MAMLOUK Haya [21107689]
OZGENC Doruk [21113927]

Sorbonne Université
2024-2025

MOGPL – Rapport de projet :
Optimisation robuste dans l'incertain total

Sommaire

Ce projet porte sur **l'optimisation robuste dans un contexte d'incertitude totale**, une problématique essentielle lorsque l'évaluation des solutions est influencée par des scénarios variés et imprévisibles. Contrairement aux méthodes d'optimisation classiques qui supposent des conditions connues, l'optimisation robuste vise à trouver des solutions performantes dans tous les scénarios possibles, même dans les cas défavorables.

Dans un premier temps, nous traiterons la linéarisation des critères maxmin et minmax regret**,** suivie de celle du critère OWA, avant de conclure avec une application à la recherche d'un chemin robuste dans un graphe.

Pour illustrer les deux premières parties, nous utiliserons le problème de sélection de projets suivant : sélectionner les projets maximisant l'utilité globale tout en tenant compte de deux scénarios d'incertitude, sous la contrainte d'un budget de 100Keuros.

projets	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
utilité dans s_1	*70*	*18*	*16*	*14*	*12*	*10*	*8*	*6*	*4*	*2*
utilité dans s_2	*2*	*4*	*6*	*8*	*10*	*12*	*14*	*16*	*18*	*70*
coût (en Keuros)	*60*	*10*	*15*	*20*	*25*	*20*	*5*	*15*	*20*	*60*

Posons pour la suite les données suivantes :

- $\boldsymbol{P}$: l'ensemble des projets
- $\boldsymbol{B}$: le budget
- $\boldsymbol{u_{ij}}$: utilité du projet *j* dans le scénario *i* avec $j \in P, i \in \{1, \ldots, n\}$
- $\boldsymbol{c_j}$: coût du projet *j* avec $j \in P$

I. Linéarisation des critères maxmin et minmax regret

a. Formulation du problème de maxmin

Le critère **maxmin** cherche à maximiser la valeur minimale obtenue dans tous les scénarios possibles. En d'autres termes, on veut trouver une solution qui garantit le meilleur résultat possible dans le pire cas.
Pour maximiser la fonction g(x) = $\min_{i=1,..,n} z_i(x)$, nous introduisons une variable continue $\boldsymbol{t}$ pour représenter le minimum des $z_i(x)$. Le problème peut alors être formulé comme un programme linéaire :

Variables :

- $\boldsymbol{x_j}$: variable binaire, $\boldsymbol{x_j} = \begin{cases} 1 \; si \; le \; projet \; j \; est \; sélectionné \\ 0 \; sinon \end{cases}$
- $\boldsymbol{t}$: variable continue représentant le minimum des $z_i(x) = \sum_{j \in P} \boldsymbol{u_{ij}} \boldsymbol{x_j}$

Fonction objectif :

$$\textbf{max } \boldsymbol{t}$$

Contraintes :

1. $\boldsymbol{t} \leq z_i(x) = \sum_{j \in P} \boldsymbol{u_{ij}} \boldsymbol{x_j} \qquad \forall i \in 1, \dots, n$
2. $\sum_{j \in P} \boldsymbol{c_j x_j} \leq \boldsymbol{B}$
3. $\boldsymbol{x_j} \in \{0,1\}, \; \forall j \in P$

Cette modélisation a été implémentée dans un fichier Python (maxmin.py), utilisant le solveur Gurobi. Une solution optimale x^*, maximisant $\boldsymbol{t}$ au sens du critère maxmin, a été déterminée. La solution obtenue est la suivante :

$x^* = (0, 1, 1, 1, 0, 0, 1, 1, 1, 0)$
$z(x^*) = (66, 66)$
La valeur de la fonction objective (maxmin) : 66

b. Formulation du problème de minmax regret

Le critère **minmax regret** cherche à minimiser le regret maximal parmi tous les scénarios possibles. Autrement dit, on cherche une solution qui réduit au minimum le "manque à gagner" dans le pire des cas.

Ce problème est résolu en deux étapes : la première consiste à calculer les regrets pour chaque scénario, et la seconde à maximiser le regret maximal à l'aide d'un programme linéaire.

Étape 1 : Calcul des regrets
Définissons les regrets $r(x, s_i) = z_i^*(x) - z_i(x) = \boldsymbol{z_i^*(x) - \sum_{j \in P} u_{ij} x_j}$ où $z_i^*(x)$ est la valeur maximale de $z_i(x)$ sur tous les x. Ainsi, pour chaque scénario i, nous devons calculer $z_i^*(x)$.

Étape 2 : Minimisation du regret maximal
Pour minimiser la fonction g(x) = $\max_{i=1,..,n} r(x, s_i)$, introduisons une variable continue $\boldsymbol{t}$ pour représenter le maximum des regrets. Le problème peut alors être formulé comme un programme linéaire.

Variables :

- $\boldsymbol{x_j}$: variable binaire, $\boldsymbol{x_j} = \begin{cases} 1 \; si \; le \; projet \; j \; est \; sélectionné \\ 0 \; sinon \end{cases}$
- $\boldsymbol{t}$: variable continue représentant le maximum des $r(x, s_i)$

Fonction objectif :

$$\textbf{max } \boldsymbol{t}$$

Contraintes :

1. $\boldsymbol{t} \geq z_i^*(x) - \sum_{j \in P} \boldsymbol{u_{ij} x_j} \qquad \forall i \in 1, \dots, n$
2. $\sum_{j \in P} \boldsymbol{c_j x_j} \leq \boldsymbol{B}$
3. $\boldsymbol{x_j} \in \{0,1\}, \; \forall j \in P$

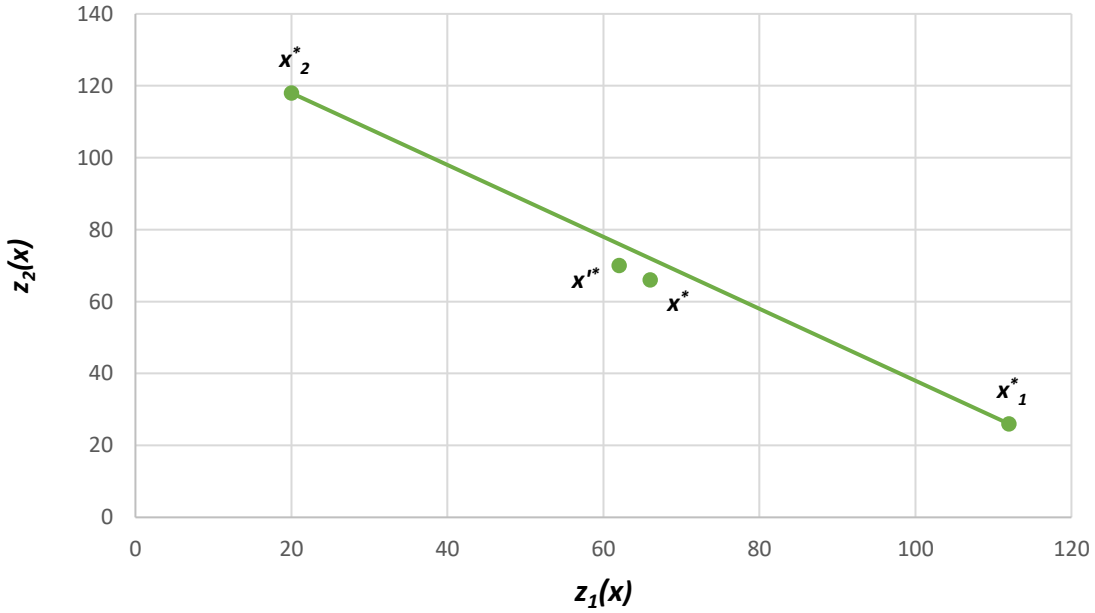
Cette modélisation a été implémentée dans un fichier Python (minmaxRegret.py), en utilisant le solveur Gurobi. Gurobi nous a permis de trouver z_1^* et z_2^* en résolvant les programmes linéaires associés aux solutions optimales x_1^* et x_2^* respectivement. Par la suite, il a résolu le programme linéaire pour déterminer la solution optimale du regret minmax, fournissant les résultats suivants :

$x_1^* = (1,1,1,0,0,0,1,0,0,0)$ et $z_1^* = 112$
$x_2^* = (0,0,0,0,0,0,1,1,1,1)$ et $z_2^* = 118$
$x'^* = (0,1,0,1,1,0,1,1,1,0)$ et $z(x'^*) = (62,70)$
$r(x'^*, s_1) = 50$ et $r(x'^*, s_2) = 48$
La valeur de la fonction objective (minmax regret) : 50

c. Représentation des points dans le plan

Nous souhaitons représenter dans le plan $z_1(x)$ et $z_2(x)$ les quatre points suivants :

$z(x_1^*) = (112,26)$
$z(x_2^*) = (20,118)$
$z(x^*) = (66,66)$
$z(x'^*) = (62,70)$



Les points $z(x^*)$ et $z(x'^*)$ ne se trouvent pas sur la ligne reliant $z(x_1^*)$ et $z(x_2^*)$. Cela démontre qu'aucune des solutions x^* et x'^* n'aurait pu être obtenue en maximisant une moyenne pondérée (avec des coefficients positifs) entre $z_1(x)$ et $z_2(x)$. Ces solutions sont spécifiques à leurs critères respectifs et ne peuvent être réduites à un compromis linéaire simple.

Si une solution optimale, telle que x^* ou x'^*, pouvait être obtenue en maximisant une moyenne pondérée, cela signifierait qu'elle correspondrait à un compromis linéaire entre $z_1(x)$ et $z_2(x)$, pondéré par des coefficients positifs. Cependant, les solutions maxmin x^* et minmax regret x'^*, ne suivent pas cette logique.

Elles sont conçues pour répondre à des critères spécifiques :

- La solution maxmin x^* vise à équilibrer les deux objectifs en minimisant le pire résultat possible.
- La solution minmax regret x'^*cherche à minimiser le regret maximal dans tous les scénarios possibles.

Ces critères ne sont pas alignés sur la logique de compromis linéaire (combinaison linéaire), car ils se concentrent sur des aspects plus complexes que la simple combinaison pondérée des deux objectifs. Ainsi, ces solutions ne peuvent être interprétées comme des résultats d'une optimisation basée sur une moyenne pondérée.

d. Généralisation du problème

L'étude des temps de résolution pour le problème de sac-à-dos robuste a été réalisée en considérant les deux critères d'optimisation : **maxmin** et **minmax regret**. Pour chaque combinaison de n∈{5,10,15} (nombre de scénarios) et p ∈ {10,15,20} (nombre de projets), 10 instances ont été générées aléatoirement. Les coûts et utilités des projets ont été choisis dans l'intervalle [1, 100], et le budget de poids a été fixé à 50% du coût total des projets. Les temps moyens de résolution ont été calculés grâce au fichier Python tps_resol.py.
Les temps moyens de résolution (en secondes) obtenus pour chaque configuration sont présentés dans le tableau suivant :

	n	**p**	Temps moyen de résolution (**maxmin**)	Temps moyen de résolution (**minmax regret**)
1	5	10	0.008989	0.022132
2	5	15	0.008715	0.026239

3	5	20	0.012839	0.028806
4	10	10	0.007990	0.055671
5	10	15	0.011154	0.066685
6	10	20	0.013054	0.053114
7	15	10	0.009295	0.061391
8	15	15	0.018638	0.057342
9	15	20	0.012659	0.068193

Les temps de résolution pour minmax Regret sont significativement plus élevés que ceux pour maxmin, indiquant une complexité supplémentaire liée aux calculs de regret. En effet, la méthode minmax Regret cherche à minimiser le regret maximal pour chaque scénario, ce qui implique de considérer une gamme plus large de possibilités et de scénarios, augmentant ainsi la charge de calcul.

Lorsque p (le nombre de projets) augmente, les temps de résolution augmentent modérément, car le problème nécessite d'examiner un plus grand nombre de variables, ce qui alourdit les calculs mais de manière relativement contrôlée. L'impact de n (le nombre de scénarios) est plus marqué, surtout pour minmax Regret, car chaque scénario ajoute des contraintes supplémentaires au modèle.

Cette analyse met en lumière l'impact des paramètres sur les contraintes du problème. L'approche maxmin reste plus rapide, car elle repose sur un calcul plus direct sans avoir à examiner tous les regrets possibles à travers les scénarios.

II. Linéarisation du critère maxOWA

a. Définition de $L_k(z)$

Soit un vecteur $z \in R$, on définit $L(z) = (L_1(z), \ldots, L_n(z))$, où $L_k(z)$ est définie par :

$$L_k(z) = \sum_{i=1}^{k} z_{(i)}$$

où $z_{(i)}$ représente le i-ème plus petit élément de z, puisque $(z_{(1)}(x), \ldots, z_{(n)}(x))$ représente le résultat d'un tri des composantes de $(z_1(x), \ldots, z_n(x))$ par ordre croissant.

On considère le programme linéaire suivant :

Fonction objectif :

$$\textbf{min} \sum_{i=1}^{n} a_{ik} z_i$$

Contraintes :

1. $\sum_{i=1}^{n} a_{ik} = k$
2. $a_{ik} \in \{0, 1\}, \quad \forall i \in 1, \ldots, n$

On veut montrer que $L_k(z)$ est la valeur optimale de ce programme linéaire.
Intuitivement, pour minimiser $\sum_{i=1}^{n} a_{ik} z_i$, on veut :

1. Sélectionner exactement k variables (pour respecter les contraints)
2. Choisir les k plus petites valeurs de z_i (on minimise)

Donc trouver la solution optimale revient à faire la somme des k plus petites valeurs de z_i, et c'est ce que représente $L_k(z)$.
Cette formulation permet une linéarisation du critère **maxOWA**, transformant un problème combinatoire complexe en un programme linéaire soluble.

b. Programme linéaire relaxé et dual

Soit le programme linéaire ci-dessus relaxé en variables continues :

Fonction objectif :

$$\textbf{min} \sum_{i=1}^{n} a_{ik} z_i$$

Contraintes :

1. $\begin{cases} \sum_{i=1}^{n} a_{ik} = k \\ a_{ik} \leq 1 \end{cases}$
2. $a_{ik} \geq 0, \quad \forall i \in 1, \ldots, n$

Soit D_k le dual du programme linéaire ci-dessus, pour $k \in \{1, \ldots, n\}$:

- Variable du dual $\boldsymbol{r_k}$: contrainte $\sum_{i=1}^{n} a_{ik} = k$
- Variable du dual $\boldsymbol{b_{ik}}$: contrainte $a_{ik} \leq 1, i = 1, \ldots, n$

$\boldsymbol{D_k}$:

Fonction objectif :

$$\textbf{max k}\boldsymbol{r_k} - \sum_{i=1}^{n} \boldsymbol{b_{ik}}$$

Contraintes :

1. $r_k - b_{ik} \leq z_i$
2. $b_{ik} \geq 0, \quad \forall i \in 1, \ldots, n$

Calcul des composantes du vecteur L(2, 9, 6, 8, 5 , 4) en utilisant la formulation du dual

Soit $\boldsymbol{z} = [\mathbf{2}, \mathbf{9}, \mathbf{6}, \mathbf{8}, \mathbf{5}, \mathbf{4}]$. En triant *z*, on obtient : [2, 4, 5, 6, 8, 9]

- $L_1(z) = 2$
- $L_2(z) = 6$
- $L_3(z) = 11$
- $L_4(z) = 17$
- $L_5(z) = 25$
- $L_6(z) = 34$

Prenons $L_2(z) = 6$ comme exemple. Cette composante correspond à la somme des deux plus petites valeurs de *z*, soit $z_1 = 2$ et $z_6 = 4$.
Dans le dual, r_2 est la variable associée à la contrainte $\sum_{i=1}^{n} a_{i2} = 2$ dans le primal. La valeur de r_2 doit être suffisamment grande pour satisfaire les deux plus petites valeurs de *z* (dans notre cas $z_1 = 2 \; et \; z_6 = 4$) tout en respectant les contraintes. Ici, $r_2 = 4$ est choisi comme valeur optimale.

Calcul des $\boldsymbol{b_{i2}}$

Pour $z_1 = 2$:

$$r_2 - b_{12} \leq z_1 \rightarrow b_{12} \geq r_2 - z_1 = 4 - 2 = 2$$

Pour $z_6 = 4$:

$$r_2 - b_{22} \leq z_2 \rightarrow b_{22} \geq r_2 - z_6 = 4 - 4 = 0$$

Pour i$\in \{2, 3, 4, 5\}$, les valeurs b_{i2} n'affectent pas le résultat car seules les deux plus petites valeurs de *z* sont pertinentes pour $L_2(z)$.

Vérification de la fonction objectif du dual :

La fonction objectif est donnée par :

$$\max 2 \,.\, r_2 \; - \; \sum_{i=1}^{n} b_{i2}$$

en substituant les valeurs $r_2 = 4,\ b_{12} = 2,\ b_{22} = 0$

on obtient : 2 . 4 - (2 + 0) = 8 - 2 = 6

Cette valeur correspond exactement à $L_2(z)$ obtenue dans le primal.

c. OWA sous forme linéaire en termes de $L_k(z(x))$

L'OWA (Ordered Weighted Average) est défini par $g(x) = \sum_{i=1}^{n} w_i z_{(i)}(x)$ où w_i sont des poids positifs et décroissants lorsque i augmente ($z_{(i)}$ a déjà été défini précédemment).

On veut montrer qu'on peut réécrire $g(x) = \sum_{k=1}^{n} w'_k L_k\big(z(x)\big)$ avec $w'_k = w_k$ - w_{k+1} pour $k = 1, \dots, n-1$ et $w'_n = w_n$.

On veut exprimer $z_{(i)}(x)$ en fonction des termes $L_k\big(z(x)\big)$.

Rappel : $z_{(i)}$ représente le i-ème plus petit élément de z

$L_k\big(z(x)\big)$ représente la somme des k premiers $z_{(i)}(x)$

Donc $z_{(i)}(x) = L_i\big(z(x)\big) \; - \; L_{i-1}\big(z(x)\big)$

En substituant $z_{(i)}(x)$ dans $g(x)$, on obtient :

$$g(x) = \sum_{i=1}^{n} w_i \,.\, L_i\big(z(x)\big) \; - \; w_i \,.\, L_{i-1}\big(z(x)\big)$$

$$g(x) = w_1 L_1\big(z(x)\big) - w_1 L_0\big(z(x)\big) \; + \; w_2 L_2\big(z(x)\big) - w_2 L_1\big(z(x)\big) + \dots$$
$$+ w_{n-1} L_{n-1}\big(z(x)\big) - \qquad w_{n-1} L_{n-2}\big(z(x)\big) \; + \; w_n L_n\big(z(x)\big)$$

avec $L_0\big(z(x)\big) = 0$

$$g(x) = \Big[\sum_{i=1}^{n-1} (w_i \; - \; w_{i+1}) \;.L_i\big(z(x)\big)\Big] + w_n L_n\big(z(x)\big)$$

$$g(x) = \sum_{k=1}^{n} w'_k L_k\big(z(x)\big)$$

On obtient le résultat par définition des w'_k et w'_n.

d. Formulation linéaire et application sur notre exemple

En utilisant les résultats des questions précédentes, le problème de l'optimisation d'un OWA peut alors être formulé comme un programme linéaire (appliqué sur notre exemple).

Variables :

- $\boldsymbol{x_j}$: variable binaire, $\boldsymbol{x_j} = \begin{cases} 1 \; si \; le \; projet \; j \; est \; sélectionné \\ 0 \; sinon \end{cases}$
- $\boldsymbol{r_k}$: variable représentant l'utilité totale dans le scénario k
- $\boldsymbol{b_{ik}}$: variable pour modéliser les contraintes de linéarisation

Fonction objectif :

$$\textbf{max} \sum_{k=1}^{n} w'_k \, (\mathbf{k}\, \boldsymbol{r_k} - \sum_{i=1}^{n} \boldsymbol{b_{ik}})$$

Contraintes :

1. $\sum_{j \in P} c_j x_j \leq B$
2. $r_k - b_{ik} \leq z_i(x), \; i \in 1, \ldots, n$
4. $b_{ik} \geq 0, \; x_j \in \{0,1\}, \; \forall j \in P, \; \forall i \in 1, \ldots, n$

Cette modélisation a été implémentée dans un fichier Python (maxOWA.py), utilisant le solveur Gurobi. Une solution optimale x^*, maximisant OWA, a été déterminée. La solution obtenue est la suivante :

x^* = (0, 1, 1, 1, 0, 0, 1, 1, 1, 0)
r_k = (66, 66)
w'_k = (1, 1)
b_{ik} = ((0, 0), (0, 0))
La valeur de la fonction objective (maxOWA) : 198
1 x (1 x 66 – 0) + 1 x (2 x 66 – 0) = 198

e. Formulation linéaire et application de minOWA des regrets

minOWA des regrets cherche à minimiser le max des sommes des regrets critère OWA

Pour un vecteur de regrets r, définissons $L_k(r)$ comme la somme des k plus grands regrets après tri.

$$L_k(r) = \sum_{i=1}^{k} r(x, s_{(i)})$$

où $r(x, s_{(i)})$ représente le i-ème plus grand élément du vecteur de regrets.

Considérant le programme linaire suivant :

Fonction objectif :

$$\textbf{max } \sum_{i=1}^{n} \boldsymbol{a_{ik}} r(x, s_i)$$

Contraintes :

1. $\sum_{i=1}^{n} a_{ik} = k$
2. $a_{ik} \in \{0, 1\}, \quad \forall i \in 1, \dots, n$

$L_k(r)$ est la valeur optimale de ce programme, car on cherche à maximiser la somme des regrets avec minOWA des regrets (même preuve que **II. a.**)

Soit le programme linéaire ci-dessus relaxé en variables continues :

Fonction objectif :

$$\textbf{max } \sum_{i=1}^{n} \boldsymbol{a_{ik} r_i}$$

Contraintes :

1. $\begin{cases} \sum_{i=1}^{n} a_{ik} = k \\ a_{ik} \leq 1 \end{cases}$
2. $a_{ik} \geq 0, \quad \forall i \in 1, \dots, n$

Le dual du programme linéaire ci-dessus, pour $k \in \{1, \dots, n\}$:

- Variable du dual $\boldsymbol{r_k}$: contrainte $\sum_{i=1}^{n} a_{ik} = k$
- Variable du dual $\boldsymbol{b_{ik}}$: contrainte $a_{ik} \leq 1, i = 1, \dots, n$

$\boldsymbol{D_k}$:

Fonction objectif :

$$\textbf{min k} \boldsymbol{r_k} + \sum_{i=1}^{n} \boldsymbol{b_{ik}}$$

Contraintes :

1. $r_k - b_{ik} \geq r_i$
2. $b_{ik} \geq 0, \quad \forall i \in 1, \dots, n$

(même processus que **II. b.)**

En effet, la fonction $g(x) = \sum_{i=1}^{n} w_i r\left(x, s_{(i)}\right)$ peut s'écrire :

$$g(x) = \sum_{k=1}^{n} w'_k L_k\left(r(x)\right)$$

par définition de w'_k (même preuve que **II. c.**)

A partir de ce qui précède, on a une formule linéaire finale du problème minOWA des regrets avec des contraintes adaptés à notre exemple.

Variables :

- $\boldsymbol{x_j}$: variable binaire, $\boldsymbol{x_j} = \begin{cases} 1 \; si \; le \; projet \; j \; est \; sélectionné \\ 0 \; sinon \end{cases}$
- $\boldsymbol{r_k}$: variable représentant le regret du scénario k
- $\boldsymbol{b_{ik}}$: variable pour modéliser les contraintes de linéarisation

Fonction objectif :

$$\textbf{min} \sum_{k=1}^{n} w'_k \left(\mathbf{k}\, \boldsymbol{r_k} + \sum_{i=1}^{n} \boldsymbol{b_{ik}}\right)$$

Contraintes :

1. $\sum_{j \in P} c_j x_j \leq B$
2. $r_k - b_{ik} \geq r_i(x), \; i \in 1, \ldots, n$
3. $b_{ik} \geq 0, \; x_j \in \{0,1\}, \; \forall j \in P, \; \forall i \in 1, \ldots, n$

Cette modélisation a été implémentée dans un fichier Python (minOWA.py), utilisant le solveur Gurobi. Une solution optimale x^*, minimisant OWA des regrets, a été déterminée. La solution obtenue est la suivante :

x^* = (0, 1, 1, 0, 0, 1, 1, 1, 1, 0)
r_k = (50, 50) et r_i =(50, 48)
w'_k = (1, 1)
b_{ik} = ((0, 0), (0, 0))
La valeur de la fonction objective (minOWA) : 150
1 x (1 x 50 – 0) + 1 x (2 x 50 – 0) = 50

f. Généralisation du problème

L'étude des temps de résolution pour le problème de sac à dos robuste a été réalisée en considérant deux critères d'optimisation : maxOWA et minOWA des regrets. La génération des valeurs de n (nombre de scénarios), p (nombre de projets/objets), des utilités, des coûts et des budgets a été effectuée comme précédemment (voir I.d.). Pour les poids, nous avons généré n poids distincts, chacun ayant une valeur comprise entre 1

et n inclus. Les temps moyens de résolution (en secondes) obtenus pour chaque configuration sont présentés dans le tableau ci-dessous.

	n	**p**	Temps moyen de résolution (**maxOWA**)	Temps moyen de résolution (**minOWA regret**)
1	5	10	0.005992	0.00885
2	5	15	0.005794	0.010240
3	5	20	0.007257	0.012443
4	10	10	0.018399	0.016411
5	10	15	0.012841	0.025568
6	10	20	0.014663	0.032899
7	15	10	0.018421	0.034338
8	15	15	0.020994	0.047872
9	15	20	0.031896	0.059813

Les temps de résolution pour minOWA des regret sont significativement plus élevés que ceux pour maxOWA, indiquant une complexité supplémentaire liée aux calculs de regret. Lorsque *p* (le nombre de projets) augmente. L'impact de *n* (le nombre de scénarios) est plus marqué, car chaque scénario ajoute des variables de regret r_k et des contraintes supplémentaires pour garantir la cohérence des regrets et des linéarisations b_{ik}, augmentant ainsi le nombre total de calculs nécessaires pour résoudre le problème. En particulier, le nombre de contraintes et de variables b_{ik} croît proportionnellement à n^2, ce qui peut entraîner une augmentation exponentielle des temps de résolution lorsque *n* devient grand.

III. Application à la recherche d'un chemin robuste dans un graphe

Dans un réseau modélisé par un graphe orienté, les temps de transport sur les arcs varient selon un ensemble de scénarios S = {1, …, n}. Chaque arc (i, j) est associé à un vecteur $\left(t_{ij}^1, \ldots, t_{ij}^n\right)$ représentant les temps dans chaque scénario. Les temps étant additifs le long des chemins, on peut calculer facilement la durée totale d'un trajet pour un scénario donné. Par la suite ***P*** représenté un chemin, et pour s ∈ S : $t^s(P) = \sum_{(i,j)\in P} t_{ij}^s$.

a. Chemin plus rapide

Nous cherchons à modéliser le chemin le plus rapide dans un graphe orienté sous un scénario donné s ∈ S, où chaque arc (i, j) a un coût en temps spécifique t_{ij}^s. Le problème peut être reformulé comme un problème de flot maximum à coût minimum.

Variables :

- $\boldsymbol{x_{ij}}$: variable binaire, $\boldsymbol{x_j} = \begin{cases} 1\ si\ l'arc\ (i,j)\ fait\ partie\ du\ chemin\ choisi \\ 0\ sinon \end{cases}$

Fonction objectif :

$$\textbf{max } \boldsymbol{z = \sum_{(i,j)\in G} t_{ij}^s \, . \, x_{ij}}$$

Contraintes :

1. $\boldsymbol{\sum_{j\in succ(i)} x_{ij} - \sum_{k\in pred(i)} x_{ki}} =$
$$\begin{cases} 1\ si\ i\ départ\ (pas\ d'arc\ entrant) \\ -1\ si\ i\ destination\ (pas\ d'arc\ sortant) \\ 0\ sinon\ (conservation\ :\ si\ on\ entre, on\ sort) \end{cases}$$
2. $\boldsymbol{x_{ij}} \in \{0,1\},\ \forall (i,j) \in G$

Cette modélisation a été implémentée dans un fichier Python (cheminPlusRapide.py), utilisant le solveur Gurobi.

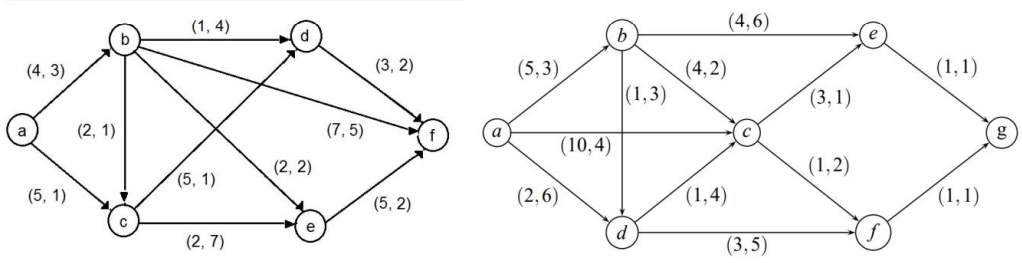


FIGURE 1 – Deux instance du problème de plus court chemin robuste à 2 scénarios

Pour chaque instance de l'exemple, nous avons trouvé par programmation linéaire (les chemins les plus rapides dans chacun des deux scenarios :

Instance 1, scenario 1 : [(a, b), (b, d), (d, f)], temps = 8
Instance 1, scenario 2 : [(a, c), (c, d), (d, f)], temps = 4
Instance 2, scenario 1 : [(a, d), (d, c), (c, f), (f, g)], temps = 5
Instance 2, scenario 2 : [(a, c), (c, e), (e, g)], temps = 6

b. Formulation et application

Dans cette sous-partie, nous proposons des programmes linéaires adaptés à la résolution du problème de chemin robuste, en fonction des quatre fonctions considérées précédemment. Ces programmes seront appliqués aux graphes de la figure 1 pour déterminer un chemin robuste selon les différents scénarios, en utilisant un vecteur de pondération w = (*k*, 1) pour *k*= 2, 4, 8, 16.

- ***g(x)* : maxmin**

Variables :

- x_{ij} : variable binaire, $x_j = \begin{cases} 1 \text{ si } l'arc\ (i,j)\ fait\ partie\ du\ chemin\ choisi \\ 0\ sinon \end{cases}$
- z : variable continue représentant le minimum des $-t^s(P) = -\sum_{(i,j)\in P} t_{ij}^s$

Fonction objectif :

$$\textbf{max } z$$

Contraintes :

1. $z \leq -t^s(P) = -\sum_{(i,j)\in P} t_{ij}^s \qquad \forall\, s \in S$
2. $\sum_{j\in succ(i)} x_{ij} - \sum_{k\in pred(i)} x_{ki} \begin{cases} 1\ si\ i\ départ \\ -1\ si\ i\ destination \\ 0\ sinon \end{cases}$
3. $x_{ij} \in \{0,1\},\ \forall (i,j) \in P$

Instance 1 (maxmin) : [(a, b), (b, d), (d, f)]
Temps maximal sur le chemin : 9
Instance 2 (maxmin) : [(a, b), (b, e), (e, g)]
Temps maximal sur le chemin : 10

- ***g(x)* : mimax Regret**

Variables :

- x_{ij} : variable binaire, $x_j = \begin{cases} 1 \; si \; l'arc \; (i,j) \; fait \; partie \; du \; chemin \; choisi \\ 0 \; sinon \end{cases}$
- z : variable continue représentant le maximum des $r^s(P) = t^s(P^*) - t^s(P)$ *avec P^* le chemin optimal dans le scenario s.*

Fonction objectif :

$$\textbf{min } z$$

Contraintes :

1. $z \geq t^s(P^*) - t^s(P) \qquad \forall \, s \in S$
2. $\sum_{j \in succ(i)} x_{ij} \; - \sum_{k \in pred(i)} x_{ki} \begin{cases} 1 \; si \; i \; départ \\ -1 \; si \; i \; destination \\ 0 \; sinon \end{cases}$
3. $x_{ij} \in \{0,1\}, \; \forall (i,j) \in P$

Instance 1 (minmax Regret) : [(a, b), (b, e), (e, f)]
Regret maximal sur le chemin : 3
Instance 2 (minmax Regret): [(a, b), (b, e), (e, g)]
Regret maximal sur le chemin : 5

- ***g(x)* : maxOWA**

Variables :

- x_{ij} : variable binaire, $x_j = \begin{cases} 1 \; si \; l'arc \; (i,j) \; fait \; partie \; du \; chemin \; choisi \\ 0 \; sinon \end{cases}$
- r_k : variable représentant le temps total dans le scénario k
- b_{sk} : variable pour modéliser les contraintes de linéarisation

Fonction objectif :

$$\textbf{max} \sum_{k=1}^{n} w'_k \, (\mathbf{k} \, r_k \; - \sum_{i=1}^{n} b_{sk})$$

Contraintes :

1. $r_k \; - \; b_{sk} \; \leq \; -t^s(P) \qquad \forall \, s \in S$
2. $\sum_{j \in succ(i)} x_{ij} \; - \sum_{k \in pred(i)} x_{ki} \begin{cases} 1 \; si \; i \; départ \\ -1 \; si \; i \; destination \\ 0 \; sinon \end{cases}$
3. $b_{sk} \geq 0, \; x_{ij} \in \{0,1\}, \; \forall (i,j) \in P, \forall \, s \in S$

Instance 1 :
Résultats pour k = 4:
Chemin sélectionné: [('a', 'b'), ('b', 'c'), ('c', 'e'), ('e', 'f')]
Valeur de la fonction objectif: 65.0
Valeurs des rk: [13.0, 13.0]

Instance 2 :
Résultats pour k = 8:
Chemin sélectionné: [('a', 'b'), ('b', 'd'), ('c', 'e'), ('d', 'c'), ('e', 'g')]
Valeur de la fonction objectif: 100.0
Valeurs des rk: [11.0, 12.0]

- ***g(x)* : minOWA des regrets**

Variables :

- $\boldsymbol{x_{ij}}$: variable binaire, $\boldsymbol{x_j} = \begin{cases} 1\ si\ l'arc\ (i,j)\ fait\ partie\ du\ chemin\ choisi \\ 0\ sinon \end{cases}$
- $\boldsymbol{r_k}$: variable représentant le regret du scénario k
- $\boldsymbol{b_{sk}}$: variable pour modéliser les contraintes de linéarisation

Fonction objectif :

$$\textbf{min} \sum_{k=1}^{n} w'_k \left(\mathbf{k}\, \boldsymbol{r_k} + \sum_{i=1}^{n} \boldsymbol{b_{sk}}\right)$$

Contraintes :

1. $r_k - b_{sk} \geq r^s(P) = t^s(P^*) - t^s(P) \quad \forall\, s \in S$
2. $\sum_{j \in succ(i)} \boldsymbol{x_{ij}} - \sum_{k \in pred(i)} \boldsymbol{x_{ki}} \begin{cases} 1\ si\ i\ départ \\ -1\ si\ i\ destination \\ 0\ sinon \end{cases}$
3. $b_{sk} \geq 0,\ \boldsymbol{x_{ij}} \in \{0,1\},\ \forall (i,j) \in P, \forall\, s \in S$

Ces 4 modélisations ont été implémentée dans un fichier Python (cheminRobuste.py), utilisant le solveur Gurobi.

En conclusion, ce projet a mis en lumière l'importance de l'optimisation robuste dans des contextes marqués par une incertitude totale, où les scénarios peuvent varier de manière imprévisible. Contrairement aux approches traditionnelles qui supposent une connaissance précise des conditions, l'optimisation robuste cherche à identifier des solutions fiables, capables de maintenir une performance satisfaisante dans les situations les plus défavorables.

Nous avons exploré plusieurs concepts clés de l'optimisation robuste, notamment :

1. Les critères d'évaluation comme maxmin, minmax regret, maxOWA, et minOWA des regrets.
2. La linéarisation des problèmes non linéaires, permettant leur résolution via des outils de programmation linéaire tels que Gurobi.
3. L'application des méthodes sur des cas pratiques, comme le problème du sac à dos et la recherche de chemins robustes dans des graphes.

Les méthodes étudiées ont démontré leur capacité à répondre à des objectifs spécifiques, tels que minimiser le pire cas ou limiter le regret maximal. La résolution grâce à Gurobi a permis non seulement de résoudre efficacement les modèles mais aussi d'illustrer leur application sur des exemples concrets, tels que le problème du sac à dos robuste ou la recherche de chemins robustes dans des graphes.