Project 1

E14086020 洪緯宸

Setting Variables

```
for d in Demand:
    y[d] = model.addVar(vtype = "B",name = "y(%s)"%d)
    for s in Supply:
        x[d,s] = model.addVar(vtype="B",name = "x(%s,%s)"%(d,s))
        b[d,s] = model.addVar(lb = 0, vtype="C", name = "b(%s,%s)"%(d,s))
model.update()
```

- **x**: variable set represent if the No.d car park into the No.s parking lot, so it is a boolean variable, if No.d car park in No.s parking lot, then x[d,s]=1
- Y: variable set represent the penalty decision variable. If No.d car not park in the lot, then y[d] = 1
- b: variable set represent the benefit (penalty/R) that each car “d” park in parking lot “s”, we will obtain. ie. b[d,s] = penalty[d]/ distance[location_d[d],location_s[s]

Adding constraint

```
#demand constraint one car can at most park in one lot
for d in Demand:
    model.addConstr(quicksum(x[d,s] for s in Supply) <= 1,name = "Demand constraint")
```

$$\sum_{j=1}^{s} Xij \leq 1 \ \forall i = 1 \ldots d$$

- Adding constraint on one car can at most park in one parking lot
- Time complexity : O(n)

```
#type constraint
for d in Demand:
    for s in Supply:
        if Type_d[d] > Type_s[s]:
            model.addConstr(x[d,s] == 0, name = "Type constraint")
```

- Adding constraint on type demand if car_type > supply_type, then x[d,s] ==0
- Time complexity : O(n*m)

Adding constraint

```
#time constraint
for i in Demand:
    for j in range(i+1,Demand_num+1):
        if check_time_overlap(Time[i],Time[j]) == True:
            for s in Supply:
                model.addConstr((x[i,s]+x[j,s]) <= 1, name = "Time overlap constraint" )
```

- Adding constraint on time, compare every pair of demand check if the time overlap. If overlapping, then add constraint that two car can’t park in the same lot

$$Xaj + Xbj \leq 1 \ \forall j = 1 \ldots . s$$

- Time complexity : O(n^2)

Adding constraint

```
#distance constriant
for d in Demand:
    for s in Supply:
        if distance(Location_d[d],Location_s[s]) > 20:
            model.addConstr(x[d,s] == 0, name = "distance constraint")
```

- Constraint on distance : loop over each car to every supply, if the distance is greater than the R, then add the constraint that Xij =0

Setting variable constraint

```
#penalty contraint sum(x[d,s] for Supply)+y ==1
for d in Demand:
    model.addConstr(quicksum(x[d,s] for s in Supply) + y[d] == 1, name = "penalty constraint")
```

- Setting variable Y[d] opposite from whether the car park or not, so if the car is parked, the penalty variable Y[d] = 0

$$\sum_{j=0}^{s} Xij + Yi = 1 \ \forall i = 1 \dots d$$

- Time complexity = O(n)

Setting variable constraint

```
#benefit constraint
for d in Demand:
    for s in Supply:
        radius = distance(Location_d[d],Location_s[s])
        model.addConstr(b[d,s] == (Penalty[d]/radius), name = "benefit define")
```

- Adding constraint on benefit variables
- Bij = penalty[d] / radius

Function setting

- Check if each car time overlap
- Calculate the distance

```
def check_time_overlap(a,b):
    if (a[0]<b[0]):
        if (b[0]-a[0]) < (a[1] - a[0]):
            return True
        else:
            return False
    elif (a[0]>b[0]):
        if (a[0]-b[0]) < (b[1] - b[0]):
            return True
        else:
            return False
    else :
        return True

def distance(a,b):
    return math.sqrt((a[0]-b[0])**2 + (a[1]-b[1])**2)
```

Setting objective

```
model.setObjective(
    quicksum(b[d,s]*x[d,s] for (d,s) in x)
    -quicksum(Penalty[d]*y[d] for d in Demand),
    GRB.MAXIMIZE)
```

- Maximize $\sum_{i=1}^{d}\sum_{j=1}^{s} Xij * Bij - \sum_{i=1}^{d} Pi * Yi$

Optimize setting constraint

- Observing the constraint above we can combine each other together

```
for d in Demand:
    model.addConstr(quicksum(x[d,s] for s in Supply) <= 1,name = "Demand constraint")
    model.addConstr(quicksum(x[d,s] for s in Supply) + y[d] == 1, name = "penalty constraint")
    for s in Supply:
        if Type_d[d] > Type_s[s]:
            model.addConstr(x[d,s] == 0, name = "Type constraint")
            continue
        if distance(Location_d[d],Location_s[s]) > 20:
            model.addConstr(x[d,s] == 0, name = "distance constraint")
            continue
        radius = distance(Location_d[d],Location_s[s])
        model.addConstr(b[d,s] == (Penalty[d]/radius), name = "benefit define")
```

- Cause comparing type is the simplest so we put in the first place. If the type doesn’t match, we don’t need to compare location and add the benefit. By doing so we can save lots of time, cause calculate distance cause lots of time

Comparing the before and after optimize

Before

```
Explored 1 nodes (1649 simplex iterations) in 6.15 seconds
Thread count was 8 (of 8 available processors)

Solution count 8: 19610.8 19382.5 19360.4 ... -52424
```

After

```
Explored 1 nodes (2101 simplex iterations) in 5.53 seconds
Thread count was 8 (of 8 available processors)

Solution count 7: 19610.8 19379.5 19379.3 ... -52424
```

- Decrease the time by (6.15-5.53)/6.15 = 10%
- Not enough!!
- The most time consume is in comparing time, cause it time complexity is O(n^2) on demand if the demand increase the time comsume will be square.

Optimize the time constraint

```
vari={}
for d in Demand:
    model.addConstr(quicksum(x[d,s] for s in Supply) <= 1,name = "Demand constraint")
    model.addConstr(quicksum(x[d,s] for s in Supply) + y[d] == 1, name = "penalty constraint")
    for s in Supply:
        if Type_d[d] > Type_s[s]:
            model.addConstr(x[d,s] == 0, name = "Type constraint")
            vari[d,s]=0
            continue
        if distance(Location_d[d],Location_s[s]) > R:
            model.addConstr(x[d,s] == 0, name = "distance constraint")
            vari[d,s]=0
            continue
        radius = distance(Location_d[d],Location_s[s])
        vari[d,s]=1 #means the car is suitable for the supply
        model.addConstr(b[d,s] == (Penalty[d]/radius), name = "benefit define")

for i in Demand:
    for j in range(i+1,Demand_num+1):
        if check_time_overlap(Time[i],Time[j]) == True:
            for s in Supply:
                if vari[i,s] == 0 or vari[j,s] == 0:
                    continue
                model.addConstr((x[i,s]+x[j,s]) <= 1, name = "Time overlap constraint" )
```

Optimize time constraint explain & compare

```
Explored 1 nodes (1739 simplex iterations) in 1.36 seconds
Thread count was 8 (of 8 available processors)

Solution count 5: 19610.8 19546.5 19483 ... -52424
```

- By add a variable array to record if the car is suitable for the supply, and check the array when adding the time constraint, I optimize the program by (6.15-1.36)/6.15 = 77.8%
- When solving the biggest data demand = 1000 supply = 200 it takes less less 5 mins, before optimizing I can’t even run this file

```
Explored 16 nodes (52996 simplex iterations) in 350.95 seconds
Thread count was 8 (of 8 available processors)

Solution count 10: -39931.2 -40737.9 -47510.8 ... -278990
No other solutions better than -39931.2
```