
L = numberOfCustomers
T = 999999
for i in range(1, len(costMatrix) - 1):
    A.append((0, i))
    for j in range(1, len(costMatrix) - 1):
        if (i != j):
            A.append((i, j))
    A.append((i, len(costMatrix) - 1))
print(len(A))
```

30

In [8]:

```
cvrp = CVRP2_GUROBI(numberOfCustomers, numberOfVehicles, capacityOfVehicles)
```

Using license file /Users/shreyas/gurobi.lic
Academic license – for non-commercial use only

In [9]:

```
cvrp.solve()  
result = cvrp.getResult()
```

Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)

Optimize a model with 110 rows, 70 columns and 386 nonzeros

Model fingerprint: 0x6b9ab9d9

Variable types: 21 continuous, 49 integer (49 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+06]

Objective range [1e+01, 5e+01]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 1e+06]

Found heuristic solution: objective 162.0000000

Presolve removed 8 rows and 22 columns

Presolve time: 0.00s

Presolved: 102 rows, 48 columns, 330 nonzeros

Variable types: 18 continuous, 30 integer (30 binary)

Best objective: objective 1.500000e+02 25 iterations

In [16]:

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

pathList = []
setList = []
start = 0
for v in variables:
    if(v[0] == start):
        path = recursiveList(v[1], variables, [v], [v[0]])
        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), (0, 3), (0, 5), (1, 2), (2, 6), (3, 6), (4, 6),
(5, 4), (6, 1), (6, 3), (6, 5)]
```

In [11]:

```
xCoordinates = [30, 20, 10, 10, 40, 50, 35]
yCoordinates = [30, 40, 45, 30, 10, 30, 35]

coordinateList = []
for s in variables:
    coordinate = []
    for j in s:
        coordinate.append([float(xCoordinates[j]), float(yCoordinates[j])])
    coordinateList.append(coordinate)
# print(coordinateList[0])
```

In [15]:

```
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', linewidth=1)
    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, color=newColor)

    # 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, color=newColor)

for i in range(len(coordinateList)):
    addToPlot(coordinateList[i])
plt.scatter(coordinateList[0][0][0], coordinateList[0][0][1], c='black')
plt.show()
```

In [0]:

TESTING WITH PRE_DEFINED DATA N=15

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

In [17]:

```
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 = {0:0,1:19,2:30,3:16,4:23,5:11,6:31,7:15,8:28,9:30,10:19,11:16,12:23,13:11,14:31,15:15,16:28,17:30,18:19,19:16,20:23,21:11,22:31,23:15,24:28,25:30,26:19,27:16,28:23,29:11,30:31,31:15,32:28,33:30,34:19,35:16,36:23,37:11,38:31,39:15,40:28,41:30,42:19,43:16,44:23,45:11,46:31,47:15,48:28,49:30,50:19,51:16,52:23,53:11,54:31,55:15,56:28,57:30,58:19,59:16,60:23,61:11,62:31,63:15,64:28,65:30,66:19,67:16,68:23,69:11,70:31,71:15,72:28,73:30,74:19,75:16,76:23,77:11,78:31,79:15,80:28,81:30,82:19,83:16,84:23,85:11,86:31,87:15,88:28,89:30,90:19,91:16,92:23,93:11,94:31,95:15,96:28,97:30,98:19,99:16,100:23,101:11,102:31,103:15,104:28,105:30,106:19,107:16,108:23,109:11,110:31,111:15,112:28,113:30,114:19,115:16,116:23,117:11,118:31,119:15,120:28,121:30,122:19,123:16,124:23,125:11,126:31,127:15,128:28,129:30,130:19,131:16,132:23,133:11,134:31,135:15,136:28,137:30,138:19,139:16,140:23,141:11,142:31,143:15,144:28,145:30,146:19,147:16,148:23,149:11,150:31,151:15,152:28,153:30,154:19,155:16,156:23,157:11,158:31,159:15,160:28,161:30,162:19,163:16,164:23,165:11,166:31,167:15,168:28,169:30,170:19,171:16,172:23,173:11,174:31,175:15,176:28,177:30,178:19,179:16,180:23,181:11,182:31,183:15,184:28,185:30,186:19,187:16,188:23,189:11,190:31,191:15,192:28,193:30,194:19,195:16,196:23,197:11,198:31,199:15,200:28,201:30,202:19,203:16,204:23,205:11,206:31,207:15,208:28,209:30,210:19,211:16,212:23,213:11,214:31,215:15,216:28,217:30,218:19,219:16,220:23,221:11,222:31,223:15,224:28,225:30,226:19,227:16,228:23,229:11,230:31,231:15,232:28,233:30,234:19,235:16,236:23,237:11,238:31,239:15,240:28,241:30,242:19,243:16,244:23,245:11,246:31,247:15,248:28,249:30,250:19,251:16,252:23,253:11,254:31,255:15,256:28,257:30,258:19,259:16,260:23,261:11,262:31,263:15,264:28,265:30,266:19,267:16,268:23,269:11,270:31,271:15,272:28,273:30,274:19,275:16,276:23,277:11,278:31,279:15,280:28,281:30,282:19,283:16,284:23,285:11,286:31,287:15,288:28,289:30,290:19,291:16,292:23,293:11,294:31,295:15,296:28,297:30,298:19,299:16,300:23,301:11,302:31,303:15,304:28,305:30,306:19,307:16,308:23,309:11,310:31,311:15,312:28,313:30,314:19,315:16,316:23,317:11,318:31,319:15,320:28,321:30,322:19,323:16,324:23,325:11,326:31,327:15,328:28,329:30,330:19,331:16,332:23,333:11,334:31,335:15,336:28,337:30,338:19,339:16,340:23,341:11,342:31,343:15,344:28,345:30,346:19,347:16,348:23,349:11,350:31,351:15,352:28,353:30,354:19,355:16,356:23,357:11,358:31,359:15,360:28,361:30,362:19,363:16,364:23,365:11,366:31,367:15,368:28,369:30,370:19,371:16,372:23,373:11,374:31,375:15,376:28,377:30,378:19,379:16,380:23,381:11,382:31,383:15,384:28,385:30,386:19,387:16,388:23,389:11,390:31,391:15,392:28,393:30,394:19,395:16,396:23,397:11,398:31,399:15,400:28,401:30,402:19,403:16,404:23,405:11,406:31,407:15,408:28,409:30,410:19,411:16,412:23,413:11,414:31,415:15,416:28,417:30,418:19,419:16,420:23,421:11,422:31,423:15,424:28,425:30,426:19,427:16,428:23,429:11,430:31,431:15,432:28,433:30,434:19,435:16,436:23,437:11,438:31,439:15,440:28,441:30,442:19,443:16,444:23,445:11,446:31,447:15,448:28,449:30,450:19,451:16,452:23,453:11,454:31,455:15,456:28,457:30,458:19,459:16,460:23,461:11,462:31,463:15,464:28,465:30,466:19,467:16,468:23,469:11,470:31,471:15,472:28,473:30,474:19,475:16,476:23,477:11,478:31,479:15,480:28,481:30,482:19,483:16,484:23,485:11,486:31,487:15,488:28,489:30,490:19,491:16,492:23,493:11,494:31,495:15,496:28,497:30,498:19,499:16,500:23,501:11,502:31,503:15,504:28,505:30,506:19,507:16,508:23,509:11,510:31,511:15,512:28,513:30,514:19,515:16,516:23,517:11,518:31,519:15,520:28,521:30,522:19,523:16,524:23,525:11,526:31,527:15,528:28,529:30,530:19,531:16,532:23,533:11,534:31,535:15,536:28,537:30,538:19,539:16,540:23,541:11,542:31,543:15,544:28,545:30,546:19,547:16,548:23,549:11,550:31,551:15,552:28,553:30,554:19,555:16,556:23,557:11,558:31,559:15,560:28,561:30,562:19,563:16,564:23,565:11,566:31,567:15,568:28,569:30,570:19,571:16,572:23,573:11,574:31,575:15,576:28,577:30,578:19,579:16,580:23,581:11,582:31,583:15,584:28,585:30,586:19,587:16,588:23,589:11,590:31,591:15,592:28,593:30,594:19,595:16,596:23,597:11,598:31,599:15,600:28,601:30,602:19,603:16,604:23,605:11,606:31,607:15,608:28,609:30,610:19,611:16,612:23,613:11,614:31,615:15,616:28,617:30,618:19,619:16,620:23,621:11,622:31,623:15,624:28,625:30,626:19,627:16,628:23,629:11,630:31,631:15,632:28,633:30,634:19,635:16,636:23,637:11,638:31,639:15,640:28,641:30,642:19,643:16,644:23,645:11,646:31,647:15,648:28,649:30,650:19,651:16,652:23,653:11,654:31,655:15,656:28,657:30,658:19,659:16,660:23,661:11,662:31,663:15,664:28,665:30,666:19,667:16,668:23,669:11,670:31,671:15,672:28,673:30,674:19,675:16,676:23,677:11,678:31,679:15,680:28,681:30,682:19,683:16,684:23,685:11,686:31,687:15,688:28,689:30,690:19,691:16,692:23,693:11,694:31,695:15,696:28,697:30,698:19,699:16,700:23,701:11,702:31,703:15,704:28,705:30,706:19,707:16,708:23,709:11,710:31,711:15,712:28,713:30,714:19,715:16,716:23,717:11,718:31,719:15,720:28,721:30,722:19,723:16,724:23,725:11,726:31,727:15,728:28,729:30,730:19,731:16,732:23,733:11,734:31,735:15,736:28,737:30,738:19,739:16,740:23,741:11,742:31,743:15,744:28,745:30,746:19,747:16,748:23,749:11,750:31,751:15,752:28,753:30,754:19,755:16,756:23,757:11,758:31,759:15,760:28,761:30,762:19,763:16,764:23,765:11,766:31,767:15,768:28,769:30,770:19,771:16,772:23,773:11,774:31,775:15,776:28,777:30,778:19,779:16,780:23,781:11,782:31,783:15,784:28,785:30,786:19,787:16,788:23,789:11,790:31,791:15,792:28,793:30,794:19,795:16,796:23,797:11,798:31,799:15,800:28,801:30,802:19,803:16,804:23,805:11,806:31,807:15,808:28,809:30,810:19,811:16,812:23,813:11,814:31,815:15,816:28,817:30,818:19,819:16,820:23,821:11,822:31,823:15,824:28,825:30,826:19,827:16,828:23,829:11,830:31,831:15,832:28,833:30,834:19,835:16,836:23,837:11,838:31,839:15,840:28,841:30,842:19,843:16,844:23,845:11,846:31,847:15,848:28,849:30,850:19,851:16,852:23,853:11,854:31,855:15,856:28,857:30,858:19,859:16,860:23,861:11,862:31,863:15,864:28,865:30,866:19,867:16,868:23,869:11,870:31,871:15,872:28,873:30,874:19,875:16,876:23,877:11,878:31,879:15,880:28,881:30,882:19,883:16,884:23,885:11,886:31,887:15,888:28,889:30,890:19,891:16,892:23,893:11,894:31,895:15,896:28,897:30,898:19,899:16,900:23,901:11,902:31,903:15,904:28,905:30,906:19,907:16,908:23,909:11,910:31,911:15,912:28,913:30,914:19,915:16,916:23,917:11,918:31,919:15,920:28,921:30,922:19,923:16,924:23,925:11,926:31,927:15,928:28,929:30,930:19,931:16,932:23,933:11,934:31,935:15,936:28,937:30,938:19,939:16,940:23,941:11,942:31,943:15,944:28,945:30,946:19,947:16,948:23,949:11,950:31,951:15,952:28,953:30,954:19,955:16,956:23,957:11,958:31,959:15,960:28,961:30,962:19,963:16,964:23,965:11,966:31,967:15,968:28,969:30,970:19,971:16,972:23,973:11,974:31,975:15,976:28,977:30,978:19,979:16,980:23,981:11,982:31,983:15,984:28,985:30,986:19,987:16,988:23,989:11,990:31,991:15,992:28,993:30,994:19,995:16,996:23,997:11,998:31,999:15,1000:28,1001:30,1002:19,1003:16,1004:23,1005:11,1006:31,1007:15,1008:28,1009:30,1010:19,1011:16,1012:23,1013:11,1014:31,1015:15,1016:28,1017:30,1018:19,1019:16,1020:23,1021:11,1022:31,1023:15,1024:28,1025:30,1026:19,1027:16,1028:23,1029:11,1030:31,1031:15,1032:28,1033:30,1034:19,1035:16,1036:23,1037:11,1038:31,1039:15,1040:28,1041:30,1042:19,1043:16,1044:23,1045:11,1046:31,1047:15,1048:28,1049:30,1050:19,1051:16,1052:23,1053:11,1054:31,1055:15,1056:28,1057:30,1058:19,1059:16,1060:23,1061:11,1062:31,1063:15,1064:28,1065:30,1066:19,1067:16,1068:23,1069:11,1070:31,1071:15,1072:28,1073:30,1074:19,1075:16,1076:23,1077:11,1078:31,1079:15,1080:28,1081:30,1082:19,1083:16,1084:23,1085:11,1086:31,1087:15,1088:28,1089:30,1090:19,1091:16,1092:23,1093:11,1094:31,1095:15,1096:28,1097:30,1098:19,1099:16,1100:23,1101:11,1102:31,1103:15,1104:28,1105:30,1106:19,1107:16,1108:23,1109:11,1110:31,1111:15,1112:28,1113:30,1114:19,1115:16,1116:23,1117:11,1118:31,1119:15,1120:28,1121:30,1122:19,1123:16,1124:23,1125:11,1126:31,1127:15,1128:28,1129:30,1130:19,1131:16,1132:23,1133:11,1134:31,1135:15,1136:28,1137:30,1138:19,1139:16,1140:23,1141:11,1142:31,1143:15,1144:28,1145:30,1146:19,1147:16,1148:23,1149:11,1150:31,1151:15,1152:28,1153:30,1154:19,1155:16,1156:23,1157:11,1158:31,1159:15,1160:28,1161:30,1162:19,1163:16,1164:23,1165:11,1166:31,1167:15,1168:28,1169:30,1170:19,1171:16,1172:23,1173:11,1174:31,1175:15,1176:28,1177:30,1178:19,1179:16,1180:23,1181:11,1182:31,1183:15,1184:28,1185:30,1186:19,1187:16,1188:23,1189:11,1190:31,1191:15,1192:28,1193:30,1194:19,1195:16,1196:23,1197:11,1198:31,1199:15,1200:28,1201:30,1202:19,1203:16,1204:23,1205:11,1206:31,1207:15,1208:28,1209:30,1210:19,1211:16,1212:23,1213:11,1214:31,1215:15,1216:28,1217:30,1218:19,1219:16,1220:23,1221:11,1222:31,1223:15,1224:28,1225:30,1226:19,1227:16,1228:23,1229:11,1230:31,1231:15,1232:28,1233:30,1234:19,1235:16,1236:23,1237:11,1238:31,1239:15,1240:28,1241:30,1242:19,1243:16,1244:23,1245:11,1246:31,1247:15,1248:28,1249:30,1250:19,1251:16,1252:23,1253:11,1254:31,1255:15,1256:28,1257:30,1258:19,1259:16,1260:23,1261:11,1262:31,1263:15,1264:28,1265:30,1266:19,1267:16,1268:23,1269:11,1270:31,1271:15,1272:28,1273:30,1274:19,1275:16,1276:23,1277:11,1278:31,1279:15,1280:28,1281:30,1282:19,1283:16,1284:23,1285:11,1286:31,1287:15,1288:28,1289:30,1290:19,1291:16,1292:23,1293:11,1294:31,1295:15,1296:28,1297:30,1298:19,1299:16,1300:23,1301:11,1302:31,1303:15,1304:28,1305:30,1306:19,1307:16,1308:23,1309:11,1310:31,1311:15,1312:28,1313:30,1314:19,1315:16,1316:23,1317:11,1318:31,1319:15,1320:28,1321:30,1322:19,1323:16,1324:23,1325:11,1326:31,1327:15,1328:28,1329:30,1330:19,1331:16,1332:23,1333:11,1334:31,1335:15,1336:28,1337:30,1338:19,1339:16,1340:23,1341:11,1342:31,1343:15,1344:28,1345:30,1346:19,1347:16,1348:23,1349:11,1350:31,1351:15,1352:28,1353:30,1354:19,1355:16,1356:23,1357:11,1358:31,1359:15,1360:28,1361:30,1362:19,1363:16,1364:23,1365:11,1366:31,1367:15,1368:28,1369:30,1370:19,1371:16,1372:23,1373:11,1374:31,1375:15,1376:28,1377:30,1378:19,1379:16,1380:23,1381:11,1382:31,1383:15,1384:28,1385:30,1386:19,1387:16,1388:23,1389:11,1390:31,1391:15,1392:28,1393:30,1394:19,1395:16,1396:23,1397:11,1398:31,1399:15,1400:28,1401:30,1402:19,1403:16,1404:23,1405:11,1406:31,1407:15,1408:28,1409:30,1410:19,1411:16,1412:23,1413:11,1414:31,1415:15,1416:28,1417:30,1418:19,1419:16,1420:23,1421:11,1422:31,1423:15,1424:28,1425:30,1426:19,1427:16,1428:23,1429:11,1430:31,1431:15,1432:28,1433:30,1434:19,1435:16,1436:23,1437:11,1438:31,1439:15,1440:28,1441:30,1442:19,1443:16,1444:23,1445:11,1446:31,1447:15,1448:28,1449:30,1450:19,1451:16,1452:23,1453:11,1454:31,1455:15,1456:28,1457:30,1458:19,1459:16,1460:23,1461:11,1462:31,1463:15,1464:28,1465:30,1466:19,1467:16,1468:23,1469:11,1470:31,1471:15,1472:28,1473:30,1474:19,1475:16,1476:23,1477:11,1478:31,1479:15,1480:28,1481:30,1482:19,1483:16,1484:23,1485:11,1486:31,1487:15,1488:28,1489:30,1490:19,1491:16,1492:23,1493:11,1494:31,1495:15,1496:28,1497:30,1498:19,1499:16,1500:23,1501:11,1502:31,1503:15,1504:28,1505:30,1506:19,1507:16,1508:23,1509:11,1510:31,1511:15,1512:28,1513:30,1514:19,1515:16,1516:23,1517:11,1518:31,1519:15,1520:28,1521:30,1522:19,1523:16,1524:23,1525:11,1526:31,1527:15,1528:28,1529:30,1530:19,1531:16,1532:23,1533:11,1534:31,1535:15,1536:28,1537:30,1538:19,1539:16,1540:23,1541:11,1542:31,1543:15,1544:28,1545:30,1546:19,1547:16,1548:23,1549:11,1550:31,1551:15,1552:28,1553:30,1554:19,1555:16,1556:23,1557:11,1558:31,1559:15,1560:28,1561:30,1562:19,1563:16,1564:23,1565:11,1566:31,1567:15,1568:28,1569:30,1570:19,1571:16,1572:23,1573:11,1574:31,1575:15,1576:28,1577:30,1578:19,1579:16,1580:23,1581:11,1582:31,1583:15,1584:28,1585:30,1586:19,1587:16,1588:23,1589:11,1590:31,1591:15,1592:28,1593:30,1594:19,1595:16,1596:23,1597:11,1598:31,1599:15,1600:28,1601:30,1602:19,1603:16,1604:23,1605:11,1606:31,1607:15,1608:28,1609:30,1610:19,1611:16,1612:23,1613:11,1614:31,1615:15,1616:28,1617:30,1618:19,1619:16,1620:23,1621:11,1622:31,1623:15,1624:28,1625:30,1626:19,1627:16,1628:23,1629:11,1630:31,1631:15,1632:28,1633:30,1634:19,1635:16,1636:23,1637:11,1638:31,1639:15,1640:28,1641:30,1642:19,1643:16,1644:23,1645:11,1646:31,1647:15,1648:28,1649:30,1650:19,1651:16,1652:23,1653:11,1654:31,1655:15,1656:28,1
```


In [19]:

```
A = []
L = numberOfCustomers
T = 999999
for i in range(1, len(costMatrix) - 1):
    A.append((0, i))
    for j in range(1, len(costMatrix) - 1):
        if (i != j):
            A.append((i, j))
    A.append((i, len(costMatrix) - 1))
print(len(A))
```

240

In [20]:

```
cvrp2 = CVRP2_GUROBI(numberOfCustomers, numberOfVehicles, capacity0
```



In [21]:

```
cvrp2.solve()  
result = cvrp2.getResult()
```

Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)

Optimize a model with 770 rows, 340 columns and 3096 nonzeros

Model fingerprint: 0x4f6e72e5

Variable types: 51 continuous, 289 integer (289 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+06]

Objective range [6e+00, 5e+01]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 1e+06]

Presolve removed 18 rows and 52 columns

Presolve time: 0.01s

Presolved: 752 rows, 288 columns, 2640 nonzeros

Variable types: 48 continuous, 240 integer (240 binary)

Found heuristic solution: objective 560.0000000

Best solution: objective 2.004286e+02 100 iterations

TESTING WITH PRE_DEFINED DATA N=20

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

In [27]:

```
numberOfCustomers = 19  
capacityOfVehicle = 160  
numberOfVehicles = 2  
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

A = []
L = numberOfCustomers
T = 999999
for i in range(1, len(costMatrix) - 1):
    A.append((0, i))
    for j in range(1, len(costMatrix) - 1):
        if (i != j):
            A.append((i, j))
    A.append((i, len(costMatrix) - 1))
print(len(A))

```

In [28]:

```
cvrp2 = CVRP2_GUROBI(numberOfCustomers, numberOfVehicles, capacity0.  
cvrp2.solve()  
result = cvrp2.getResult()
```

Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)

Optimize a model with 1202 rows, 504 columns and 4908 nonzeros

Model fingerprint: 0x7f4d03a4

Variable types: 63 continuous, 441 integer (441 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+06]

Objective range [6e+00, 5e+01]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 1e+06]

Presolve removed 22 rows and 64 columns

Presolve time: 0.04s

Presolved: 1180 rows, 440 columns, 4180 nonzeros

Variable types: 60 continuous, 380 integer (380 binary)

Root relaxation: objective 1.736125e+02, 134 iterations, 0.01 seconds

TESTING WITH PRE_DEFINED DATA N=32

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

```

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 = [82,96,50,49,13,29,58,84,14,2,3,5,98,84,61,1,88,91,19]
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] = 999999
```


In [24]:

```
A = []
L = numberOfCustomers
T = 999999
for i in range(1, len(costMatrix) - 1):
    A.append((0, i))
    for j in range(1, len(costMatrix) - 1):
        if (i != j):
            A.append((i, j))
    A.append((i, len(costMatrix) - 1))
print(len(A))
```

992

In [25]:

```
cvrp2 = CVRP2_GUROBI(numberOfCustomers, numberOfVehicles, capacity0
```



In [26]:

```
cvrp2.solve()  
result = cvrp2.getResult()
```

Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)

Optimize a model with 3074 rows, 1188 columns and 12840 nonzeros

Model fingerprint: 0x7332ad83

Variable types: 99 continuous, 1089 integer (1089 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+06]

Objective range [2e+00, 1e+02]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 1e+06]

Presolve removed 34 rows and 100 columns

Presolve time: 0.04s

Presolved: 3040 rows, 1088 columns, 10912 nonzeros

Variable types: 96 continuous, 992 integer (992 binary)

Root relaxation: objective 5.349900e+02, 190 iterations, 0.01 seconds

In []:

CSCI 5654 - Linear Programming - Project

Team Members

1. Ketan Ramesh
2. Shreyas Gopalakrishna

Vehicle Routing Problem

In [1]:

Collecting pulp

Downloading <https://files.pythonhosted.org/packages/41/34/757c88c320f80ce602199603afe63aed1e0bc11180b9a9fb6018fb2ce7ef/PuLP-2.1-py3-none-any.whl>
(<https://files.pythonhosted.org/packages/41/34/757c88c320f80ce602199603afe63aed1e0bc11180b9a9fb6018fb2ce7ef/PuLP-2.1-py3-none-any.whl>) (40.6MB)

██████████ 40.6MB 71kB/s

```
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from pulp) (2.4.7)
```

Installing collected packages: pulp

Successfully installed pulp-2.1

In [7]:

Capacity Vehicle Routing Problem - Model 2

Paper Link - [Integer linear programming formulation for a vehicle routing problem](https://doi.org/10.1016/0377-2217(91)90338-V)
([https://doi.org/10.1016/0377-2217\(91\)90338-V](https://doi.org/10.1016/0377-2217(91)90338-V))

The code provided below creates a class to frame and solve the Capacity Vehicle Routing Problem (CVRP) using a Vehicle Path variant of the **Kulkarni-Bhave formulation** as mentioned in the paper. The problem as a Mixed ILP is as follows:

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\sum_{i=1, i \neq j}^N x_{ij} = 1, j = 1, \dots, N-1, - (2.1)$$

$$\sum_{j=1, j \neq i}^{N-1} x_{ij} + x_{iN'} = 1, j = 1, \dots, N-1, - (2.2)$$

$$\sum_{j=1}^{N-1} x_{Nj} = V, - (2.3)$$

$$\sum_{i=1}^{N-1} x_{iN'} = V, - (2.4)$$

$$y_i - y_j + (L+1)x_{ij} \leq L, \forall (i,j) \in A, - (2.5)$$

$$y_{N'} - y_N \leq L, \forall (i,j) \in A, - (2.6)$$

$$u_i - u_j + Wx_{ij} \leq W - Q_j, \forall (i,j) \in A, - (2.7)$$

$$u_{N'} - u_N = W, - (2.8)$$

$$v_i - v_j + (c_{ij} + T)x_{ij} \leq T, \forall (i,j) \in A, - (2.9)$$

$$v_{N'} - v_N = T, - (2.10)$$

$$x_{ij} = \{0, 1\}, \forall (i,j) \in A$$

where,

$N - 1$ customer locations are denoted as $1, 2, \dots, N - 1$. The location of depot is denoted by N .

Q_j = Demand of customer at location j .

V = Number of delivery vehicles.

L = Number of customers that a vehicle can service in a route (preset).

T = Maximum distance that any route can reach (preset).

W = Capacity of the vehicle.

c_{ij} = Cost incurred on traveling from i to j .

Depot = N, N'

$$A = \{(N, i), (i, j), (i, N') : 1 \leq i \neq j \leq N - 1\},$$

Decision Variables:

1. $x_{ij} = \{1, \text{ if edge } (i, j) \text{ present in any route. } 0, \text{ otherwise}\}.$
2. y_{ij}, u_{ij} and $v_{ij} =$ A set of continuous non-negative variables.

The objective is to minimize the sum of the distances of all routes that satisfy the given constraints. The constraint (2, 1) ensures that all customer nodes are visited by exactly once while the constraint (2, 2) ensures that the number of paths arising from a node is exactly one (prevents multiple visits to the same node). The constraints (2, 3). and (2, 4) add the requirement that all vehicles be used for routing. The constraints (2, 5) and (2, 6) are used to eliminate sub tours in the problem. The constraints (2, 7) and (2, 8) ensure that the cumulative load on the route doesn't exceed the vehicle capacity. The constraints (2, 9) and (2, 10) ensure that each vehicle route doesn't exceed a set distance.

In [1]:

Random Testing

In [4]:

Out [4]:

```
array([[ 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.]])
```

In [5]:

30

In [8]:

Using license file /Users/shreyas/gurobi.lic
Academic license – for non-commercial use only

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 110 rows, 70 columns and 386 nonzeros
Model fingerprint: 0x6b9ab9d9
Variable types: 21 continuous, 49 integer (49 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+06]
  Objective range   [1e+01, 5e+01]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 1e+06]
Found heuristic solution: objective 162.0000000
Presolve removed 8 rows and 22 columns
Presolve time: 0.00s
Presolved: 102 rows, 48 columns, 330 nonzeros
Variable types: 18 continuous, 30 integer (30 binary)
Best relaxation: objective 1.500000e+02, 25 iterations
```

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

Age Group	Percentage
18-24	28%
25-34	22%
35-44	18%
45-54	15%
55-64	12%
65-74	8%
75-84	5%
85+	2%

TESTING WITH PRE_DEFINED DATA N=15

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

In [17]:

In [18]:

In [19]:

240

In [20]:

In [21]:

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 770 rows, 340 columns and 3096
nonzeros
Model fingerprint: 0x4f6e72e5
Variable types: 51 continuous, 289 integer (289 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+06]
  Objective range   [6e+00, 5e+01]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 1e+06]
Presolve removed 18 rows and 52 columns
Presolve time: 0.01s
Presolved: 752 rows, 288 columns, 2640 nonzeros
Variable types: 48 continuous, 240 integer (240 binary)
Found heuristic solution: objective 560.0000000

Best objective: 560.0000000, 100 iterations
```

TESTING WITH PRE_DEFINED DATA N=20

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

In [27]:

380

In [28]:

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 1202 rows, 504 columns and 4908 nonzeros
Model fingerprint: 0x7f4d03a4
Variable types: 63 continuous, 441 integer (441 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+06]
  Objective range   [6e+00, 5e+01]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 1e+06]
Presolve removed 22 rows and 64 columns
Presolve time: 0.04s
Presolved: 1180 rows, 440 columns, 4180 nonzeros
Variable types: 60 continuous, 380 integer (380 binary)

Root relaxation: objective 1.736125e+02, 134 iterations, 0.01 seconds
```

TESTING WITH PRE_DEFINED DATA N=32

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

In [22]:

In [23]:

In [24]:

992

In [25]:

In [26]:

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 3074 rows, 1188 columns and 12840 nonzeros
Model fingerprint: 0x7332ad83
Variable types: 99 continuous, 1089 integer (1089 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+06]
  Objective range   [2e+00, 1e+02]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 1e+06]
Presolve removed 34 rows and 100 columns
Presolve time: 0.04s
Presolved: 3040 rows, 1088 columns, 10912 nonzeros
Variable types: 96 continuous, 992 integer (992 binary)

Root relaxation: objective 5.349900e+02, 190 iterations, 0.01 seconds
```

In []:

