Homework 4 - VRPPD

Lorenzo Sciotto, Jacopo Dallafior, Stefano Tonini*

*Group representative

At least 1000 words per homework (without code)

1. Problem description

The assignment that was proposed to us is a distribution problem of finished goods. Specifically, we were asked to imagine a situation for which a food and beverage company asked us to provide an efficient schedule of the vehicles every day to optimize their routes, in particular to find the optimal sequence of customers visits that each vehicle should perform to minimize their total travel time.This type of task is a fundamental phase for logistics and has an influence on various aspects such as the company's economy, environmental health, as it is a factor in CO2 production, and also social, as it involves physical subjects. The distribution and transportation problem falls into the category of Vehicle Routing Problems (VRP). Its resolution involves the development of a mathematical method that reduces the vehicle's route.

The mathematical approach that is used, schematizes the customers that will have to be visited as a set of nodes for which the fleet of vehicles available will have to visit them, respecting some specific restrictions. For example the maximum time for which a vehicle can work or the maximum distance for which the system can act.

The objective is to identify a path for the vehicle that ensures visiting a specific number of customers in compliance with the given restrictions.

The VRP is characterized by different figures that must be defined within it:

The nodes that can represent the customers to be satisfied and the warehouse are distinguished by their geographical location, the demand of product to be picked up or delivered and the loading and unloading time of the goods.

The arcs represent the road to follow.

The vehicles that we have are distinguished by their maximum capacity and in some cases by their cost.

The drivers of the vehicles can be distinguished in relation to the duration of their work shift and the maximum time allowed for overtime work.

Through the information listed we could thus be able to obtain the path that the vehicle will have to take to complete the shipment or collection.

Specifically, our assignment, is a version of the VRP called pickup and delivery and it differs from the following characteristics:

-each vehicle starts and ends its route from the central depot

-the delivery nodes are visitable only once

-the vehicles will be able to visit the shipping nodes and the pickup nodes without a division based on the task

-the pickup nodes must be visited before the corresponding delivery node

-the vehicle's capacity must always be checked after each passage for a node.

The category of my problem in particular is of the type Many to Many, it means that some nodes will have to make deliveries and others will have to make a pickup.

Problem interpretation and relevance

2. Mathematical model

The mathematical model aims to determine the optimal number of vehicles and the sequence in which they should visit nodes in order to minimize the total travel time.
To create an efficient mathematical model, it's beneficial to start by identifying the parameters.
In this scenario, the present indices that prove useful for solving my problem are:

- Set of pickup, delivery and depot nodes N where N = P U D U {0,2n+1}
- Set of pickup nodes P where P = 1,....,n
- Set of delivery nodes D where D = n+1,...,2n
- Set of pickup and delivery nodes N
- Set of arcs (i,j) with i,jϵ V
- Set of vehicles adopted to visit all the nodes k
- Demand at node i ϵ P where the value = 0 is related to the depot and the negative value for the delivery nodes
- Travel time of the arc (i,j) t_{ij}
- Service time at node i to pickup or dropoff products
- Maximum duration of a route L
- Maximum working time
- Vehicle capacity Q

The objective function aims to minimize the total travel time of the arcs traversed by all vehicles.

$$minimize \sum_{k\in K}\sum_{(i,j)\in A} t_{ij} * x_{ijk}$$

The constraint to take in consideration in this scenario are:

1. Each request must be served exactly once

$$\sum_{k\in K}\sum_{j\in V\setminus\{0\},\, i\neq j} x_{ijk} = 1, \ \ \forall i\in P$$

2. Pickup and delivery nodes of same request must be assigned to the same vehicle

$$\sum_{j\in V\setminus\{0\}, i\neq j} x_{ijk} - \sum_{j\in V\setminus\{0\}, n+i\neq j} x_{n+i,jk} = 0\ , \ \ \forall i\in P,\ \forall k\in K$$

3. Each vehicle travels at most one route starting at the depot

$$\sum_{j\epsilon V\setminus\{0\}} x_{0,j,k} \leq 1 \qquad \forall k\epsilon K$$

4. Each vehicle travels at most one route ending at the depot

$$\sum_{i\epsilon V\backslash\{2n+1\}} x_{i,2n+1,k} \leq 1 \quad \forall k\epsilon K$$

5. The number of arcs entering and exiting each node must be the same

$$\sum_{j\epsilon V\backslash\{0\},\, i\neq j} x_{ijk} - \sum_{j\epsilon V\backslash\{2n+1\}, i\neq j} x_{j,i,k} \quad \forall k\epsilon K,\ i\in N$$

6. Consistency of time

$$T_{ik} + s_i + t_{ij} - T_{jk} \leq \left(1 - x_{ijk}\right) * M \quad \forall i\in V\backslash\{2n+1\},\ \forall j\in V\backslash\{0\},\ \forall k\in K,\ i\neq j$$

7. Consistency of load variables

$$C_{ik} + q_j - C_{ij} \leq \left(1 - x_{ijk}\right) * M \quad \forall k\in K,\ \forall(i,j)\in A,\ i\neq 2n+1,\ j\neq 0$$

8. Each pickup point must be visited before the corresponding delivery one

$$T_{n+i,k} - T_{ik} - s_i - t_{i,n+i} \geq 0 \quad \forall i\in P,\ \forall k\in K$$

9. Vehicle capacity constraint

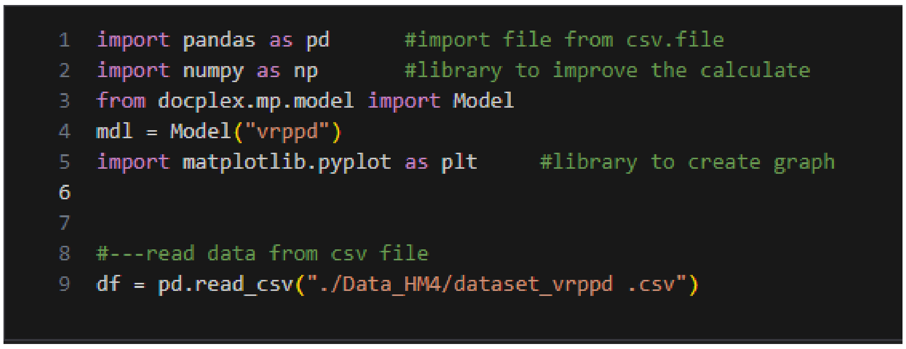
$$\left\{0, q_i\right\} \leq C_{ik} \leq \left\{Q, Q + q_i\right\} \quad \forall i\in V, \forall k\in K$$

10. Not negativity

$$C_{ik} \geq 0 \quad \forall k \in K,\ \forall i \in V$$
$$T_{ik} \geq 0 \quad \forall k \in K,\ \forall i \in V$$

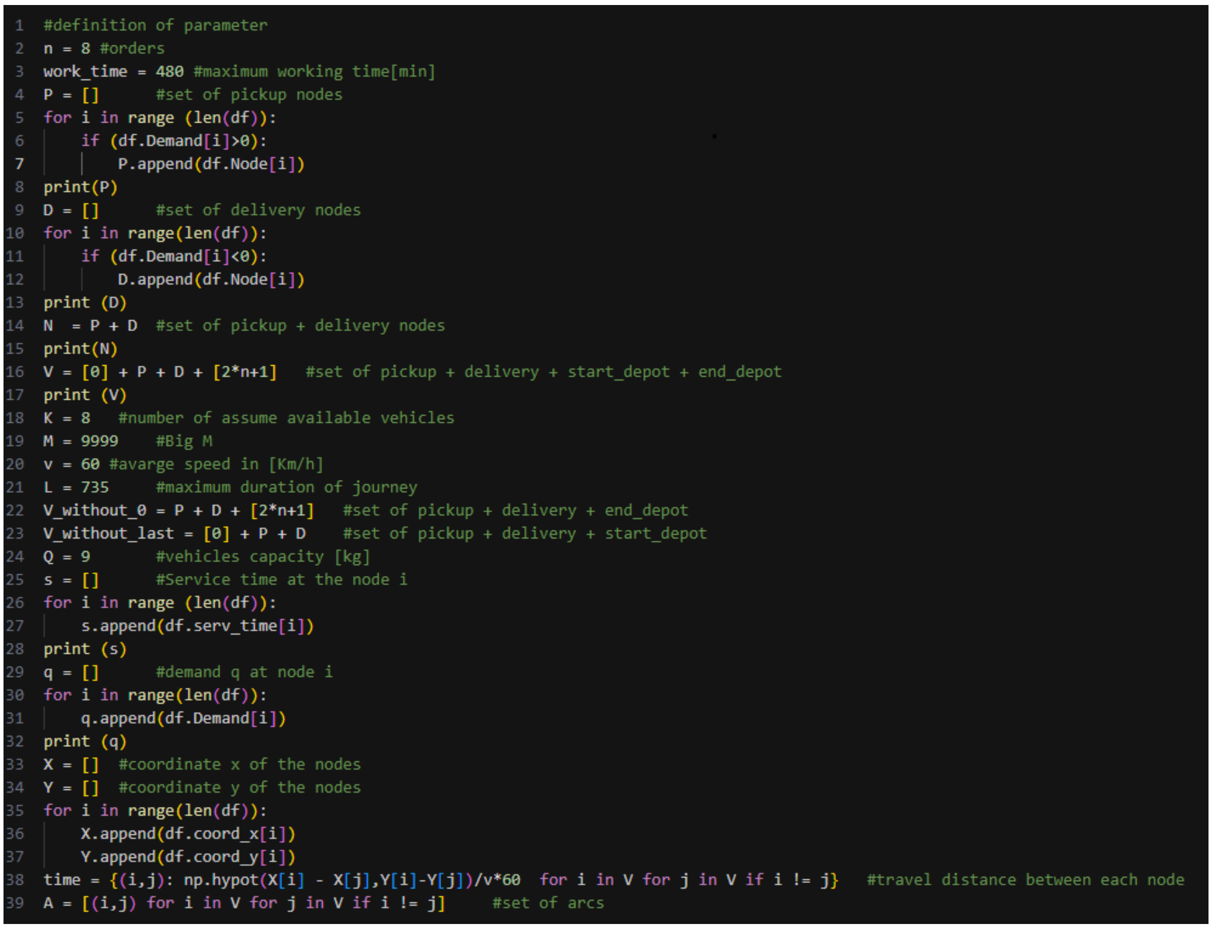
3. Main code components

To make a well-structured code, we initially wrote the libraries used in the code with their respective abbreviation must be imported (‘pandas’, ‘numpy’, ‘matplotlib’), the solver CPLEX and was imported the csv file containing some necessary information that define some parameters for the mathematical model.



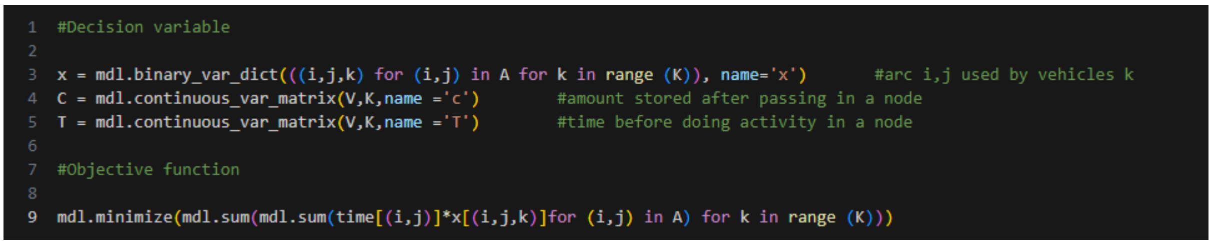
```python
import pandas as pd       #import file from csv.file
import numpy as np        #library to improve the calculate
from docplex.mp.model import Model
mdl = Model("vrppd")
import matplotlib.pyplot as plt     #library to create graph


#---read data from csv file
df = pd.read_csv("./Data_HM4/dataset_vrppd .csv")
```

After reporting the parameters provided by the problem, special attention has been given to some of them to arrange them efficiently for use.



```
#definition of parameter
n = 8 #orders
work_time = 480 #maximum working time[min]
P = []      #set of pickup nodes
for i in range (len(df)):
    if (df.Demand[i]>0):
        P.append(df.Node[i])
print(P)
D = []      #set of delivery nodes
for i in range(len(df)):
    if (df.Demand[i]<0):
        D.append(df.Node[i])
print (D)
N  = P + D  #set of pickup + delivery nodes
print(N)
V = [0] + P + D + [2*n+1]   #set of pickup + delivery + start_depot + end_depot
print (V)
K = 8   #number of assume available vehicles
M = 9999    #Big M
v = 60 #avarge speed in [Km/h]
L = 735     #maximum duration of journey
V_without_0 = P + D + [2*n+1]   #set of pickup + delivery + end_depot
V_without_last = [0] + P + D    #set of pickup + delivery + start_depot
Q = 9       #vehicles capacity [kg]
s = []      #Service time at the node i
for i in range (len(df)):
    s.append(df.serv_time[i])
print (s)
q = []      #demand q at node i
for i in range(len(df)):
    q.append(df.Demand[i])
print (q)
X = []  #coordinate x of the nodes
Y = []  #coordinate y of the nodes
for i in range(len(df)):
    X.append(df.coord_x[i])
    Y.append(df.coord_y[i])
time = {(i,j): np.hypot(X[i] - X[j],Y[i]-Y[j])/v*60  for i in V for j in V if i != j}   #travel distance between each node
A = [(i,j) for i in V for j in V if i != j]     #set of arcs
```

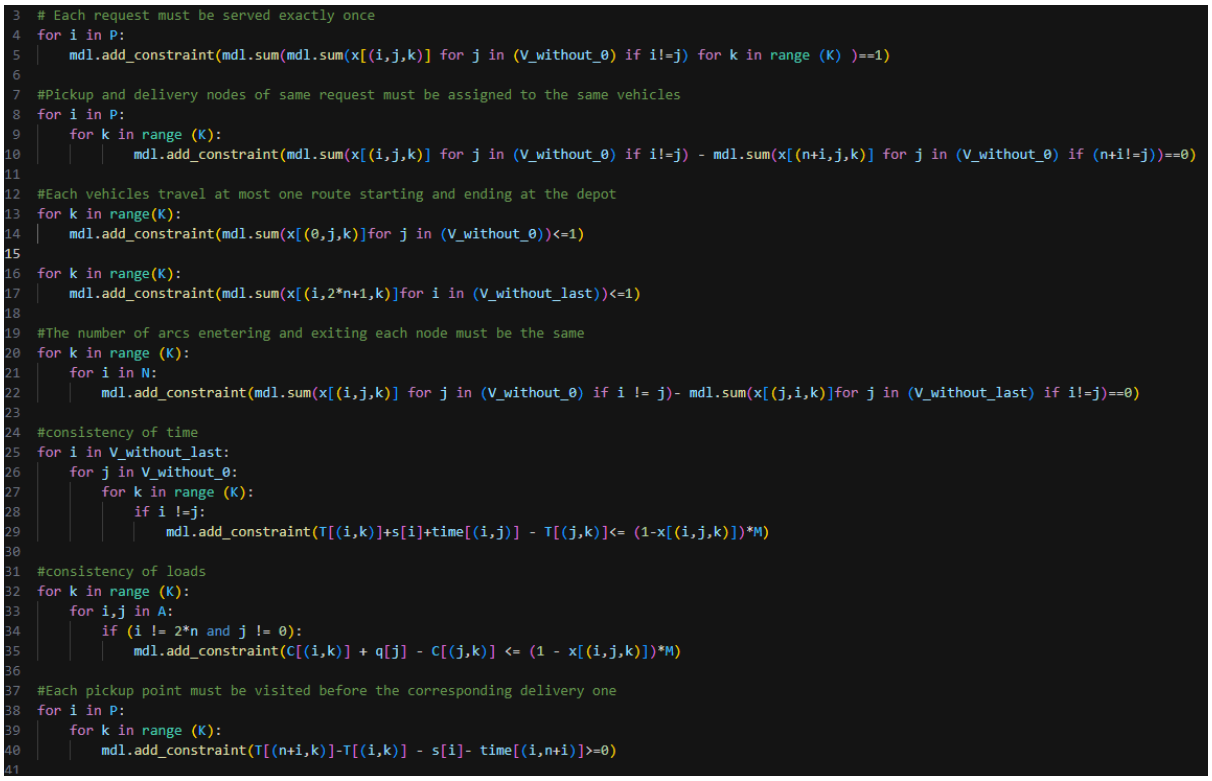
We defined the variable needed to solve the problem:
X as a dictionary of binary variables that indicates whether the arc i,j is used by the vehicle k.
C as a continuous variable that indicates the quantity stored after passing through the node.
T as a continuous variable that keeps the activity time before performing the activity at the node.
Then we defined the Objective function of the mathematical model



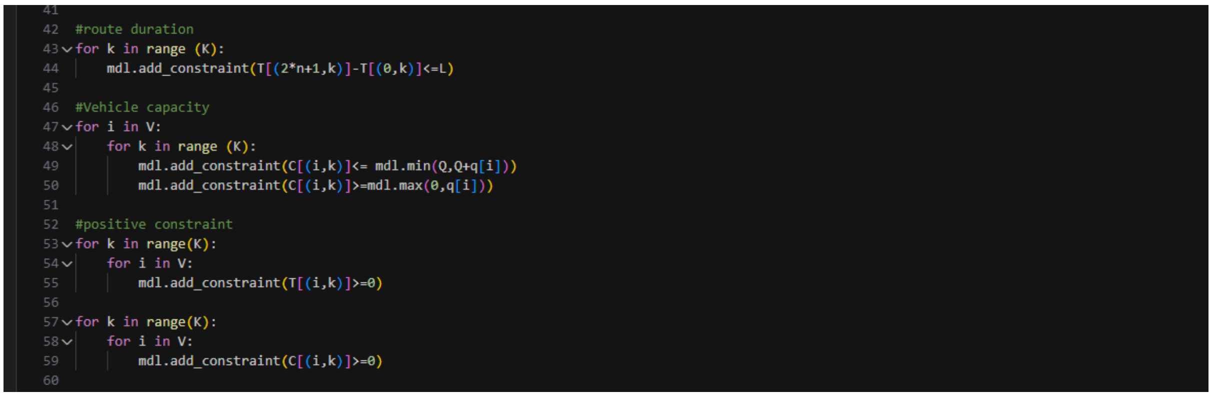
```
#Decision variable

x = mdl.binary_var_dict(((i,j,k) for (i,j) in A for k in range (K)), name='x')      #arc i,j used by vehicles k
C = mdl.continuous_var_matrix(V,K,name ='c')        #amount stored after passing in a node
T = mdl.continuous_var_matrix(V,K,name ='T')        #time before doing activity in a node

#Objective function

mdl.minimize(mdl.sum(mdl.sum(time[(i,j)]*x[(i,j,k)]for (i,j) in A) for k in range (K)))
```

The constraints used are as said in the mathematical model:

(1) Each request must be served exactly once
(2) Pickup and delivery nodes of same request must be assigned to the same vehicle
(3) Each vehicle travels at most one route starting and ending at the depot
(4) The number of arcs entering and exiting each node must be the same
(5) Consistency of time
(6) Consistency of loads
(7) Each pickup point must be visited before the corresponding delivery one
(8) Route duration
(9) Vehicle capacity
(10) Positivity of variables

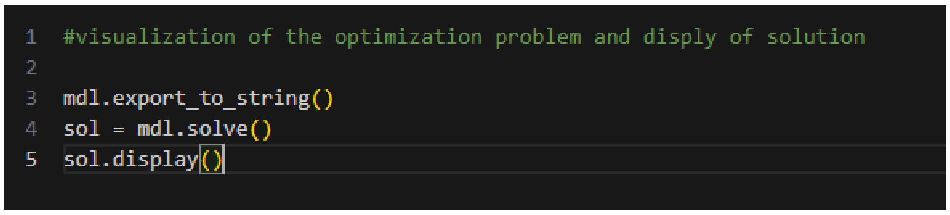


```python
# Each request must be served exactly once
for i in P:
    mdl.add_constraint(mdl.sum(mdl.sum(x[(i,j,k)] for j in (V_without_0) if i!=j) for k in range (K) )==1)

#Pickup and delivery nodes of same request must be assigned to the same vehicles
for i in P:
    for k in range (K):
            mdl.add_constraint(mdl.sum(x[(i,j,k)] for j in (V_without_0) if i!=j) - mdl.sum(x[(n+i,j,k)] for j in (V_without_0) if (n+i!=j))==0)

#Each vehicles travel at most one route starting and ending at the depot
for k in range(K):
    mdl.add_constraint(mdl.sum(x[(0,j,k)]for j in (V_without_0))<=1)

for k in range(K):
    mdl.add_constraint(mdl.sum(x[(i,2*n+1,k)]for i in (V_without_last))<=1)

#The number of arcs enetering and exiting each node must be the same
for k in range (K):
    for i in N:
        mdl.add_constraint(mdl.sum(x[(i,j,k)] for j in (V_without_0) if i != j)- mdl.sum(x[(j,i,k)]for j in (V_without_last) if i!=j)==0)

#consistency of time
for i in V_without_last:
    for j in V_without_0:
        for k in range (K):
            if i !=j:
                mdl.add_constraint(T[(i,k)]+s[i]+time[(i,j)] - T[(j,k)]<= (1-x[(i,j,k)])*M)

#consistency of loads
for k in range (K):
    for i,j in A:
        if (i != 2*n and j != 0):
            mdl.add_constraint(C[(i,k)] + q[j] - C[(j,k)] <= (1 - x[(i,j,k)])*M)

#Each pickup point must be visited before the corresponding delivery one
for i in P:
    for k in range (K):
        mdl.add_constraint(T[(n+i,k)]-T[(i,k)] - s[i]- time[(i,n+i)]>=0)
```



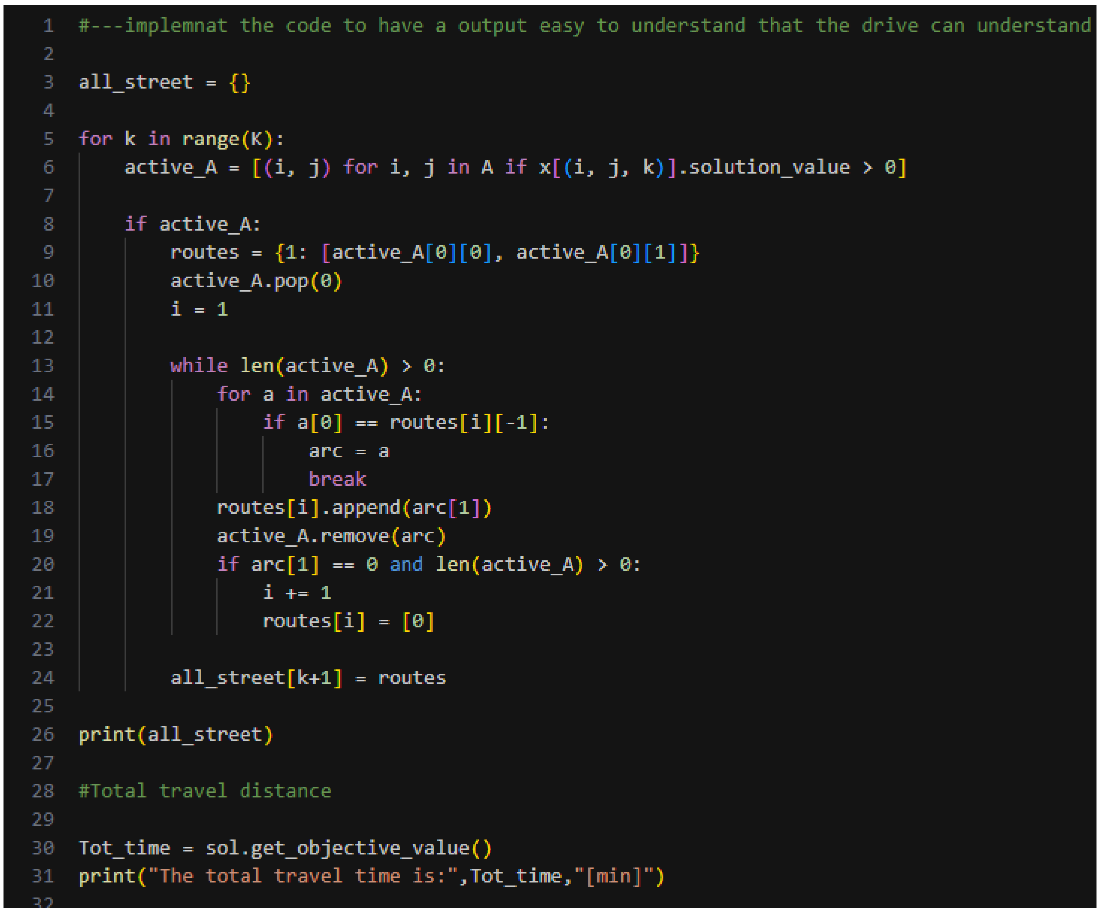
```python
#route duration
for k in range (K):
    mdl.add_constraint(T[(2*n+1,k)]-T[(0,k)]<=L)

#Vehicle capacity
for i in V:
    for k in range (K):
        mdl.add_constraint(C[(i,k)]<= mdl.min(Q,Q+q[i]))
        mdl.add_constraint(C[(i,k)]>=mdl.max(0,q[i]))

#positive constraint
for k in range(K):
    for i in V:
        mdl.add_constraint(T[(i,k)]>=0)

for k in range(K):
    for i in V:
        mdl.add_constraint(C[(i,k)]>=0)
```

Using the upcoming code it is possible to display the solution and the final value of the objective function

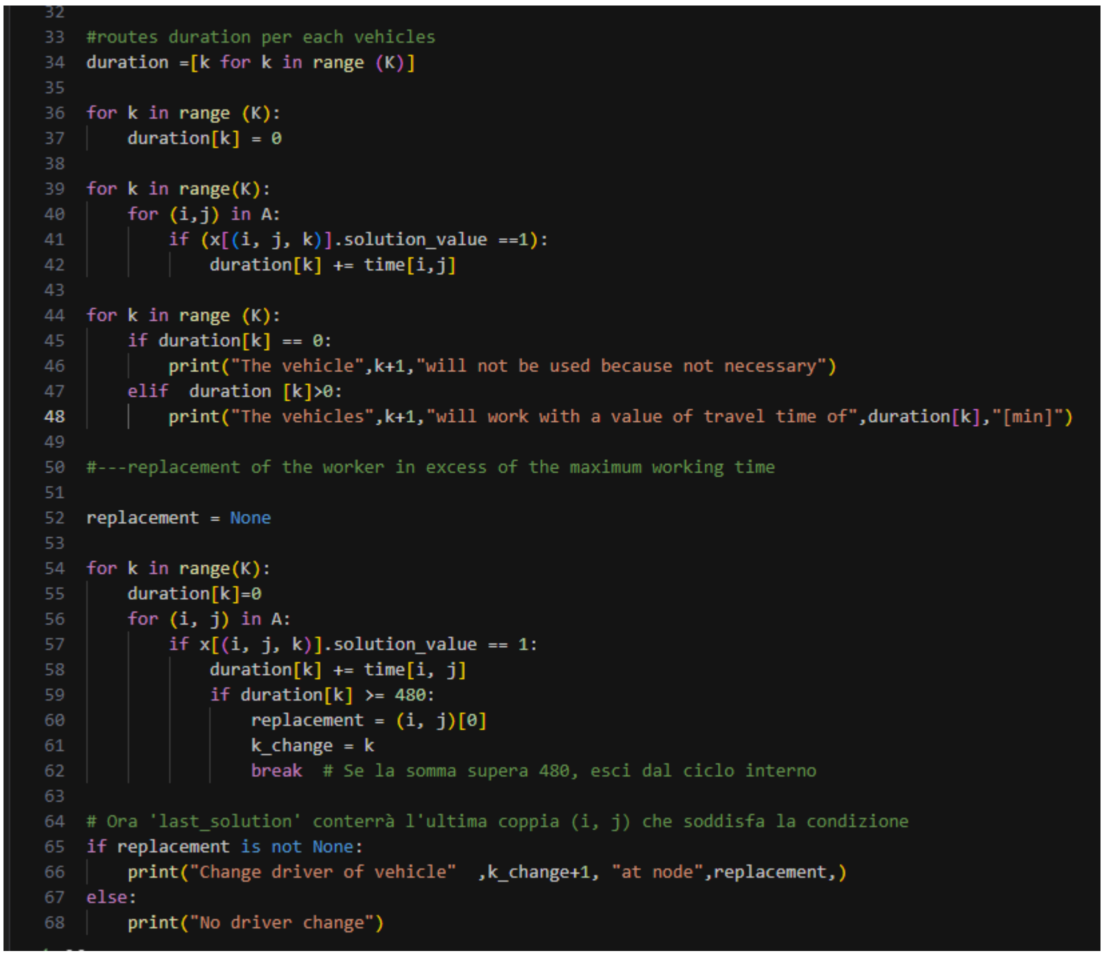


```python
#visualization of the optimization problem and disply of solution

mdl.export_to_string()
sol = mdl.solve()
sol.display()
```

In conclusion, the final part of the code is structured in order to display the solution found in a better and easy way



```python
#---implemnat the code to have a output easy to understand that the drive can understand

all_street = {}

for k in range(K):
    active_A = [(i, j) for i, j in A if x[(i, j, k)].solution_value > 0]

    if active_A:
        routes = {1: [active_A[0][0], active_A[0][1]]}
        active_A.pop(0)
        i = 1

        while len(active_A) > 0:
            for a in active_A:
                if a[0] == routes[i][-1]:
                    arc = a
                    break
            routes[i].append(arc[1])
            active_A.remove(arc)
            if arc[1] == 0 and len(active_A) > 0:
                i += 1
                routes[i] = [0]

        all_street[k+1] = routes

print(all_street)

#Total travel distance

Tot_time = sol.get_objective_value()
print("The total travel time is:",Tot_time,"[min]")
```



```python
#routes duration per each vehicles
duration =[k for k in range (K)]

for k in range (K):
    duration[k] = 0

for k in range(K):
    for (i,j) in A:
        if (x[(i, j, k)].solution_value ==1):
            duration[k] += time[i,j]

for k in range (K):
    if duration[k] == 0:
        print("The vehicle",k+1,"will not be used because not necessary")
    elif  duration [k]>0:
        print("The vehicles",k+1,"will work with a value of travel time of",duration[k],"[min]")

#---replacement of the worker in excess of the maximum working time

replacement = None

for k in range(K):
    duration[k]=0
    for (i, j) in A:
        if x[(i, j, k)].solution_value == 1:
            duration[k] += time[i, j]
            if duration[k] >= 480:
                replacement = (i, j)[0]
                k_change = k
                break  # Se la somma supera 480, esci dal ciclo interno

# Ora 'last_solution' conterrà l'ultima coppia (i, j) che soddisfa la condizione
if replacement is not None:
    print("Change driver of vehicle"  ,k_change+1, "at node",replacement,)
else:
    print("No driver change")
```

Also, in the last lines of code, a graph has been added to represent the route that the two vehicles will need to take to fulfill all the analyzed requests.

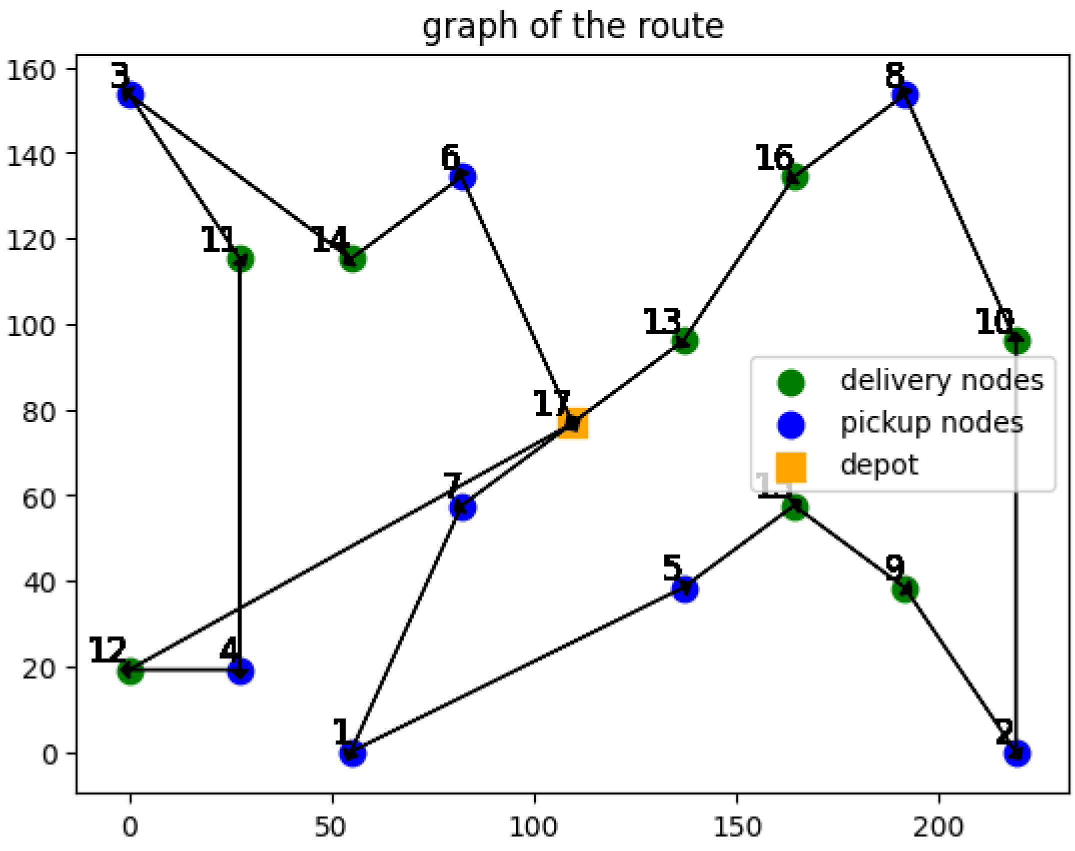
```python
#---implementation of the code to have a graph output
plt.scatter(X[n+1:-1], Y[n+1:-1], s=80, marker='o', label='delivery nodes', c='green')
plt.scatter(X[1:n+1], Y[1:n+1], s=80, marker='o', label='pickup nodes', c='blue')
plt.scatter(X[0], Y[0], s=100, marker='s', label='depot', c='orange' )

for (i,j) in A:
    for k in range(K):
        if (sol.get_value(x[(i,j,k)])==1):
            plt.arrow(X[i],Y[i],X[j]-X[i],Y[j]-Y[i],head_width=4, head_length=2, fc='black', ec='black')
        if (i !=0):
            plt.text(X[i],Y[i],i,ha='right', va='bottom', fontsize=12, color='black')

plt.legend()
plt.title('graph of the route')
```

4. Results and insights

Following the resolution of the problem, the sequence of nodes to be visited in order to achieve the minimum total travel time of 1033.8 minutes has been determined, and this is achieved by using two vehicles. Additionally, a node has been identified where a driver change must occur to comply with the constraint that a driver can work for a maximum of 8 hours.

```
{3: {1: [0, 6, 14, 3, 11, 4, 12, 17]}, 8: {1: [0, 7, 1, 5, 15, 9, 2, 10, 8, 16, 13, 17]}}
The total travel time is: 1033.8018464193374 [min]
The vehicle 1 will not be used because not necessary
The vehicle 2 will not be used because not necessary
The vehicles 3 will work with a value of travel time of 458.2243552428703 [min]
The vehicle 4 will not be used because not necessary
The vehicle 5 will not be used because not necessary
The vehicle 6 will not be used because not necessary
The vehicle 7 will not be used because not necessary
The vehicles 8 will work with a value of travel time of 575.577491176467 [min]
Change driver of vehicle 8 at node 13
```

Finally, a graph has been generated to illustrate the routes that the two vehicles must take to minimize the total travel time.


graph of the route
160
140
120
100
80
60
40
20
0
0
50
100
150
200
3
8
6
16
11
14
13
10
17
7
15
5
9
12
4
1
2
● delivery nodes
● pickup nodes
■ depot