DMOR Term Project (VRP & VRPTW)

This notebook was based on:

- X. Gan et al. Mathematical Problems in Engineering (2012).
- 2005_Book_ColumnGeneration, Desaulnier
- https://github.com/agustingu/Cplex-Python (https://github.com/agustingu/Cplex-Python)

Data

- Set of costumers $N = \{1, 2, .., n\}$, where $n = 50$ in this case.
- Set of vertices $V = \{0\} \cup N$, equal to the number of clients plus the depot located at $(0, 0)$.
- Set of arcs, $A = \{(i, j) \in V^2 : i \neq j\}$, i.e., all possible combinations of incoming and outgoing arcs excluding the case when $i = j$.
- Cost of traveling on each arc, $c_{i,j} \in A$. In this case the cost is relate to distance.
- Vehicle capacity, Q.
- Costumer demand, $q_i, \forall i \in N$.

Decision Variables

- Binary variable $x_{i,j}$ for when the arc is use (1); (0) otherwise.
- Cummulative demand of costumer i, MTZ notation $u_i \ \forall i \in N$
- Indicator variable for time

IVRs

- Decision variable is binary $x_{i,j} \in \{0, 1\}, \forall i, j \in A$.

Constraints

Each customer vertex is connected to two other vertices, which are its predecessor and successor:

- From each node we can only visit only one node, e.g., if I'm in node 2 I can only visit 1 node {0,1,3,..n}.
$$\sum_{i \in V, j \neq i} x_{i,j} = 1, \ \forall j \in N$$
- Each node could only be visisted from one node, e.g., if I'm in node 2 I can only be visited from node {0,1,3,..n}.
$$\sum_{j \in V, i \neq j} x_{i,j} = 1, \ \forall i \in N$$
- According to the literature if one arc is active, the cumulative demant until j point is the demand up to the previous node.
- Number of vehicles leaving the depot or (DC) is the same as entering
$$\sum_{i \ \in \ V} x_{i0} = K$$
$$\sum_{j \ \in \ V} x_{0j} = K$$

- Routes must be connected, demand on each rout should not exceed capacity

$$\sum_{i,j \in S} \geq r(s),\ \forall S \subseteq \{N\},\ S \neq \phi$$

Replacing the previous three constraint by MTZ notation proposed by Christofides, Mingozzi and Toth we have:

$$\text{if } x_{i,j} = 1 \ \Rightarrow \ u_i + q_j = u_j \ i, j \in A : j \neq 0, i \neq 0$$
$$q_i \leq u_i \leq Q,\ \forall i \in N$$

This last indicator constraint imposed the capacity and connectivity constraints

- If xij==1 the cummulative in uj = cum (ui) prev node + demand at node j qj

Algebraic Notation of VRP

$$\begin{aligned}
\min \quad & \sum_{i,j \in A} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i \in V, j \neq i} x_{i,j} = 1,\ \forall j \in N \\
& \sum_{j \in V, i \neq j} x_{i,j} = 1,\ \forall i \in N \\
& \text{if } x_{i,j} = 1 \ \Rightarrow \ u_j = u_i + q_j \quad \forall (i,j) \in A : j \neq 0, i \neq 0 \\
& q_i \leq u_i \leq Q,\ \forall i \in N \\
& x_{i,j} \in \{0,1\}, \forall i,j \in A
\end{aligned}$$

Data Creation

In [16]:

```
# Libraries
import numpy as np
import matplotlib.pyplot as plt
from docplex.mp.model import Model
import pandas as pd
import random
%matplotlib inline

random.seed(1)
```

In [21]:
```
# Load data
filename = '../Data.csv'
df = pd.read_csv(filename)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Node No.                 50 non-null     int64  
 1   X                        50 non-null     float64
 2   Y                        50 non-null     float64
 3   demand (pounds)          50 non-null     int64  
 4   service-time             50 non-null     object 
 5   time-window(start; h:m)  50 non-null     object 
 6   time-window(end; h:m)    50 non-null     object 
 7    Can this demand be split  50 non-null     object 
dtypes: float64(2), int64(2), object(4)
memory usage: 3.2+ KB
```

In [22]:
```
# Data revision
df.head()
```

Out[22]:

	Node No.	X	Y	demand (pounds)	service-time	time-window(start; h:m)	time-window(end; h:m)	Can this demand be split
0	1	-8.20	-96.68	2500	0:02	3:42	3:54	Y
1	2	-16.01	14.95	3250	0:03	22:58	23:53	N
2	3	-1.26	82.05	5500	0:05	18:35	18:56	N
3	4	-93.17	78.92	12500	0:10	21:29	21:52	Y
4	5	-57.10	18.67	1500	0:02	10:56	11:38	N

In [23]:

```
# Set of costumer n in N
N = df.loc[:, 'Node No.'].values.tolist()
# Set of Vertices
V = [0] + N
# demand per node or costumer
demand = df.loc[:, "demand (pounds)"].values.tolist()
# Trick capacity in pounds
Q = 20000
# Dictionary demand for each costumer
q = {i: (demand[i-1]) for i in N}
# Nodes coordinates
x = df.loc[:, "X"].values
y = df.loc[:, "Y"].values
# Insert distribution center coordinates at 0,0
loc_x = np.insert(x, 0, 0)
loc_y = np.insert(y, 0, 0)
# Travel cost
t_cost = 3
```

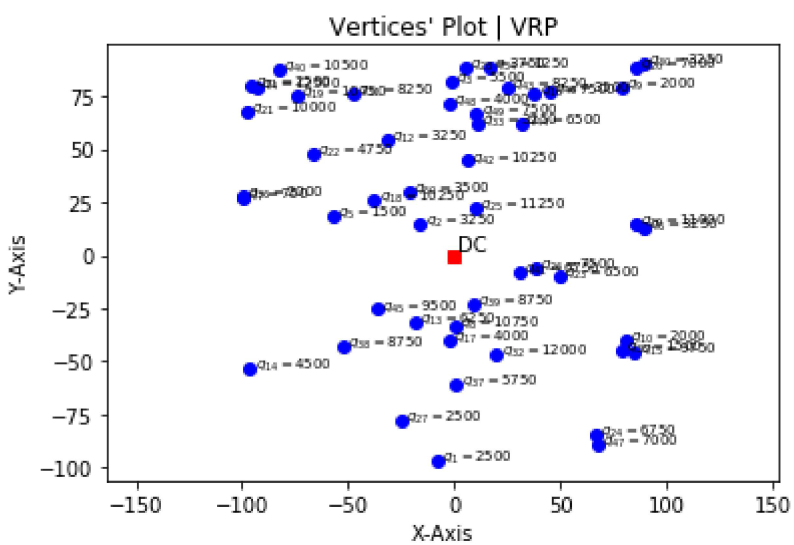
Plotting Vertices

In [24]:

```
# Plot vertices
plt.scatter(loc_x[1:], loc_y[1:], c='b')
# Add labels
for i in N:
    plt.annotate('$q_{%d}=%d$' % (i, q[i]), (loc_x[i]+2, loc_y[i]),
                 fontsize=7)
plt.annotate('DC',(loc_x[0]+2,loc_y[0]+2))
# Highlight depot
plt.plot(loc_x[0], loc_y[0], c='r', marker='s')
# Add axis
plt.axis('equal')
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
# Add title
plt.title("Vertices' Plot | VRP")
# Save plot
# plt.savefig("VRP_Plot", dpi=300, bbox_inches='tight')
```

Out[24]:

```
Text(0.5, 1.0, "Vertices' Plot | VRP")
```



Data Structure for Arcs and Distances

In [25]:

```
# Arcs
A={(i,j) for i in V for j in V if i!=j}
# Distances
c = {(i, j): np.hypot(loc_x[i]-loc_x[j], loc_y[i]-loc_y[j]) for i, j in A}
# Multiply the distance by the transportation cost $3 in this case
for k, v in c.items():
    c[k] = v*t_cost
```

Model

Variables

$$x_{i,j}$$
$$u_i, \ \forall i \in N$$

$$\begin{aligned}
\min \quad & \sum_{i,j \in A} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i \in V, j \neq i} x_{i,j} = 1, \ \forall j \in N \\
& \sum_{j \in V, i \neq j} x_{i,j} = 1, \ \forall i \in N \\
& \text{if } x_{i,j} = 1 \ \Rightarrow \ u_i + q_j = u_j \ \forall (i,j) \in A : j \neq 0, i \neq 0 \\
& q_i \leq u_i \leq Q, \ \forall i \in N \\
& x_{i,j} \in \{0,1\}, \forall i,j \in A
\end{aligned}$$

In [46]:
```python
# Creates model object
mdl = Model('VRP')
# Variables
x = mdl.binary_var_dict(A, name='x')
u = mdl.continuous_var_dict(N, ub=Q, name='u')
```

Objective

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

In [47]:
```python
# Objective
mdl.minimize(mdl.sum(c[i, j]*x[i, j] for i, j in A))
```

Constrainst

$$\begin{aligned}
\text{s.t.} \quad & \sum_{i \in V, j \neq i} x_{i,j} = 1, \ \forall j \in N \\
& \sum_{j \in V, i \neq j} x_{i,j} = 1, \ \forall i \in N \\
& \text{if } x_{i,j} = 1 \ \Rightarrow \ u_i + q_j = u_j \ i,j \in A : j \neq 0, i \neq 0 \\
& q_i \leq u_i \leq Q, \ \forall i \in N \\
& x_{i,j} \in \{0,1\}, \forall i,j \in A
\end{aligned}$$

In [28]:

```
# Constraints for enetering an leaving
mdl.add_constraints(mdl.sum(x[i, j] for j in V if j != i) == 1 for i in N)
mdl.add_constraints(mdl.sum(x[i, j] for i in V if i != j) == 1 for j in N)
# Indicator constraints for capacity and continuity
mdl.add_indicator_constraints(mdl.indicator_constraint(x[i, j], u[i]+q[j] ==
# Indicator constraint qi<=ui
mdl.add_constraints(u[i] >= q[i] for i in N)
print(mdl.export_to_string())
```

```
 0 <= x_34_40 <= 1
 0 <= x_17_32 <= 1
 0 <= x_37_45 <= 1
 0 <= x_23_14 <= 1
 0 <= x_45_29 <= 1
 0 <= x_10_1 <= 1
 0 <= x_35_27 <= 1
 0 <= x_0_23 <= 1
 0 <= x_38_4 <= 1
 0 <= x_25_17 <= 1
 0 <= x_14_45 <= 1
 0 <= x_1_8 <= 1
 0 <= x_39_31 <= 1
 0 <= x_4_3 <= 1
 0 <= x_26_48 <= 1
 0 <= x_29_5 <= 1
 0 <= x_16_6 <= 1
 0 <= x_2_41 <= 1
 0 <= x_41_2 <= 1
```

In [42]:

```
# Time limit and solving the model
mdl.parameters.timelimit = 120
solution = mdl.solve(log_output=True)
```

```
59.61%
  50764     0     3929.9872   127     9830.6882       Cuts: 76    661126
59.61%
  50764     0     3930.7236   127     9830.6882       Cuts: 47    661161
59.61%
  50764     2     3930.7236   137     9830.6882      3970.9080    661161
59.61%
  50974   176     6355.5949    29     9830.6882      3970.9080    664321
59.61%
  51433   590     6942.7731    21     9830.6882      3970.9080    670207
59.61%
  52090  1146     7040.2891    23     9830.6882      3970.9080    680603
59.61%
  52920  1806     4485.1795    69     9830.6882      3970.9080    694720
59.61%
  53867  2661     6439.3681    24     9830.6882      3970.9080    713819
59.61%
  54951  3562     4458.8425    93     9830.6882      3979.2784    732751
59.52%
```

In [43]:
```
solution.display()
```

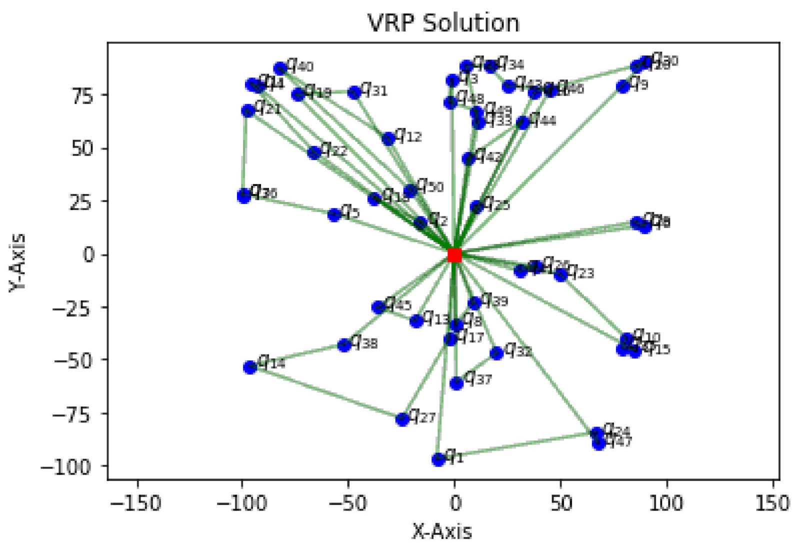
```
solution for: VRPTW
objective: 9776.813
x_9_0 = 1
x_16_20 = 1
x_23_0 = 1
x_35_10 = 1
x_48_3 = 1
x_21_7 = 1
x_30_9 = 1
x_31_19 = 1
x_24_1 = 1
x_6_0 = 1
x_0_49 = 1
x_0_8 = 1
x_8_39 = 1
x_0_15 = 1
x_19_0 = 1
x_0_29 = 1
x_38_14 = 1
```

In [44]:
```
solution.solve_status
```

Out[44]:
```
<JobSolveStatus.FEASIBLE_SOLUTION: 1>
```

In [45]:

```
# Actives arcs
active_arcs = [a for a in A if x[a].solution_value > 0.9]
plt.scatter(loc_x[1:], loc_y[1:], c='b')
# add labels to each node
for i in N:
    plt.annotate('$q_{%d}$' % (i,), (loc_x[i]+2, loc_y[i]))
# plot arcs
    for i, j in active_arcs:
        plt.plot([loc_x[i], loc_x[j]], [loc_y[i], loc_y[j]], c='g', alpha=0.0
plt.plot(loc_x[0], loc_y[0], c='r', marker='s')
plt.axis('equal')
# Add axis
plt.axis('equal')
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
# Add title
plt.title("VRP Solution")
# Save plot
plt.savefig("VRP_Solved", dpi=300, bbox_inches='tight')
```



In [33]:
```
active_arcs
```

Out[33]:
```
[(25, 3),
 (40, 11),
 (23, 0),
 (35, 10),
 (36, 0),
 (21, 7),
 (31, 19),
 (24, 1),
 (6, 0),
 (0, 49),
 (0, 8),
 (8, 39),
 (0, 15),
 (19, 0),
 (0, 29),
 (38, 14),
 (47, 24),
 (17, 0),
 (2, 0),
 (0, 45),
 (9, 20),
 (44, 0),
 (42, 44),
 (15, 35),
 (11, 5),
 (0, 46),
 (28, 33),
 (7, 36),
 (10, 23),
 (49, 48),
 (0, 31),
 (0, 22),
 (4, 2),
 (29, 6),
 (18, 50),
 (39, 0),
 (43, 34),
 (0, 47),
 (32, 37),
 (0, 38),
 (27, 17),
 (37, 0),
 (0, 25),
 (0, 16),
 (50, 0),
 (26, 41),
 (22, 21),
 (5, 0),
 (48, 0),
 (14, 27),
 (33, 0),
 (0, 32),
 (3, 0),
 (0, 12),
 (34, 28),
```

```
 (0, 26),
 (16, 9),
 (1, 0),
 (30, 0),
 (20, 30),
 (0, 42),
 (12, 40),
 (0, 4),
 (13, 0),
 (46, 43),
 (0, 18),
 (41, 0),
 (45, 13)]
```

Solution

The solution implemented in jupyter notebook and python was solved by using the cplex API for python. A feasible solution of $\$9730.86$ was found for the objective function, 18 routes were required which translate to a fix cost of $\$300 \times 18 = \$5400$. Therefore the total cost of the VRP with this solution is $\$15130.86$. The list below show the routes for each truck:

Routes No. Arcs

x_0_49=1,x_49_48=1,x_48_3=1,x_3_0=1,,

x_0_8=1,x_8_39=1,x_39_0=1,,,

x_0_22=1,x_22_21=1,x_21_7=1,x_7_36=1,x_36_0=1,

x_0_29=1,x_29_6=1,x_6_0=1,,,

x_0_15=1,x_15_35=1,x_35_10=1,x_10_23=1,x_23_0=1,

x_0_31=1,x_31_19=1,x_19_0=1,,,

x_0_47=1,x_47_24=1,x_24_1=1,x_1_27=1,x_27_0=1,

x_0_38=1,x_38_14=1,x_14_17=1,x_17_0=1,,

x_0_12=1,x_12_40=1,x_40_2=1,x_2_0=1,,

x_0_18=1,x_18_50=1,x_50_0=1,,,

x_0_45=1,x_45_13=1,x_13_0=1,,,

x_0_16=1,x_16_9=1,x_9_20=1,x_20_30=1,x_30_0=1,

x_0_32=1,x_32_37=1,x_37_0=1,,,

x_0_42=1,x_42_44=1,x_44_0=1,,,

x_0_46=1,x_46_43=1,x_43_34=1,x_34_28=1,x_28_33=1,x_33_0=1

x_0_4=1,x_4_11=1,x_11_5=1,x_5_0=1,,