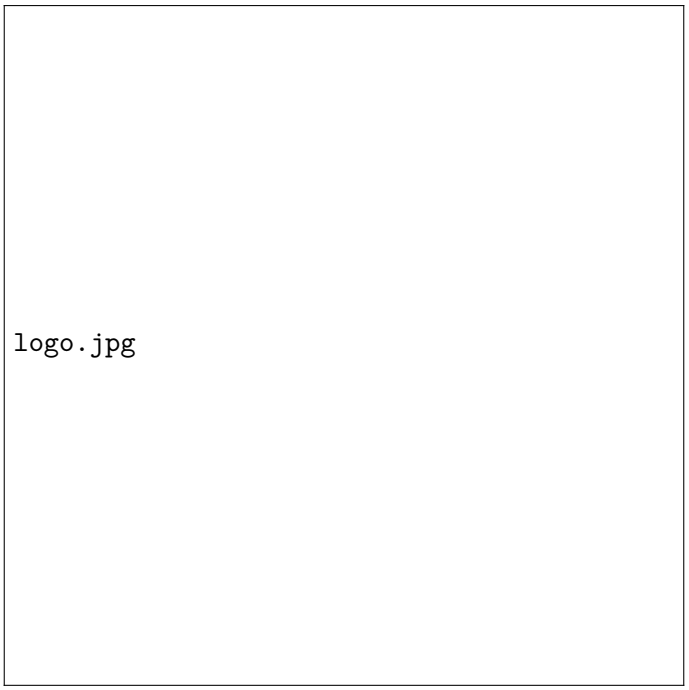


EI6IF127 Recherche Opérationnelle

# Projet : Problème du bin-packing avec objets fragiles

*Auteurs :* Laboirie Alex
Pierson Louis

*Chargé de TD :* Larroze-Jardine Sarah

28 mai 2022

# 1 Première formulation

### Question 1.

On peut borner le nombre de paquets dont on a besoin par $n$ car il y a $n$ objets.

### Question 2.

On peut modéliser le problème sous la forme d'un programme linéaire comme ceci :

$$\begin{cases} min.z = \sum_{k=1}^{n} y_k \\ s.c. \ \sum_{k=1}^{n} x_{i,k} = 1 \quad \forall i \in [\![1,n]\!] \\ \quad \sum_{i=1}^{n} x_{i,k} * w_i \leq f_i + (1 - annul_u) * M \quad \forall k \in [\![1,n]\!] \\ \quad x_{i,j} \leq y_k hspace12pt \forall i \in [\![1,n]\!], \forall k \in [\![1,n]\!] \end{cases} \tag{1}$$

### Question 3.

```
# Creation des variables x et y
x = [model1.add_var(name="x(" + str(i) + ")", lb=0, ub=1,
    var_type=BINARY) for i in range(nb_objets*nb_boites)]
y = [model1.add_var(name="y(" + str(k) + ")", lb=0, ub=1,
    var_type=BINARY) for k in range(nb_boites)]
annul = [model1.add_var(name="annul(" + str(k) + ")", lb=0, ub=1,
    var_type=BINARY) for k in range(nb_objets)]

# Ajout de la fonction objectif au modele
model1.objective = minimize(xsum(y[k] for k in range (nb_boites)))


# Ajout des contraintes au modele
for i in range(nb_objets):
    [model1.add_constr(xsum([x[i + k*nb_objets] for k in
        range(nb_boites)]) == 1)]

for k in range (nb_objets):
    [model1.add_constr(xsum(poids[i]*x[i + k*nb_objets] for
        i in range(nb_boites)) <= fragilite[i] + (1 - annul[i])*M)]

for i in range (nb_objets):
    for k in range(nb_boites):
        [model1.add_constr(x[i + k*nb_boites] <= y[k])]
```

Nous pouvons prendre $M = \sum_{i=1}^{n} f_i$ comme majorant de toutes les fragilités.

Dans la suite nous pourrons prendre $M = max(f_i)$, cette valeur ne respecte pas le modèle mais elle est facilement remplaçable à l'ordinateur par une valeur constante et donc qui respecte le modèle linéaire.

### Question 4.

Il semble en effet intéressant de remplir les boites par ordre croissant vu que le nombre d'objets que l'on peut mettre dans les boites ne dépend pas de la boite en elle même mais des objets à l'intérieur.

Pour réaliser ce remplissage des boites par ordre croissant il suffit d'ajouter la sous-condition : $y_k \leq y_{k+1} \quad \forall k \in [\![1, n-1]\!]$

### Question 5.

## 2 Seconde formulation

### Question 6.

$$\begin{cases} min.z = \sum_{i=1}^{n} r[i] \\ s.c. \ \sum_{j=1}^{n} z_{i,j} = 1 \quad \forall i \in [\![1,n]\!] \\ \quad\quad \sum_{i=1}^{n} z_{i,j} * w_i \leq f_j * r_j \quad \forall j \in [\![1,n]\!] \\ \quad\quad z_{i,j} \leq r[j] \quad \forall i \in [\![1,n]\!], \forall j \in [\![1,n]\!] \end{cases} \tag{2}$$

La condition de minimisation représente ici un nombre minimal de paquet. Comme nous traitons ici avant des représentants d'objets ce sont eux qu'il faut minimiser.

La sous-condition n°1 signifie que chaque objet doit avoir un seul et unique représentant.

La sous-condition n°2 signifie que le poids total des objets ayant le même représentant doivent avoir un poids inférieur ou égal à la fragilité de leur représentant.

Enfin la sous-condition n°3 signifie que si un objet est affecté à l'objet j alors l'objet j est un représentant.

### Question 7.

On obtient donc cette implémentation en python :

```
# Creation du modele vide
model2 = Model(name = "BPFO_2", solver_name="CBC")

# Creation des variables z et y
r = [model2.add_var(name="r(" + str(i) + ")", lb=0, ub=1,
    var_type=BINARY) for i in range(nb_objets)]
z = [model2.add_var(name="z(" + str(i) + ")", lb=0, ub=1,
    var_type=BINARY) for i in range(nb_objets*nb_objets)]

# Ajout de la fonction objectif au modele
model1.objective = minimize(xsum(r[i] for i in range (nb_objets)))

# Ajout des contraintes au modele
```

```
for i in range(nb_objets):
    [model1.add_constr(xsum(z[i + j*nb_objets] for j in range(nb_objets)) =

for j in range(nb_objets):
    [model1.add_constr(xsum(poids[j]*z[i + j*nb_objets] for i in range
        (nb_objets)) <= fragilite[j]*r[j])]

for j in range(nb_objets):
    for i in range(nb_objets):
        z[i + j*nb_objets] <= r[j]
```

Ici nous n'avons pas besoin d'introduire de variable $M$ car comme l'on connaît la fragilité minimale des objets du groupe d'objets grâce aux représentants.

**Question 8.**

**Question 9.**