Exemple complet

1 Tutoriel de prise en main de PythonMIP (partie 2)

Dans ce document, on vous explique au travers d'un exemple comment utiliser le paquet *Python-MIP* pour résoudre des programmes linéaires (en nombres entiers) **plus complexes** avec lecture des données dans un fichier.

Nous allons écrire un code permettant de résoudre le problème de localisation d'entrepôts sous contraintes de capacité.

Dans ce problème, on dispose d'un ensemble $\mathcal{W}$ d'entrepôts potentiels et d'un ensemble $\mathcal{J}$ de clients. Un coût f_i est associé à l'ouverture d'un entrepôt $i \in \mathcal{W}$. Chaque entrepôt i possède une capacité Q_i (stockage, fabrication,...) et chaque client $j \in \mathcal{J}$ a une demande $d_j > 0$. Un coût c_{ji} est associé à chaque unité de la demande d'un client j satisfaite par l'entrepôt i. Un client peut voir sa demande satisfaite par plusieurs entrepôts. Il s'agit de décider de l'ouverture d'entrepôts parmi ceux de $\mathcal{W}$ et de l'affectation de la demande des clients aux entrepôts qui auront été ouverts. L'objectif est de minimiser la somme des coûts d'ouverture et d'affectation. Une solution réalisable du problème doit satisfaire les contraintes suivantes :

- la totalité de la demande de chaque client doit être affectée aux entrepôts ouverts ;
- la somme des demandes des clients affectés à un entrepôt ouvert $i \in \mathcal{W}$ doit être plus petite ou égale à sa capacité Q_i.

Ce problème peut se modéliser en utilisant la programmation linéaire en nombres entiers. Soit z_i la variable binaire égale à 1 si et seulement si l'entrepôt $i \in \mathcal{W}$ est ouvert. Soit y_{ji} la variable continue indiquant la fraction de la demande du client $j \in \mathcal{J}$ servie par l'entrepôt i. On peut écrire le programme linéaire en nombres entiers suivant :

$$
\begin{aligned}
\min \quad & \sum_{i \in \mathcal{W}} f_i z_i + \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{W}} c_{ji} y_{ji} && && (1)\\
\text{s.c.} \quad & \sum_{i \in \mathcal{W}} y_{ji} = 1 && \forall j \in \mathcal{J} && (2)\\
& \sum_{j \in \mathcal{J}} d_j y_{ji} \leq Q_i z_i && \forall i \in \mathcal{W} && (3)\\
& z_i \in \{0,1\} && \forall i \in \mathcal{W} && (4)\\
& y_{ji} \leq 1 && \forall j \in \mathcal{J}, \forall i \in \mathcal{W} && (5)\\
& y_{ji} \geq 0 && \forall j \in \mathcal{J}, \forall i \in \mathcal{W} && (6)
\end{aligned}
$$

La fonction objectif (1) consiste à minimiser la somme des coûts d'ouverture des entrepôts et des coûts d'affectations des clients. Les contraintes (2) imposent la satisfaction totale de la demande de

chacun des clients. Les contraintes (3) imposent le respect des capacités de chacun des entrepôts. Enfin, les contraintes (4) et (5) définissent le domaine des variables.

1.1 Lire les données dans un fichier

Voyons comme lire les données d'un fichier. Ces données sont organisées dans le fichier *cap41.txt* de la manière suivante :
#Taille des données (ligne 1)
$|\mathcal{W}|$ $|\mathcal{J}|$
#Pour chaque entrepôt $i \in \mathcal{W}$ *(ligne 2 à* $1 + |\mathcal{W}|$*)*
Q_i f_i
#Pour chaque client $j \in \mathcal{J}$*(ligne* $2 + |\mathcal{W}|$ *à* $2 + |\mathcal{W}| + 2|\mathcal{J}|$*)*
d_j
c_{j1} c_{j2} ... $c_{j|\mathcal{W}|}$

Nous considérons que les données seront toujours dans ce format. Le code suivant permettra alors de résoudre plusieurs jeux de données en n'ayant qu'à modifier le nom du fichier à lire.

[ ]:
```
#chemin relatif vers le fichier (l'utilisation .. permet de revenir au dossier␣
 ↪parent)
datafileName = 'data_CWFL/cap41.txt'

#ouverture du fichier, le ferme automatiquement à la fin et gère les exceptions
with open(datafileName, "r") as file:
    # lecture de la 1ère ligne et séparation des éléments de la ligne
    # dans un tableau en utilisant l'espace comme séparateur
    line = file.readline()
    lineTab = line.split()

    # la valeur de la 1ère case correspond au nombre d'entrepôts
    # (attention de penser à convertir la chaîne de caractère en un entier)
    nb_warehouses = int(lineTab[0])

    # la valeur de la 2ème case correspond au nombre de clients
    nb_customers = int(lineTab[1])

    # création d'un tableau qui stockera les capacités des entrepôts
    capacity = []
    # création d'un tableau qui stockera les coûts d'ouverture des entrepôts
    opening_cost = []

    # pour chaque ligne contenant les informations sur les entrepôts
    for i in range(nb_warehouses):
        # lecture de la ligne suivante et séparation des éléments de la ligne
        # dans un tableau en utilisant l'espace comme séparateur
        line = file.readline()
        lineTab = line.split()
```

```
        # ajout de l'élément de la 1ère case au tableau qui contient les␣
↪capacités
        capacity.append(int(lineTab[0]))
        # ajout de l'élément de la 2ème case au tableau qui contient les coûts␣
↪d'ouverture
        opening_cost.append(float(lineTab[1]))

    # création d'un tableau qui stockera les demandes des clients
    demand = []
    # création d'un tableau qui stockera les tableaux de coûts d'affectation aux␣
↪entrepôts de chaque client
    assignment_cost = []

    # pour chaque ligne contenant les informations sur les clients
    for j in range(nb_customers):
        # lecture de la ligne suivante
        line = file.readline()
        # ajoute l'élément de la 1ère case au tableau qui contient la demande␣
↪des clients
        demand.append(int(line.split()[0]))

        # création du tableau des coûts d'affectation du client j aux entrepôts
        cost =[]

        # lecture de la ligne suivante et séparation des éléments de la ligne
        # dans un tableau en utilisant l'espace comme séparateur
        line = file.readline()
        lineTab = line.split()

        for i in range(nb_warehouses):
            # ajout de l'élément de la case i au tableau contenant
            # les coûts d'affectation du client j aux entrepôts
            cost.append(float(lineTab[i]))

        # ajout du tableau cost au tableau aux entrepôts au tableau
        # contenant les coûts d'affectations de tous les clients au dépôt
        assignment_cost.append(cost)

# Affichage des informations lues
print("Nombre d'entrepôts = ", nb_warehouses)
print("Capacité des entrepôts = ", capacity)
print("Coût d'ouverture des entrepôts = ", opening_cost)
print("Nombre de clients = ", nb_customers)
print("Demande des clients = ", demand)
#for j in range(nb_customers):
#    print(assignment_cost[j])
```

1.2 Création et définition du modèle

On commence par faire les imports nécessaires.

[ ]:
```
# Import du paquet PythonMIP et de toutes ses fonctionnalités
from mip import *
# Import du paquet time pour calculer le temps de résolution
import time
```

On crée le modèle.

[ ]:
```
# Création du modèle vide
model = Model(name = "CWFL", solver_name="CBC")  # Utilisation de CBC (remplacer␣
 ↪par GUROBI pour utiliser cet autre solveur)
```

On va maintenant créer les variables.

[ ]:
```
# Création des variables z et y
z = [model.add_var(name="z(" + str(i) + ")", lb=0, ub=1, var_type=BINARY) for i␣
 ↪in range(nb_warehouses)]
y = [[model.add_var(name="y(" + str(j) + "," + str(i) + ")", lb=0, ub= 1,␣
 ↪var_type=CONTINUOUS) for i in range(nb_warehouses)] for j in␣
 ↪range(nb_customers)]

#Il aurait été possible de créer les variables comme ceci (cela est équivalent)
#z = []
#for i in range(nb_warehouses):
#    z.append(model.add_var(name="z(" + str(i) + ")", var_type=BINARY))
#y = []
#for j in range(nb_customers):
#    y.append([])
#    for i in range(nb_warehouses):
#        y[j].append(model.add_var(name="y(" + str(j) + "," + str(i) + ")",␣
 ↪lb=0, ub= 1, var_type=CONTINUOUS))
```

On ajoute ensuite la fonction objectif au modèle à l'aide de la commande `model+=` et la fonction `xsum`

[ ]:
```
# Ajout de la fonction objectif au modèle
model.objective = minimize(xsum(opening_cost[i] * z[i] for i in␣
 ↪range(nb_warehouses))+xsum(assignment_cost[j][i]*y[j][i] for j in␣
 ↪range(nb_customers) for i in range(nb_warehouses)) )
```

On ajoute ensuite les contraintes au modèle à la l'aide de la commande `model+=`et la fonction `xsum`

[ ]:
```
# Ajout des contraintes au modèle
for j in range(nb_customers):
    model.add_constr(xsum([y[j][i] for i in range(nb_warehouses)]) == 1)  #␣
 ↪Contraintes (2)
```

```
for i in range(nb_warehouses):
    model.add_constr(xsum([demand[j]*y[j][i] for j in range(nb_customers)]) <=␣
 ↪capacity[i]*z[i])  # Contraintes (3)
```

On peut écrire le modèle `model` que l'on a généré dans un fichier en utilisant la fonction `write`.

[ ]:
```
# Ecrire le modèle (ATTENTION ici le modèle est très grand)
model.write("cwfl.lp") #à décommenter si vous le souhaitez
```

Nous allons maintenant voir comment indiquer des seuils d'optimalité au solveur lors de la résolution d'un programme linéaire en nombres entiers. Une solution calculée sera considérée comme optimale si ces seuils sont satisfaits dans l'état courant de la résolution.

- La modification du champ `max_mip_gap` de `model` modifie le critère lié au *gap relatif*. Si une solution avec un coût c et une borne inférieure l sont disponibles et $\frac{(c-l)}{l} <$ `max_mip_gap` (`max_mip_gap`=1e-4 par défaut), alors la résolution sera stoppée et la solution de coût c retournée comme étant une solution optimale.
- La modification du champ `max_mip_gap_abs` de `model` modifie le critère lié au *gap absolu*. Si une solution avec un coût c et une borne inférieure l sont disponibles et $c - l <$ `max_mip_gap_abs` (`max_mip_gap_abs`=1e-10 par défaut), alors la résolution sera stoppée et la solution de coût c retournée comme étant une solution optimale.

[ ]:
```
# Indication au solveur d'un critère d'optimalité : gap relatif en dessous␣
 ↪duquel la résolution sera stoppée et la solution considérée comme optimale
model.max_mip_gap = 1e-6
# Indication au solveur d'un critère d'optimalité : gap absolu en dessous duquel␣
 ↪la résolution sera stoppée et la solution considérée comme optimale
model.max_mip_gap_abs = 1e-8
```

Il nous reste plus qu'à lancer la résolution de notre programme linéaire `model` en appelant la fonction `optimize`.

[ ]:
```
# Lancement du chronomètre
start = time.perf_counter()

# Résolution du modèle
status = model.optimize(max_seconds = 60)

# Arrêt du chronomètre et calcul du temps de résolution
runtime = time.perf_counter() - start
```

Avant de chercher à récupérer le résultat, on vérifie le status de la solution retournée par le solveur.

Selon le status, on récupère la solution obtenue (valeur des variables et de la fonction objectif) et on l'affiche.

```
[ ]: print("\n---------------------------------")
if status == OptimizationStatus.OPTIMAL:
    print("Status de la résolution: OPTIMAL")
elif status == OptimizationStatus.FEASIBLE:
    print("Status
      de la résolution: TEMPS LIMITE et UNE SOLUTION REALISABLE CALCULEE")
elif status == OptimizationStatus.NO_SOLUTION_FOUND:
    print("Status de la résolution: TEMPS LIMITE et AUCUNE SOLUTION CALCULEE")
elif status == OptimizationStatus.INFEASIBLE or status == OptimizationStatus.
 ↪INT_INFEASIBLE:
    print("Status de la résolution: IRREALISABLE")
elif status == OptimizationStatus.UNBOUNDED:
    print("Status de la résolution: NON BORNE")

print("Temps de résolution (s) : ", runtime)
print("---------------------------------")

# Si le modèle a été résolu à l'optimalité ou si une solution a été trouvée dans␣
 ↪le temps limite accordé
if model.num_solutions>0:
    print("Solution calculée")
    print("-> Valeur de la fonction objectif de la solution calculée : ",  model.
 ↪objective_value)  # ne pas oublier d'arrondir si le coût doit être entier
    print("-> Meilleure borne inférieure sur la valeur de la fonction objectif =␣
 ↪", model.objective_bound)
    for i in range(nb_warehouses):
        if (z[i].x >= 0.5):
            print("- L'entrepôt ",i , " est ouvert [capacité = ", capacity[i],␣
 ↪"] et les clients suivants lui sont affectés")
            for j in range(nb_customers):
                if (y[j][i].x >= 1e-4):
                    print("\t Client ",j, " pour ", round(y[j][i].x * 100,1), "␣
 ↪% de sa demande -> ",round(y[j][i].x * demand[j],1))
else:
    print("Pas de solution calculée")
print("---------------------------------\n")
```

1.3 Ecriture du résultat dans un fichier

Si une solution a été calculée, on l'écrit dans un fichier dans le format suivant (arbitraire) facile à lire.
#Ligne 1
Valeur de la solution
#Pour chaque entrepôt ouvert
Numéro de l'entrepôt ouvert
#Pour chaque client affecté à cet entrepôt
Numéro du client et pourcentage de la demande de ce client satisfait par l'entrepôt

[ ]:
```
if model.num_solutions>0: # Si une solution a été calculée
    solutionfileName = 'solution_cap41.txt' #nom du fichier solution
    with open(solutionfileName, 'w') as file:  #ouvre le fichier, le ferme␣
↪automatiquement à la fin et gère les exceptions
        file.write(str(model.objective_value)) #Il faut convertir les valeurs␣
↪numériques en chaîne de caractères
        file.write("\n") #Je passe à la ligne suivante
        for i in range(nb_warehouses):
            if (z[i].x >= 0.5):
                file.write(str(i))
                file.write("\n") #Je passe à la ligne suivante
                for j in range(nb_customers):
                    if (y[j][i].x >= 1e-4):
                        file.write(str(j)+" "+str(round(y[j][i].x * 100,2)))
                        file.write("\n") #Je passe à la ligne suivante
```