Projet MOGPL

Jules MAZLUM
Zahra SIDDIQUE

Novembre - Décembre 2024

Contents

1 1/ Linéarisation des critères maxmin et minmax regret

1.1 maxmin

D'après le sujet maxmin consiste à faire :

$$\max g(x)$$

$$\max \min_{i=1}^{n} z_i(x)$$

Pour l'exemple 1 :

$$\max \min_{i=1}^{2} z_i(x)$$

$$\max \min\{z_1(x), z_2(x)\}$$

Pour linéariser ce programme on introduit la variable z tel que $z = \min\{z_1(x), z_2(x)\}$
Et on introduit également la contrainte sur le budget.

$$\max z$$

$$z \leq z_1(x) = 70x_1 + 18x_2 + 16x_3 + 14x_4 + 12x_5 + 10x_6 + 8x_7 + 6x_8 + 4x_9 + 2x_{10}$$

$$z \leq z_2(x) = 2x_1 + 4x_2 + 6x_3 + 8x_4 + 10x_5 + 12x_6 + 14x_7 + 16x_8 + 18x_9 + 70x_{10}$$

$$60x_1 + 10x_2 + 15x_3 + 20x_4 + 25x_5 + 20x_6 + 5x_7 + 15x_8 + 20x_9 + 60x_{10} \leq 100$$

$$x_i \geq 0, \quad i \in \{1, 2, \ldots, 10\}, \quad \text{et} \quad z \geq 0$$

On peut réécrire notre programme linéaire tel que :

$$\max z$$

$$-70x_1 - 18x_2 - 16x_3 - 14x_4 - 12x_5 - 10x_6 - 8x_7 - 6x_8 - 4x_9 - 2x_{10} + z \leq 0$$

$$-2x_1 - 4x_2 - 6x_3 - 8x_4 - 10x_5 - 12x_6 - 14x_7 - 16x_8 - 18x_9 - 70x_{10} + z \leq 0$$

$$60x_1 + 10x_2 + 15x_3 + 20x_4 + 25x_5 + 20x_6 + 5x_7 + 15x_8 + 20x_9 + 60x_{10} \leq 100$$

$$x_i \geq 0, \quad i \in \{1, 2, \ldots, 10\}, \quad \text{et} \quad z \geq 0$$

Grâce au solveur on obtient la solution :

$$x^* = (0, 1, 1, 1, 0, 0, 1, 1, 1, 0) \quad \text{et} \quad z_1(x^*) = 66, \quad z_2(x^*) = 66$$

Donc :

$$z(x^*) = (66, 66)$$

Vous pouvez exécuter le script **maxmin.py** pour retrouver cette solution.

1.2 minmax regret

D'après le sujet minmax regret consiste à faire :

$$\min g(x)$$
$$\min \max_{i=1}^{n} r(x, s_i)$$
$$\min \max_{i=1}^{n} z_i^* - z_i(x)$$

Sachant que $r(x, s_i)$ est le regret d'avoir choisit x pour s_i et z_i^* est la solution optimale pour s_i.

Pour l'exemple 1 :

$$\min \max_{i=1}^{2} z_i^* - z_i(x)$$
$$\min \max\{z_1^* - z_1(x), z_2^* - z_2(x)\}$$

Il faut donc dans un premier temps calculer z_1^* et z_2^*

Pour z_1^* :

$$\max z_1(x)$$
$$\max 70x_1 + 18x_2 + 16x_3 + 14x_4 + 12x_5 + 10x_6 + 8x_7 + 6x_8 + 4x_9 + 2x_{10}$$
$$60x_1 + 10x_2 + 15x_3 + 20x_4 + 25x_5 + 20x_6 + 5x_7 + 15x_8 + 20x_9 + 60x_{10} \leq 100$$
$$x_i \geq 0, \quad i \in \{1, 2, \ldots, 10\}$$

Grâce au solveur :

$$x_1^* = (1, 1, 1, 0, 0, 0, 1, 0, 0, 0) \quad \text{et} \quad z_1(x_1^*) = 112, \quad z_2(x_1^*) = 26$$

Donc :

$$z(x_1^*) = (112, 26) \quad \text{et} \quad z_1^* = 112$$

Vous pouvez exécuter le script **s1_optimal.py** pour retrouver cette solution.

Pour z_2^* :

$$\max z_2(x)$$
$$\max 2x_1 + 4x_2 + 6x_3 + 8x_4 + 10x_5 + 12x_6 + 14x_7 + 16x_8 + 18x_9 + 70x_{10}$$
$$60x_1 + 10x_2 + 15x_3 + 20x_4 + 25x_5 + 20x_6 + 5x_7 + 15x_8 + 20x_9 + 60x_{10} \leq 100$$
$$x_i \geq 0, \quad i \in \{1, 2, \ldots, 10\}$$

Grâce au solveur :

$$x_2^* = (0, 0, 0, 0, 0, 0, 1, 1, 1, 1) \quad \text{et} \quad z_1(x_2^*) = 20, \quad z_2(x_2^*) = 118$$

Donc :

$$z(x_2^*) = (20, 118) \quad \text{et} \quad z_2^* = 118$$

Vous pouvez exécuter le script **s2_optimal.py** pour retrouver cette solution.
On a maintenant $z_1^* = 112$ et $z_2^* = 118$

On peut revenir sur :

$$\min \max\{z_1^* - z_1(x), z_2^* - z_2(x)\}$$

Pour linéariser ce programme on introduit la variable z tel que $z = \max\{z_1^* - z_1(x), z_2^* - z_2(x)\}$
Et on introduit également la contrainte sur le budget.

$$\min z$$
$$z \geq z_1^* - z_1(x) = 112 - (70x_1 + 18x_2 + 16x_3 + 14x_4 + 12x_5 + 10x_6 + 8x_7 + 6x_8 + 4x_9 + 2x_{10})$$
$$z \geq z_2^* - z_2(x) = 118 - (2x_1 + 4x_2 + 6x_3 + 8x_4 + 10x_5 + 12x_6 + 14x_7 + 16x_8 + 18x_9 + 70x_{10})$$
$$60x_1 + 10x_2 + 15x_3 + 20x_4 + 25x_5 + 20x_6 + 5x_7 + 15x_8 + 20x_9 + 60x_{10} \leq 100$$
$$x_i \geq 0, \quad i \in \{1, 2, \ldots, 10\}, \quad \text{et} \quad z \geq 0$$

On peut simplifier nos expressions tel que :

$$\min z$$
$$z \geq 112 - 70x_1 - 18x_2 - 16x_3 - 14x_4 - 12x_5 - 10x_6 - 8x_7 - 6x_8 - 4x_9 - 2x_{10}$$
$$z \geq 118 - 2x_1 - 4x_2 - 6x_3 - 8x_4 - 10x_5 - 12x_6 - 14x_7 - 16x_8 - 18x_9 - 70x_{10}$$
$$60x_1 + 10x_2 + 15x_3 + 20x_4 + 25x_5 + 20x_6 + 5x_7 + 15x_8 + 20x_9 + 60x_{10} \leq 100$$
$$x_i \geq 0, \quad i \in \{1, 2, \ldots, 10\}, \quad \text{et} \quad z \geq 0$$

On peut réécrire notre programme linéaire tel que :

$$\min z$$
$$70x_1 + 18x_2 + 16x_3 + 14x_4 + 12x_5 + 10x_6 + 8x_7 + 6x_8 + 4x_9 + 2x_{10} + z \geq 112$$
$$2x_1 + 4x_2 + 6x_3 + 8x_4 + 10x_5 + 12x_6 + 14x_7 + 16x_8 + 18x_9 + 70x_{10} + z \geq 118$$
$$-60x_1 - 10x_2 - 15x_3 - 20x_4 - 25x_5 - 20x_6 - 5x_7 - 15x_8 - 20x_9 - 60x_{10} \geq -100$$
$$x_i \geq 0, \quad i \in \{1, 2, \ldots, 10\}, \quad \text{et} \quad z \geq 0$$

Grâce au solveur :

$$x^* = (0, 1, 0, 1, 1, 0, 1, 1, 1, 0) \quad \text{et} \quad z_1(x'^*) = 62, \quad z_2(x'^*) = 70$$

Donc :

$$z(x'^*) = (62, 70)$$

Vous pouvez exécuter le script **minmax_regret.py** pour retrouver cette solution.

1.3 Représentation graphique

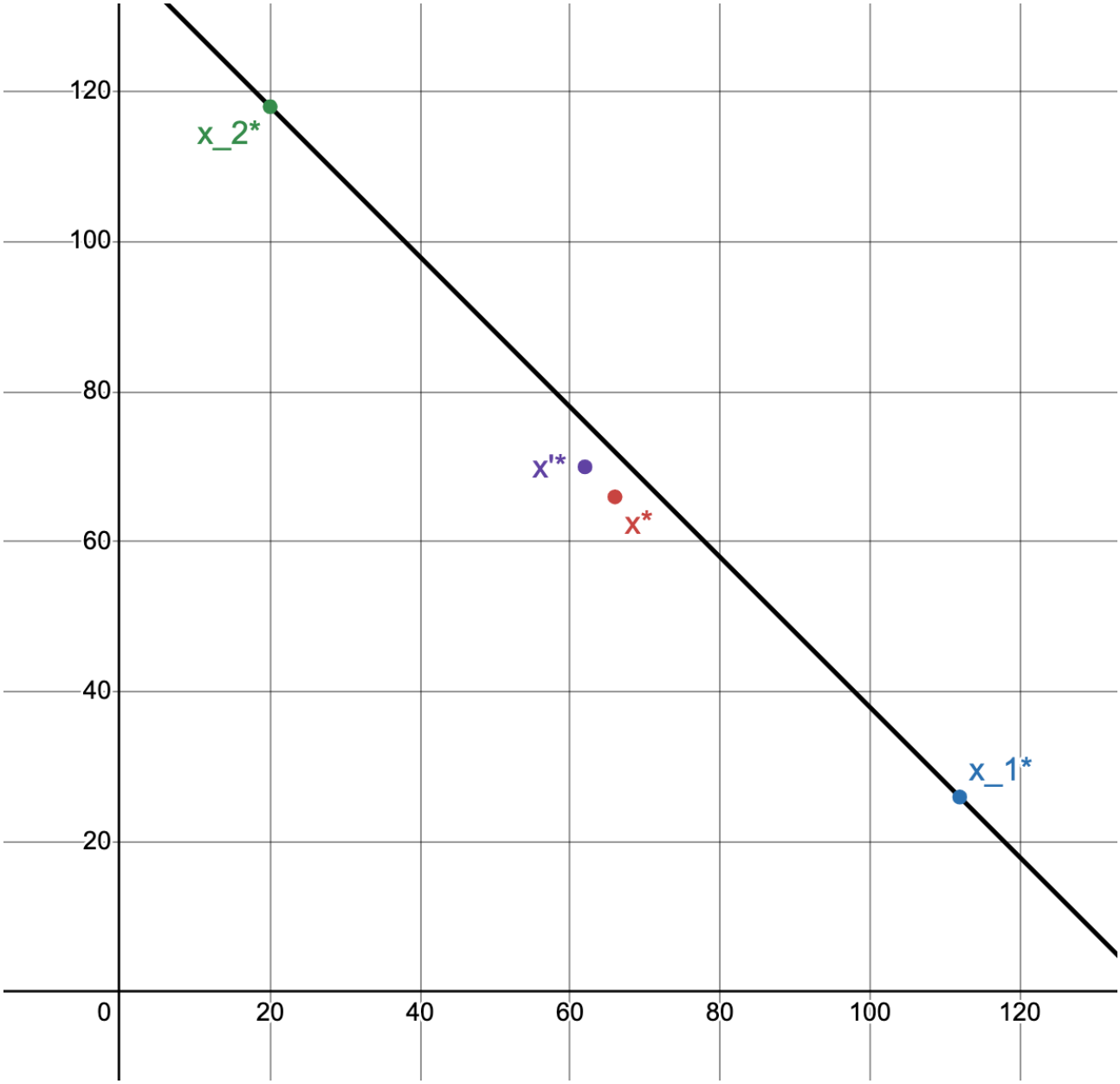


Figure 1: Les 4 points $z(x_1^*)$, $z(x_2^*)$, $z(x*)$ et $z(x'^*)$, dans le plan $z_1(x)$,$z_2(x)$

Les solutions x^* et x'^* n'aurait pas pu être trouvée en maximisant une moyenne pondérée à coefficients positifs entre $z_1(x)$ et $z_2(x)$.
Cela reviendrait à maximiser une combinaison linéaire tel que $w_1 \times z_1(x) + w_2 \times z_2(x)$ avec $w_1, w_2 > 0$.

Or, avec maxmin on maximise le pire score ce qui est non linéaire et avec minmax regret on minimise le regret maximum ce qui n'est pas linéaire non plus.

Sur le graphe, on peut vérifier cela en traçant une droite de $z(x_1^*)$ à $z(x_2^*)$ et regarder si nos solutions sont sur cette droite.
Si elles le sont, alors elles peuvent être écrites comme une combinaison linéaire de $z(x_1^*)$ à $z(x_2^*)$ et donc on pourrait les retrouver en maximisant une moyenne pondéré.
Mais comme on peut le voir sur le graphe, les points $z(x*)$, $z(x'^*)$ ne sont pas sur la droite, ce qui signifie que les solutions x^* et x'^* ne sont pas des combinaison linéaire de x_1^* et x_2^*.

Finalement, on aurait pas pu les trouver en faisant simplement une maximisation sur la moyenne pondérée.

1.4 Évolution du temps de résolution en fonction de n et p

Dans cette section, nous cherchons à généraliser l'exemple 1 en considérant p projets et n scénarios. L'objectif est d'analyser l'évolution du temps de résolution en fonction de n et p. Plus précisément, pour n = 5, 10, 15 et p = 10, 15, 20, nous générerons 10 instances du problème pour chaque couple (n, p). Le budget sera fixé à 50% du coût total des p projets, et les coûts et utilités seront tirés aléatoirement dans l'intervalle [0, 100].

Et on procède ainsi pour les 2 méthodes : maxmin et minmax regret.

On remarque que résoudre le PL maxmin revient à avoir les paramètres :
Matrice des contraintes :

$$a = \begin{bmatrix} \text{les utilités dans [-100; 0]} & \dots & 1 \\ \dots & \dots & 1 \\ \text{les couts dans [0;100]} & \dots & 0 \end{bmatrix}$$

Second membre :

$$b = \begin{bmatrix} 0 \\ 0 \\ budget \end{bmatrix}$$

Fonction objectif :

$$c = \begin{bmatrix} 0 & \dots & 1 \end{bmatrix}$$

On a alors écrit la fonction **generator_maxmin(n,p)** qui prend en paramètre le nombre de scénario n et projets p et retourne les 3 matrices a, b et c correspondant à notre PL. En ayant généré aléatoirement les coûts et utilités tout en respectant nos conditions.
Vous pouvez consulter son fonctionnement et le tester dans le script **generator.py**.

Pour minmax regret on remarque également que résoudre le PL revient à avoir les paramètres :
Matrice des contraintes :

$$a = \begin{bmatrix} \text{les utilités dans [0; 100]} & \dots & 1 \\ \dots & \dots & 1 \\ \text{les couts dans [-100;0]} & \dots & 0 \end{bmatrix}$$

Second membre :

$$b = \begin{bmatrix} \text{la solution optimale pour la première ligne de a} \\ \dots \\ -budget \end{bmatrix}$$

Fonction objectif :

$$c = \begin{bmatrix} 0 & \dots & 1 \end{bmatrix}$$

De même on a écrit la fonction **generator_minmaxregret(n,p)** que vous pouvez également consulter et tester dans les script **generator.py**.

Maintenant qu'on peut générer autant de PL qu'on veut.
On peut calcluer le temps d'éxecution en fonction de n et p.
Pour cela vous pouvez consulter le script **main.py**

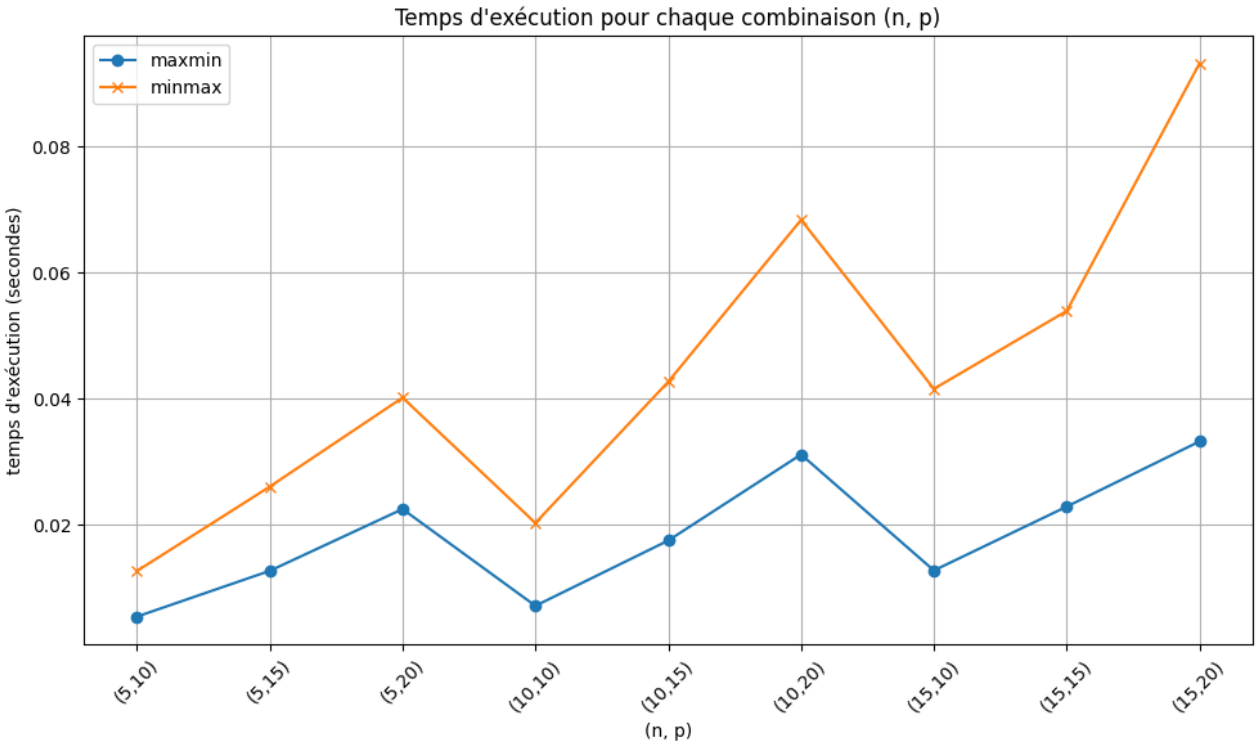


Figure 2: Temps d'execution en fonction de n et p

On peut tout d'abord voir que minmax regret prend beaucoup plus de temps comparé à maxmin. Ce qui peut être expliqué par le fait que beaucoup plus de PL doivent être résolu dans le cas du minmax regret. En effet on se rappelle que pour résoudre un minmax regret on doit dans un premier temps résoudre des PL pour chaque scénario pour trouver la solution optimale, pour ensuite en déduire le regret.
Quand on observe les points indépendamment, on voit que p donc le nombre de projets impacte le temps beaucoup plus que n le nombre de scénario.
On a une augmentation plus forte quand n est constant et p augmente que quand p est constant et n augmente.

2 Linéarisation du critère maxOWA

Rappel : Le critère **maxOWA** (**Ordered Weighted Average**) consiste à maximiser une moyenne pondérée des performances de la solution x dans les différents scénarios, en accordant plus d'importance aux pires performances.

2.1 Justification de la solution optimale du programme linéaire en variables 0-1

Pour sélectionner les k plus petites valeurs d'un vecteur z, nous utilisons un programme linéaire en variables binaires. Voici pourquoi cette formulation garantit que la solution optimale correspond à la somme des k plus petites valeurs, notée $L_k(z)$.

Définition de $L_k(z)$

$L_k(z)$ est défini comme la somme des k plus petites valeurs du vecteur z trié par ordre croissant :

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

où $z_{(i)}$ représente la $i^{\text{ème}}$ plus petite valeur de z.

Formulation du programme linéaire en variables binaires

Le programme linéaire est formulé comme suit :

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

Sous les contraintes :

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

Interprétation des contraintes

- **Contrainte $\sum_{i=1}^{n} a_{ik} = k$:** Cette contrainte impose que exactement k variables a_{ik} prennent la valeur 1. Cela signifie que l'objectif consiste à sélectionner précisément k valeurs parmi les n composantes de z.
- **Variables binaires $a_{ik} \in \{0,1\}$:** Si $a_{ik} = 1$, la valeur z_i est incluse dans la somme. Si $a_{ik} = 0$, z_i n'est pas incluse.

Pourquoi cette formulation donne les k plus petites valeurs ?

- **Minimisation de la somme :** L'objectif est de minimiser la somme $\sum_{i=1}^{n} a_{ik} z_i$. Puisque chaque a_{ik} vaut soit 0, soit 1, seules les valeurs z_i correspondant aux k plus petites valeurs seront choisies pour minimiser cette somme.
- **Importance de l'ordre croissant :** Les k premières valeurs de z trié par ordre croissant sont les plus petites. Si une valeur plus grande était choisie à la place de l'une des k plus petites, la somme totale augmenterait, ce qui irait à l'encontre de l'objectif de minimisation.
- **Optimalité par contradiction :** Supposons que la solution optimale ne sélectionne pas les k plus petites valeurs. Cela signifie qu'au moins une des valeurs sélectionnées est plus grande qu'une valeur non sélectionnée. En remplaçant cette valeur plus grande par une valeur plus petite non sélectionnée, la somme totale diminuerait, contredisant ainsi l'hypothèse d'optimalité.

Afin d'illustrer : Soit $z = (15, 5, 10, 20)$ et $k = 2$.

- **Tri des valeurs de z :** z trié par ordre croissant donne : $(5, 10, 15, 20)$.
- **Objectif :** Minimiser $\sum a_{i2} z_i$ sous la contrainte $\sum a_{i2} = 2$.
- **Combinaisons possibles :**

- $(a_{12}, a_{22}, a_{32}, a_{42}) = (1, 1, 0, 0)$ sélectionne 5 et 10 : $5 + 10 = 15$ (somme minimale).
- Autres combinaisons :
  - $(1, 0, 1, 0)$ sélectionne 5 et 15 : $5 + 15 = 20$.
  - $(0, 1, 1, 0)$ sélectionne 10 et 15 : $10 + 15 = 25$.

Ainsi, solution optimale sélectionne les k plus petites valeurs de z ($L_k(z)$) parce que :

- La contrainte $\sum a_{ik} = k$ force la sélection de k valeurs.
- La minimisation de $\sum a_{ik} z_i$ garantit que seules les valeurs les plus petites seront choisies.
- Toute autre combinaison incluant des valeurs plus grandes donnerait une somme plus élevée, ce qui contredirait l'optimalité.

2.2 Dual D_k du programme linéaire relaxées en variables continues

On admet que le programme mathématique ci-dessous peut être relaxé en variables continues (les solutions optimales du problème relaxé sont naturellement entières). Ainsi, $L_k(z)$ est aussi la valeur à l'optimum du programme linéaire suivant :

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

Sous les contraintes :

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

Relaxation continue

Les variables binaires a_{ik} (qui étaient initialement égales à 0 ou 1) peuvent être remplacées par des variables continues $a_{ik} \in [0, 1]$. À l'optimum, les solutions de ce programme relaxé sont naturellement entières, ce qui signifie que les valeurs a_{ik} prennent soit 0, soit 1.

Programme primal

Le programme primal est donné par :

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

Sous les contraintes :

$$\sum_{i=1}^{n} a_{ik} = k, \quad 0 \leq a_{ik} \leq 1$$

Variables duales

Pour chaque contrainte du primal, nous associons des variables duales :

- r_k : Variable duale associée à la contrainte $\sum_{i=1}^{n} a_{ik} = k$.
- b_{ik} : Variables duales associées aux contraintes $a_{ik} \leq 1$.

Formulation du dual D_k

Le dual du programme primal s'écrit :

$$\max \left(r_k k - \sum_{i=1}^{n} b_{ik} \right)$$

Sous les contraintes :

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

$$b_{ik} \geq 0$$

Calcul des composantes du vecteur $L(z)$

Pour calculer les composantes du vecteur $L(z)$ à partir du dual, nous résolvons successivement les duals D_k pour chaque $k \in \{1, \ldots, n\}$.

Exemple : Calcul des composantes de $L(z)$ pour $z = (2, 9, 6, 8, 5, 4)$

Les composantes $L_k(z)$ correspondent à la somme des k plus petites valeurs de z .

$$L_1(z) = 2$$

$$L_2(z) = 2 + 4 = 6$$

$$L_3(z) = 2 + 4 + 5 = 11$$

$$L_4(z) = 2 + 4 + 5 + 6 = 17$$

$$L_5(z) = 2 + 4 + 5 + 6 + 8 = 25$$

$$L_6(z) = 2 + 4 + 5 + 6 + 8 + 9 = 34$$

2.3 Réécriture de l'OWA en fonction des $L_k(z(x))$

Nous allons montrer que l'OWA (Ordered Weighted Average) défini par l'équation suivante :

$$g(x) = \sum_{i=1}^{n} w_i z_{(i)}(x)$$

peut être réécrit comme :

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

où w'_k est défini par :

$$w'_k = w_k - w_{k+1} \quad \text{pour} \quad k = 1, \ldots, n-1 \quad \text{et} \quad w'_n = w_n$$

Nous montrerons également que toutes les composantes de w' sont positives ou nulles, étant donné que w est un vecteur de poids décroissants.

Rappel des définitions

- $z(x) = (z_1(x), z_2(x), \ldots, z_n(x))$ représente les performances de la solution x dans n scénarios.
- $z_{(i)}(x)$ est la $i^{\text{ème}}$ plus petite valeur de $z(x)$ (c'est-à-dire $z(x)$ trié par ordre croissant).
- $L_k(z(x))$ est la somme des k plus petites valeurs de $z(x)$:

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

Réécriture de $g(x)$

Expression de $\sum_{k=1}^{n} w'_k L_k(z(x))$
Développons l'expression :

$$\sum_{k=1}^{n} w'_k L_k(z(x)) = w'_1 L_1(z(x)) + w'_2 L_2(z(x)) + \cdots + w'_n L_n(z(x))$$

Substituons $L_k(z(x))$ par sa définition :

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

Donc :

$$\sum_{k=1}^{n} w'_k L_k(z(x)) = w'_1 z_{(1)}(x) + w'_2(z_{(1)}(x) + z_{(2)}(x)) + \cdots + w'_n(z_{(1)}(x) + \cdots + z_{(n)}(x))$$

Réorganisation des termes
En développant cette somme, chaque $z_{(i)}(x)$ apparaît plusieurs fois :

$$w'_1 z_{(1)}(x) + w'_2(z_{(1)}(x) + z_{(2)}(x)) + \cdots + w'_n(z_{(1)}(x) + \cdots + z_{(n)}(x))$$

Cela peut être réorganisé en regroupant les coefficients de chaque $z_{(i)}(x)$:

$$= (w'_1 + w'_2 + \cdots + w'_n)z_{(1)}(x) + (w'_2 + w'_3 + \cdots + w'_n)z_{(2)}(x) + \cdots + w'_n z_{(n)}(x)$$

Utilisation de la relation $w'_k = w_k - w_{k+1}$
On sait que :

$$w'_1 + w'_2 + \cdots + w'_n = w_1, \quad w'_2 + w'_3 + \cdots + w'_n = w_2, \quad \ldots, \quad w'_n = w_n$$

Ainsi, l'expression devient :

$$\sum_{k=1}^{n} w'_k L_k(z(x)) = w_1 z_{(1)}(x) + w_2 z_{(2)}(x) + \cdots + w_n z_{(n)}(x) = \sum_{i=1}^{n} w_i z_{(i)}(x)$$

Conclusion

Nous avons montré que l'OWA peut être réécrit sous la forme suivante :

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

avec $w'_k = w_k - w_{k+1}$ pour $k = 1, \ldots, n-1$ et $w'_n = w_n$.

Propriétés des poids w'_k

Les poids w'_k sont positifs ou nuls car w_k est décroissant :

$$w_1 \geq w_2 \geq \cdots \geq w_n \implies w_k - w_{k+1} \geq 0 \quad (\text{pour} \quad k = 1, \ldots, n-1)$$

et $w'_n = w_n \geq 0$.
Ainsi, $w' = (w'_1, w'_2, \ldots, w'_n)$ a toutes ses composantes positives ou nulles.

2.4 Reformulation linéaire de la maximisation de l'OWA

Nous allons reformuler le problème de maximisation du critère OWA en utilisant la formulation linéaire suivante:

$$\max \sum_{k=1}^{n} w'_k \left(k r_k - \sum_{i=1}^{n} b_{ik} \right)$$

Sous les contraintes :

$$r_k - b_{ik} \leq z_i(x), \quad \forall i \in \{1, \ldots, n\}$$
$$x \in X$$
$$b_{ik} \geq 0, \quad \forall i, k \in \{1, \ldots, n\}$$

Cette formulation utilise n^2 variables réelles positives b_{ik} pour $i, k \in \{1, \ldots, n\}$ et n variables réelles non-signées r_k, pour $k \in \{1, \ldots, n\}$.

Application à l'exemple 1 : Formulation du problème OWA linéarisé

Données de l'exemple

- **Nombre de projets** : 10
- **Nombre de scénarios** : 2 (s_1 et s_2)
- **Utilités des projets** :
  - Scénario 1 (u_1) : [70, 18, 16, 14, 12, 10, 8, 6, 4, 2]
  - Scénario 2 (u_2) : [2, 4, 6, 8, 10, 12, 14, 16, 18, 70]
- **Coûts des projets** : [60, 10, 15, 20, 25, 20, 5, 15, 20, 60]
- **Budget total** : 100
- **Poids OWA** : $w_1 = 2$ et $w_2 = 1$

Variables de décision

- $x_i \in \{0, 1\}$: Indique si le projet i est sélectionné ($x_i = 1$) ou non ($x_i = 0$).
- $r_k \in R^+$: Performance associée au scénario k.
- $b_{ik} \in R^+$: Variables auxiliaires pour linéariser les performances.

Fonction objectif

Maximiser la somme pondérée des performances :

$$\max g(x) = w_1' r_1 + w_2' r_2$$

où $w_1' = 1$ et $w_2' = 1$, donc :

$$\max g(x) = r_1 + r_2$$

Contraintes

1. **Contraintes de performance :**

$$r_1 - b_{i1} \leq \sum_{j=1}^{10} u_{1j} x_j \quad \forall i = 1, \ldots, 10$$

$$r_2 - b_{i2} \leq \sum_{j=1}^{10} u_{2j} x_j \quad \forall i = 1, \ldots, 10$$

2. **Contrainte de budget :**

$$\sum_{i=1}^{10} c_i x_i \leq 100$$

3. **Contraintes de non-négativité et binaires :**

$$x_i \in \{0, 1\}, \quad r_k \geq 0, \quad b_{ik} \geq 0 \quad \forall i, k$$

Solution optimale avec Gurobi

Les valeurs des variables de décision sont les suivantes :

$$x_1 = -0.0$$
$$x_2 = 1.0$$
$$x_3 = 1.0$$
$$x_4 = 1.0$$
$$x_5 = -0.0$$
$$x_6 = -0.0$$
$$x_7 = 1.0$$
$$x_8 = 1.0$$
$$x_9 = 1.0$$
$$x_{10} = -0.0$$

Les valeurs des variables continues r_k sont :

$$r_1 = 66.0$$
$$r_2 = 66.0$$

Les valeurs des variables auxiliaires $b_{i,k}$ sont :

$$b_{1,1} = 0.0$$
$$b_{1,2} = 0.0$$
$$b_{2,1} = 0.0$$
$$b_{2,2} = 0.0$$

La valeur optimale de $g(x)$ est :

$$g(x) = 198.0$$

Les fichiers **maxOWA.py** et **solver2.py** implémentent la solution.

2.5 Linéariser le critère minOWA des regrets

Rappelons que dans le *minOWA des regrets*, l'objectif est de minimiser la fonction

$$g(x) = \sum_{i=1}^{n} w_i r(x, s(i))$$

où $r(x, s(i))$ est le regret du fait d'avoir choisi x pour le scénario $s(i)$. Le regret est défini comme la différence entre la solution optimale z_i^* et $z_i(x)$, qui est la valeur de l'utilité pour le scénario $s(i)$ lorsque l'on choisit x. Le regret est donc défini comme :

$$r(x, s(i)) = z_i^* - z_i(x)$$

Linéarisation du critère

La formulation linéaire doit transformer cette fonction de minimisation du regret en un problème linéaire que l'on peut résoudre avec Gurobi.

Rappel du critère :

$$\min g(x) = \sum_{i=1}^{n} w_i r(x, s(i)) = \sum_{i=1}^{n} w_i (z_i^* - z_i(x))$$

Cela revient à minimiser la somme pondérée des regrets $r(x, s(i))$, où z_i^* est la valeur optimale pour chaque scénario $s(i)$.

Linéarisation du regret : Pour chaque scénario i, on remplace $r(x, s(i))$ par :

$$r(x, s(i)) = z_i^* - z_i(x)$$

Le critère devient alors :

$$\min \sum_{i=1}^{n} w_i (z_i^* - z_i(x))$$

Ce qui donne la fonction objectif suivante :

$$\min \sum_{i=1}^{n} w_i z_i^* - \sum_{i=1}^{n} w_i z_i(x)$$

Puisque $z_i(x)$ est une fonction linéaire des variables binaires x, on peut décomposer la fonction objectif en deux parties :

$$\min \sum_{i=1}^{n} w_i z_i^* - \sum_{i=1}^{n} w_i \left(\sum_{j=1}^{n} a_{ij} x_j \right)$$

Formalisation

Variables :

- x_j : variable binaire pour chaque projet j.
- $z_i(x)$: la valeur de l'utilité pour le scénario $s(i)$.
- $r(x, s(i))$: regret de choisir x pour le scénario $s(i)$.

Contraintes : Pour chaque i, on doit garantir que le regret $r(x, s(i))$ est bien calculé comme la différence entre la solution optimale z_i^* et la valeur $z_i(x)$. On peut exprimer cela comme :

$$r(x, s(i)) \geq z_i^* - \sum_{j=1}^{n} a_{ij} x_j$$

où a_{ij} représente les utilités pour chaque projet dans chaque scénario.

Solution optimale avec Gurobi

Les valeurs des variables de décision sont les suivantes :

$$x_1 = -0.0$$
$$x_2 = 1.0$$
$$x_3 = 0.0$$
$$x_4 = 1.0$$
$$x_5 = 1.0$$
$$x_6 = -0.0$$
$$x_7 = 1.0$$
$$x_8 = 1.0$$
$$x_9 = 1.0$$
$$x_{10} = -0.0$$

Les valeurs des variables continues r_k sont :

$$r_1 = 50.0$$
$$r_2 = 50.0$$

Les valeurs des variables auxiliaires $b_{i,k}$ sont :

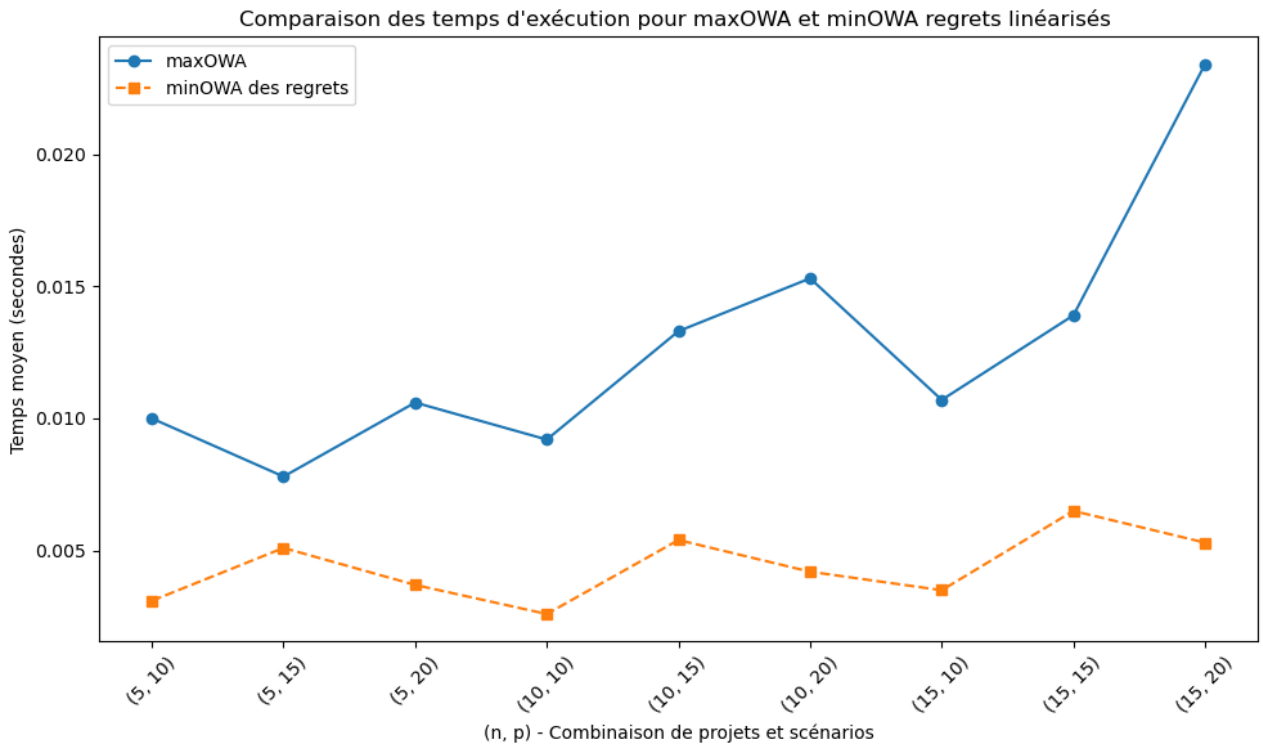
$$b_{1,1} = 0.0$$
$$b_{1,2} = 0.0$$
$$b_{2,1} = 0.0$$
$$b_{2,2} = 0.0$$

La valeur optimale de $g(x)$ est :

$$g(x) = 150.0$$

Les fichiers **minOWA-regrets.py** et **solver2.py** implémentent la solution.

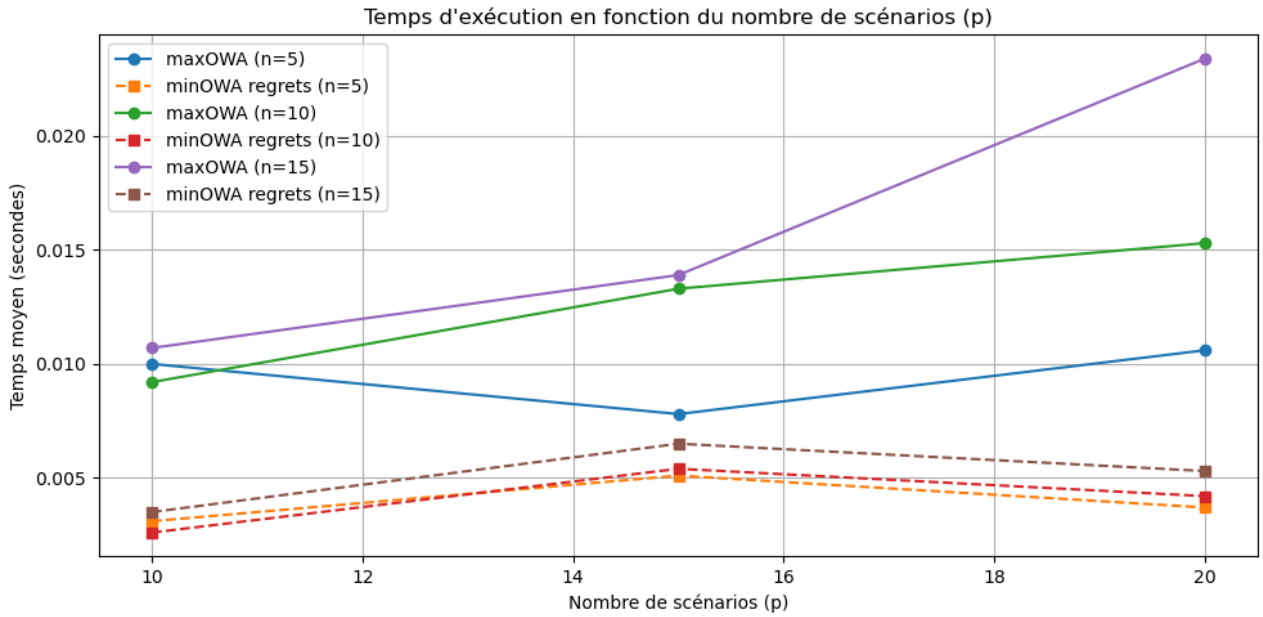
2.6 Evolution du temps de résolution en fonction de n et p



Comme pour la question **1.4**, on modélise avec les fichiers **generator2.py** et **main2.py** avec la même méthode.

Analyse des Résultats

- **Performance** : L'approche **minOWA des regrets linéarisé** est globalement plus rapide que **maxOWA**, avec des temps de calcul en moyenne 2 à 3 fois inférieurs.
- **Scalabilité** : Pour les deux approches, le temps de résolution augmente légèrement avec la taille du problème (n et p). Cette croissance reste modérée, ce qui indique une bonne scalabilité des deux méthodes.
- **Implications** : Pour les problèmes nécessitant une réponse rapide, le **minOWA linéarisé** est préférable. Toutefois, la qualité de la solution obtenue avec **maxOWA** peut être mieux adaptée pour des applications nécessitant plus de robustesse.



Tendance globale :

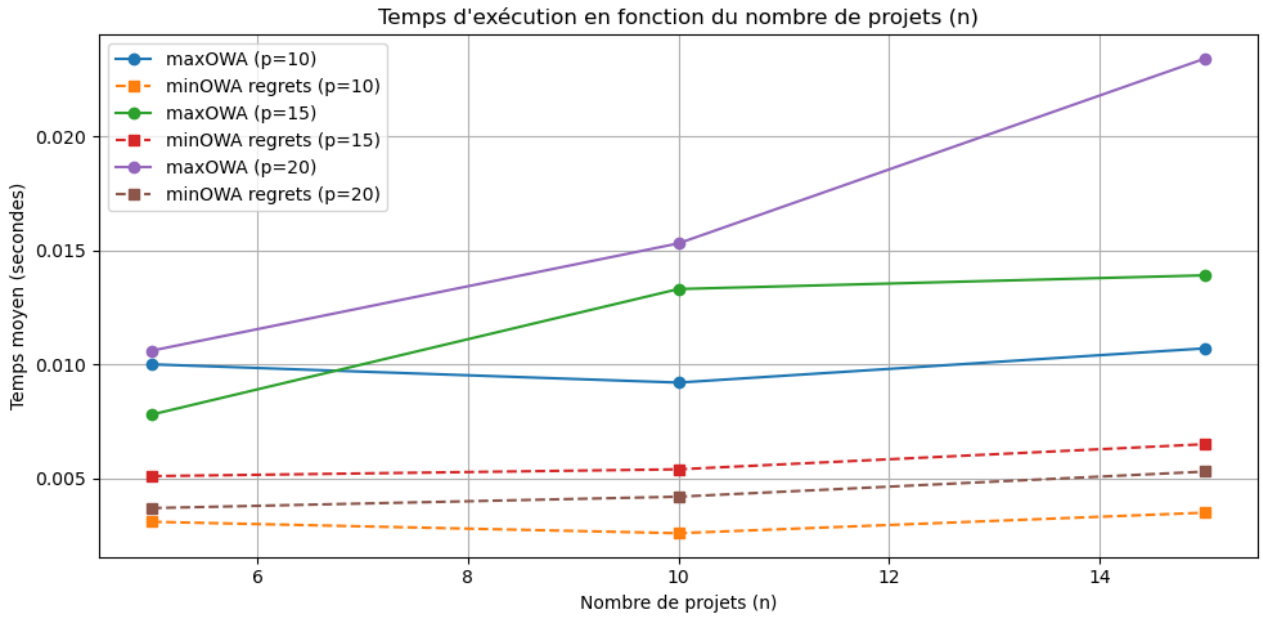
- Le temps d'exécution augmente avec le nombre de scénarios (p), surtout pour l'approche **maxOWA**.
- Pour l'approche **minOWA regrets** , l'augmentation du temps est moins marquée et reste très faible, même avec un nombre de scénarios plus important.

Comparaison entre maxOWA et minOWA regrets :

- **maxOWA** est systématiquement plus lent que **minOWA regrets**. Cette différence de performance devient plus prononcée lorsque le nombre de scénarios (p) et de projets (n) augmente.
- L'écart entre les deux méthodes se voit avec l'augmentation du nombre de projets (n), en particulier pour n=15.

Stabilité des temps pour minOWA regrets :

- Les temps d'exécution pour **minOWA regrets** restent faibles et relativement constants, ce qui montre que cette méthode est plus efficace et moins sensible à l'augmentation du nombre de scénarios.



Tendance globale :

- Le temps d'exécution augmente avec le nombre de projets (n), particulièrement pour l'approche **maxOWA**.
- Pour **minOWA regrets** , l'augmentation est beaucoup moins marquée, et les temps d'exécution restent faibles même pour un nombre de projets plus élevé.

Comparaison entre maxOWA et minOWA regrets :

- **maxOWA** montre une croissance significative du temps d'exécution en fonction du nombre de projets, surtout lorsque le nombre de scénarios (p) est grand (p=20).
- **minOWA regrets** reste beaucoup plus efficace en termes de temps de calcul, avec des temps quasiment constants, quelle que soit la valeur de n.

Impact du nombre de scénarios (p) :

- Pour **maxOWA**, l'augmentation du nombre de scénarios accentue la croissance du temps d'exécution. Par exemple, les courbes pour p=20 montrent une pente plus forte.
- Pour **minOWA regrets** , l'influence du nombre de scénarios est minime. Les courbes restent proches les unes des autres, indiquant une stabilité du temps de résolution.

Pour résumer :

- **maxOWA** est sensible à l'augmentation du nombre de projets, particulièrement pour des instances avec un grand nombre de scénarios.
- **minOWA regrets linéarisé** est beaucoup plus stable et efficace, ce qui en fait une approche plus adaptée pour des applications où le nombre de projets peut être élevé.

3 Application à la recherche d'un chemin robuste dans un graphe

3.1 Programme linéaire plus court chemin

Soit $G = (V, E)$ un graphe orienté quelconque, on peut écrire un programme linéaire pour trouver le chemin le plus rapide de s le sommet source à t le sommet destination tel que :
On note V l'ensemble des sommets, E l'ensemble des arcs, chaque arc $(i, j) \in E$ ayant un coût t_{ij} et x_{ij} une variable binaire indiquant si l'arc (i, j) est utilisé.

$$\min \sum_{(i,j)\in E} t_{ij} \times x_{ij}$$

$$\sum_{(s,j)\in E} x_{sj} = 1 \quad \text{le flot total sortant du sommet source} = 1$$

$$\sum_{(i,t)\in E} x_{it} = 1 \quad \text{le flot total entrant au sommet destination} = 1$$

$$\sum_{(i,v)\in E} x_{iv} = \sum_{(v,j)\in E} x_{vj} \quad \text{flot entrant} = \text{flot sortant}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

3.2 Pour l'exemple 2

3.2.1 Graphe de gauche (a à f)

Scénario 1 :

$$\min 4x_{ab} + 5x_{ac} + 2x_{bc} + 1x_{bd} + 2x_{be} + 7x_{bf} + 5x_{cd} + 2x_{ce} + 3x_{df} + 5x_{ef}$$

$$x_{ab} + x_{ac} = 1$$

$$x_{bf} + x_{df} + x_{ef} = 1$$

$$x_{ab} = x_{bc} + x_{bd} + x_{be} + x_{bf}$$

$$x_{ac} + x_{bc} = x_{cd} + x_{ce}$$

$$x_{bd} + x_{cd} = x_{df}$$

$$x_{be} + x_{ce} = x_{ef}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

On peut réécrire notre programme linéaire tel que :

$$\min 4x_{ab} + 5x_{ac} + 2x_{bc} + 1x_{bd} + 2x_{be} + 7x_{bf} + 5x_{cd} + 2x_{ce} + 3x_{df} + 5x_{ef}$$

$$x_{ab} + x_{ac} = 1$$

$$x_{bf} + x_{df} + x_{ef} = 1$$

$$x_{ab} - x_{bc} - x_{bd} - x_{be} - x_{bf} = 0$$

$$x_{ac} + x_{bc} - x_{cd} - x_{ce} = 0$$

$$x_{bd} + x_{cd} - x_{df} = 0$$

$$x_{be} + x_{ce} - x_{ef} = 0$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

Grâce au solveur :
Soit P le chemin de a à f :

$$x^*_{g1} = (1, 0, 0, 1, 0, 0, 0, 0, 1, 0) \quad \text{correspondant au chemin } (a, b), (b, d), (d, f)$$

Avec x^*_{g1}, $t^1(P) = 8$ et $t^2(P) = 9$ Donc :

$$t(x^*_{g1}) = (8, 9) \quad \text{et} \quad t^*_{g1} = 8$$

Vous pouvez exécuter le script **sg1_optimal.py** pour retrouver cette solution.

Scénario 2 :

$$\min 3x_{ab} + 1x_{ac} + 1x_{bc} + 4x_{bd} + 2x_{be} + 5x_{bf} + 1x_{cd} + 7x_{ce} + 2x_{df} + 2x_{ef}$$
$$x_{ab} + x_{ac} = 1$$
$$x_{bf} + x_{df} + x_{ef} = 1$$
$$x_{ab} = x_{bc} + x_{bd} + x_{be} + x_{bf}$$
$$x_{ac} + x_{bc} = x_{cd} + x_{ce}$$
$$x_{bd} + x_{cd} = x_{df}$$
$$x_{be} + x_{ce} = x_{ef}$$
$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E$$

On peut réécrire notre programme linéaire tel que :

$$\min 3x_{ab} + 1x_{ac} + 1x_{bc} + 4x_{bd} + 2x_{be} + 5x_{bf} + 1x_{cd} + 7x_{ce} + 2x_{df} + 2x_{ef}$$
$$x_{ab} + x_{ac} = 1$$
$$x_{bf} + x_{df} + x_{ef} = 1$$
$$x_{ab} - x_{bc} - x_{bd} - x_{be} - x_{bf} = 0$$
$$x_{ac} + x_{bc} - x_{cd} - x_{ce} = 0$$
$$x_{bd} + x_{cd} - x_{df} = 0$$
$$x_{be} + x_{ce} - x_{ef} = 0$$
$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E$$

Grâce au solveur :
Soit P le chemin de a à f :

$$x^*_{g2} = (0,1,0,0,0,0,1,0,1,0) \quad \text{correspondant au chemin } (a,c),(c,d),(d,f)$$

Avec x^*_{g2}, $t^1(P) = 13$ et $t^2(P) = 4$
Donc :

$$t(x^*_{g2}) = (13,4) \quad \text{et} \quad t^*_{g2} = 4$$

Vous pouvez exécuter le script **sg2_optimal.py** pour retrouver cette solution.

Voici les deux solutions présentées sur le graphe.

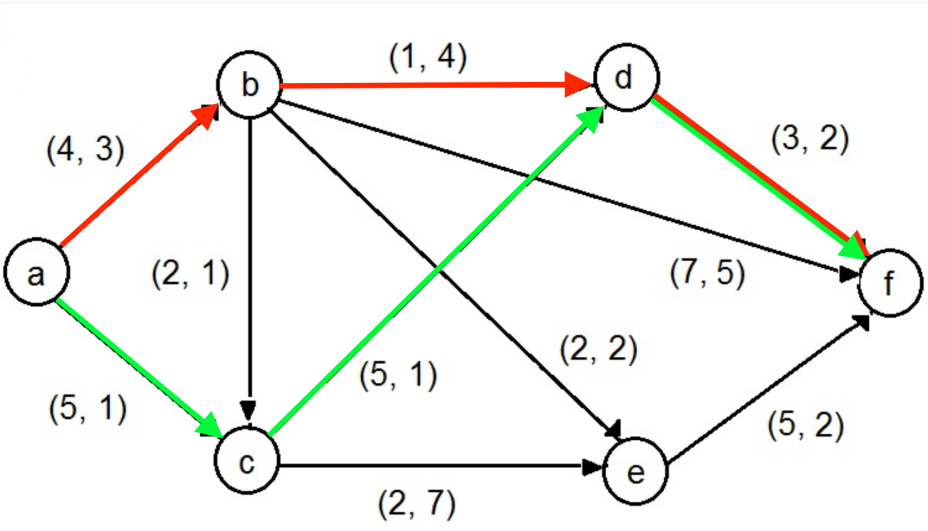


Figure 3: rouge : solution optimale 1, vert : solution optimale 2

3.2.2 Graphe de droite (a à g)

Scénario 1 :

$$\min 5x_{ab} + 10x_{ac} + 2x_{ad} + 4x_{bc} + 1x_{bd} + 4x_{be} + 3x_{ce} + 1x_{cf} + 1x_{dc} + 3x_{df} + 1x_{eg} + 1x_{fg}$$
$$x_{ab} + x_{ac} + x_{ad} = 1$$
$$x_{eg} + x_{fg} = 1$$
$$x_{ab} = x_{bc} + x_{bd} + x_{be}$$
$$x_{ac} + x_{bc} + x_{dc} = x_{ce} + x_{cf}$$
$$x_{ad} + x_{bd} = x_{dc} + x_{df}$$
$$x_{be} + x_{ce} = x_{eg}$$
$$x_{cf} + x_{df} = x_{fg}$$
$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E$$

On peut réécrire notre programme linéaire tel que :

$$\min 5x_{ab} + 10x_{ac} + 2x_{ad} + 4x_{bc} + 1x_{bd} + 4x_{be} + 3x_{ce} + 1x_{cf} + 1x_{dc} + 3x_{df} + 1x_{eg} + 1x_{fg}$$
$$x_{ab} + x_{ac} + x_{ad} = 1$$
$$x_{eg} + x_{fg} = 1$$
$$x_{ab} - x_{bc} - x_{bd} - x_{be} = 0$$
$$x_{ac} + x_{bc} + x_{dc} - x_{ce} - x_{cf} = 0$$
$$x_{ad} + x_{bd} - x_{dc} - x_{df} = 0$$
$$x_{be} + x_{ce} - x_{eg} = 0$$
$$x_{cf} + x_{df} - x_{fg} = 0$$
$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E$$

Grâce au solveur :
Soit P le chemin de a à g :

$$x^*_{d1} = (0,0,1,0,0,0,0,1,1,0,0,1) \quad \text{correspondant au chemin } (a,d),(d,c),(c,f),(f,g)$$

Avec x^*_{d1}, $t(P) = 5$ et $t(P) = 13$ Donc :

$$t(x^*_{d1}) = (5,13) \quad \text{et} \quad t^*_{d1} = 5$$

Vous pouvez exécuter le script **sd1_optimal.py** pour retrouver cette solution.

Scénario 2 :

$$\min 3x_{ab} + 4x_{ac} + 6x_{ad} + 2x_{bc} + 3x_{bd} + 6x_{be} + 1x_{ce} + 2x_{cf} + 4x_{dc} + 5x_{df} + 1x_{eg} + 1x_{fg}$$
$$x_{ab} + x_{ac} + x_{ad} = 1$$
$$x_{eg} + x_{fg} = 1$$
$$x_{ab} = x_{bc} + x_{bd} + x_{be}$$
$$x_{ac} + x_{bc} + x_{dc} = x_{ce} + x_{cf}$$
$$x_{ad} + x_{bd} = x_{dc} + x_{df}$$
$$x_{be} + x_{ce} = x_{eg}$$
$$x_{cf} + x_{df} = x_{fg}$$
$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E$$

On peut réécrire notre programme linéaire tel que :

$$\min 3x_{ab} + 4x_{ac} + 6x_{ad} + 2x_{bc} + 3x_{bd} + 6x_{be} + 1x_{ce} + 2x_{cf} + 4x_{dc} + 5x_{df} + 1x_{eg} + 1x_{fg}$$

$$x_{ab} + x_{ac} + x_{ad} = 1$$

$$x_{eg} + x_{fg} = 1$$

$$x_{ab} - x_{bc} - x_{bd} - x_{be} = 0$$

$$x_{ac} + x_{bc} + x_{dc} - x_{ce} - x_{cf} = 0$$

$$x_{ad} + x_{bd} - x_{dc} - x_{df} = 0$$

$$x_{be} + x_{ce} - x_{eg} = 0$$

$$x_{cf} + x_{df} - x_{fg} = 0$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

Grâce au solveur :
Soit P le chemin de a à g :

$$x^*_{d2} = (0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0) \quad \text{correspondant au chemin } (a, c), (c, e), (e, g)$$

Avec x^*_{d2}, $t(P) = 14$ et $t(P) = 6$ Donc :

$$t(x^*_{d2}) = (14, 6) \quad \text{et} \quad t^*_{d2} = 6$$

Vous pouvez exécuter le script **sd2_optimal.py** pour retrouver cette solution.

Voici les deux solutions présentées sur le graphe.

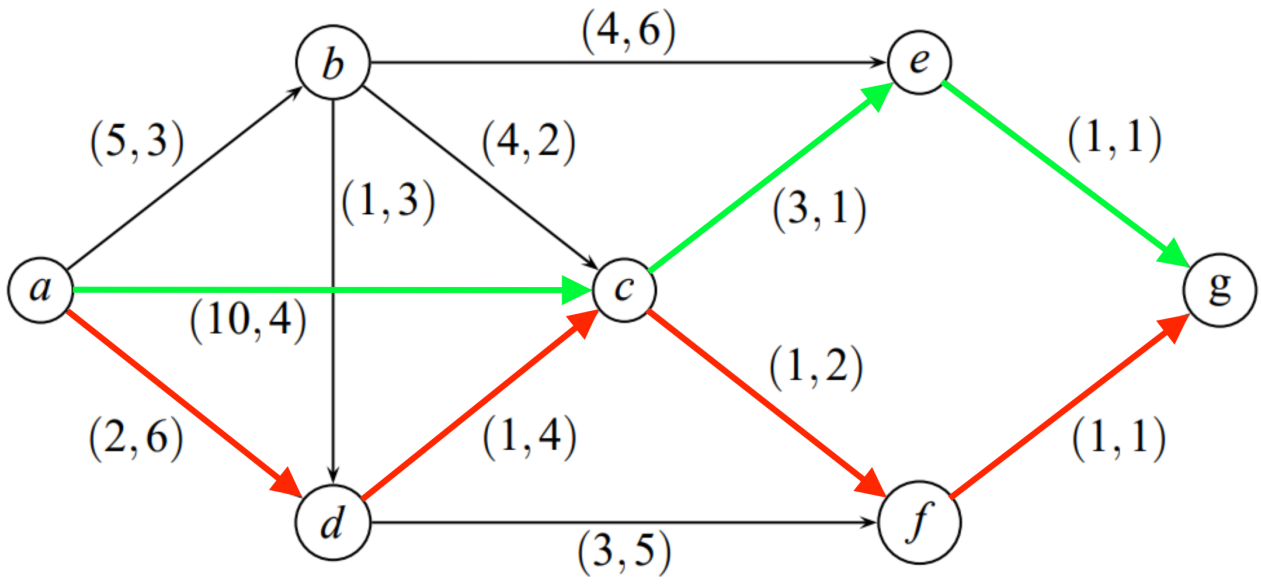


Figure 4: rouge : solution optimale 1, vert : solution optimale 2

3.3 Question 3.3

3.3.1 maxmin (-t)

On veut appliquer la méthode maxmin qu'on a vu précédemment à l'exemple 2.
Graphe de gauche

$$\max \min_{i=1}^{2} -t^i(x)$$

$$\max \min\{-t^1(x), -t^2(x)\}$$

Pour linéariser ce programme on introduit la variable z tel que $z = \min\{-t^1(x), -t^2(x)\}$
Et on introduit également les contraintes sur le flot.

$$\max z$$

$$z \leq -t^1(x) = -4x_{ab} - 5x_{ac} - 2x_{bc} - 1x_{bd} - 2x_{be} - 7x_{bf} - 5x_{cd} - 2x_{ce} - 3x_{df} - 5x_{ef}$$

$$z \leq -t^2(x) = -3x_{ab} - 1x_{ac} - 1x_{bc} - 4x_{bd} - 2x_{be} - 5x_{bf} - 1x_{cd} - 7x_{ce} - 2x_{df} - 2x_{ef}$$

$$x_{ab} + x_{ac} = 1$$

$$x_{bf} + x_{df} + x_{ef} = 1$$

$$x_{ab} = x_{bc} + x_{bd} + x_{be} + x_{bf}$$

$$x_{ac} + x_{bc} = x_{cd} + x_{ce}$$

$$x_{bd} + x_{cd} = x_{df}$$

$$x_{be} + x_{ce} = x_{ef}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

On peut réécrire notre programme linéaire tel que :

$$\max z$$

$$4x_{ab} + 5x_{ac} + 2x_{bc} + 1x_{bd} + 2x_{be} + 7x_{bf} + 5x_{cd} + 2x_{ce} + 3x_{df} + 5x_{ef} + z \leq 0$$

$$3x_{ab} + 1x_{ac} + 1x_{bc} + 4x_{bd} + 2x_{be} + 5x_{bf} + 1x_{cd} + 7x_{ce} + 2x_{df} + 2x_{ef} + z \leq 0$$

$$x_{ab} + x_{ac} = 1$$

$$x_{bf} + x_{df} + x_{ef} = 1$$

$$x_{ab} - x_{bc} - x_{bd} - x_{be} - x_{bf} = 0$$

$$x_{ac} + x_{bc} - x_{cd} - x_{ce} = 0$$

$$x_{bd} + x_{cd} - x_{df} = 0$$

$$x_{be} + x_{ce} - x_{ef} = 0$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

Pour résoudre certains problèmes de compatibilité des contraintes, on peut réécrire tel que:

$$\max -z$$

$$-4x_{ab} - 5x_{ac} - 2x_{bc} - 1x_{bd} - 2x_{be} - 7x_{bf} - 5x_{cd} - 2x_{ce} - 3x_{df} - 5x_{ef} + z \geq 0$$

$$-3x_{ab} - 1x_{ac} - 1x_{bc} - 4x_{bd} - 2x_{be} - 5x_{bf} - 1x_{cd} - 7x_{ce} - 2x_{df} - 2x_{ef} + z \geq 0$$

$$x_{ab} + x_{ac} = 1$$

$$x_{bf} + x_{df} + x_{ef} = 1$$

$$x_{ab} - x_{bc} - x_{bd} - x_{be} - x_{bf} = 0$$

$$x_{ac} + x_{bc} - x_{cd} - x_{ce} = 0$$

$$x_{bd} + x_{cd} - x_{df} = 0$$

$$x_{be} + x_{ce} - x_{ef} = 0$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

Grâce au solveur : Soit P le chemin de a à f :

$$x_g^* = (1,0,0,1,0,0,0,0,1,0) \quad \text{correspondant au chemin } (a,b),(b,d),(d,f)$$

Avec x_g^*, $t^1(P) = 8$ et $t^2(P) = 9$ Donc :

$$t(x_g^*) = (8,9)$$

Vous pouvez exécuter le script **maxmin_g.py** pour retrouver cette solution.

Graphe de droite

$$\max \min\{t^1(x), t^2(x)\}$$

$$\max -z$$
$$z \geq t^1(x) = 5x_{ab} + 10x_{ac} + 2x_{ad} + 4x_{bc} + 1x_{bd} + 4x_{be} + 3x_{ce} + 1x_{cf} + 1x_{dc} + 3x_{df} + 1x_{eg} + 1x_{fg}$$
$$z \geq t^2(x) = 3x_{ab} + 4x_{ac} + 6x_{ad} + 2x_{bc} + 3x_{bd} + 6x_{be} + 1x_{ce} + 2x_{cf} + 4x_{dc} + 5x_{df} + 1x_{eg} + 1x_{fg}$$
$$x_{ab} + x_{ac} + x_{ad} = 1$$
$$x_{eg} + x_{fg} = 1$$
$$x_{ab} = x_{bc} + x_{bd} + x_{be}$$
$$x_{ac} + x_{bc} + x_{dc} = x_{ce} + x_{cf}$$
$$x_{ad} + x_{bd} = x_{dc} + x_{df}$$
$$x_{be} + x_{ce} = x_{eg}$$
$$x_{cf} + x_{df} = x_{fg}$$
$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E$$

On peut réécrire notre programme linéaire tel que :

$$\max -z$$
$$-5x_{ab} - 10x_{ac} - 2x_{ad} - 4x_{bc} - 1x_{bd} - 4x_{be} - 3x_{ce} - 1x_{cf} - 1x_{dc} - 3x_{df} - 1x_{eg} - 1x_{fg} + z \geq 0$$
$$-3x_{ab} - 4x_{ac} - 6x_{ad} - 2x_{bc} - 3x_{bd} - 6x_{be} - 1x_{ce} - 2x_{cf} - 4x_{dc} - 5x_{df} - 1x_{eg} - 1x_{fg} + z \geq 0$$
$$x_{ab} + x_{ac} + x_{ad} = 1$$
$$x_{eg} + x_{fg} = 1$$
$$x_{ab} - x_{bc} - x_{bd} - x_{be} = 0$$
$$x_{ac} + x_{bc} + x_{dc} - x_{ce} - x_{cf} = 0$$
$$x_{ad} + x_{bd} - x_{dc} - x_{df} = 0$$
$$x_{be} + x_{ce} - x_{eg} = 0$$
$$x_{cf} + x_{df} - x_{fg} = 0$$
$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E$$

Grâce au solveur : Soit P le chemin de a à g :

$$x_d^* = (1,0,0,0,0,1,0,0,0,0,1,0) \quad \text{correspondant au chemin } (a,b),(b,e),(e,g)$$

Avec x_d^*, $t^1(P) = 10$ et $t^2(P) = 10$ Donc :

$$t(x_d^*) = (10,10)$$

Vous pouvez exécuter le script **maxmin_d.py** pour retrouver cette solution.

3.3.2 minmax regret -t

On veut appliquer la méthode minmax regret qu'on a vu précédemment à l'exemple 2:
Graphe de gauche

$$\min \max_{i=1}^{2} t^*_{gi} - (-t^i(x))$$

$$\min \max\{t^*_{g1} + t^1(x), t^*_{g2} + t^2(x)\}$$

Comme dans la question **3.2** on a calculé les t^*_{gi}.
On a directement : $t^*_{g1} = 8$ et $t^*_{g2} = 4$
Pour linéariser ce programme on introduit la variable z tel que $z = \max\{t^*_{g1} + t^1(x), t^*_{g2} + t^2(x)\}$
Et on introduit également la contrainte sur le flot.

$$\min z$$

$$z \geq t^*_{g1} + t^1(x) = 8 + 4x_{ab} + 5x_{ac} + 2x_{bc} + 1x_{bd} + 2x_{be} + 7x_{bf} + 5x_{cd} + 2x_{ce} + 3x_{df} + 5x_{ef}$$

$$z \geq t^*_{g2} + t^2(x) = 4 + 3x_{ab} + 1x_{ac} + 1x_{bc} + 4x_{bd} + 2x_{be} + 5x_{bf} + 1x_{cd} + 7x_{ce} + 2x_{df} + 2x_{ef}$$

$$x_{ab} + x_{ac} = 1$$

$$x_{bf} + x_{df} + x_{ef} = 1$$

$$x_{ab} = x_{bc} + x_{bd} + x_{be} + x_{bf}$$

$$x_{ac} + x_{bc} = x_{cd} + x_{ce}$$

$$x_{bd} + x_{cd} = x_{df}$$

$$x_{be} + x_{ce} = x_{ef}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

On peut réécrire notre programme linéaire tel que :

$$\min z$$

$$-4x_{ab} - 5x_{ac} - 2x_{bc} - 1x_{bd} - 2x_{be} - 7x_{bf} - 5x_{cd} - 2x_{ce} - 3x_{df} - 5x_{ef} + z \geq 8$$

$$-3x_{ab} - 1x_{ac} - 1x_{bc} - 4x_{bd} - 2x_{be} - 5x_{bf} - 1x_{cd} - 7x_{ce} - 2x_{df} - 2x_{ef} + z \geq 4$$

$$x_{ab} + x_{ac} = 1$$

$$x_{bf} + x_{df} + x_{ef} = 1$$

$$x_{ab} - x_{bc} - x_{bd} - x_{be} - x_{bf} = 0$$

$$x_{ac} + x_{bc} - x_{cd} - x_{ce} = 0$$

$$x_{bd} + x_{cd} - x_{df} = 0$$

$$x_{be} + x_{ce} - x_{ef} = 0$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

Grâce au solveur : Soit P le chemin de a à f :

$$x^{'*}_g = (1, 0, 0, 1, 0, 0, 0, 0, 1, 0) \quad \text{correspondant au chemin } (a, b), (b, d), (d, f)$$

Avec $x^{'*}_g$, $t^1(P) = 8$ et $t^2(P) = 9$ Donc :

$$t(x^{'*}_g) = (8, 9)$$

Vous pouvez exécuter le script **minmax_regret_g.py** pour retrouver cette solution.

Graphe de droite

$$\min \max_{i=1}^{2} t^*_{di} - (-t^i(x))$$

$$\min \max\{t^*_{d1} + t^1(x), t^*_{d2} + t^2(x)\}$$

Comme dans la question **3.2** on a calculé les t^*_{di}.
On a directement : $t^*_{d1} = 5$ et $t^*_{d2} = 6$
Pour linéariser ce programme linéaire on introduit la variable z tel que $z = \max\{t^*_{d1} + t^1(x), t^*_{d2} + t^2(x)\}$
Et on introduit également la contrainte sur le flot.

$$\min z$$

$$z \geq t^*_{d1} + t^1(x) = 5 + 5x_{ab} + 10x_{ac} + 2x_{ad} + 4x_{bc} + 1x_{bd} + 4x_{be} + 3x_{ce} + 1x_{cf} + 1x_{dc} + 3x_{df} + 1x_{eg} + 1x_{fg}$$

$$z \geq t^*_{g2} + t^2(x) = 6 + 3x_{ab} + 4x_{ac} + 6x_{ad} + 2x_{bc} + 3x_{bd} + 6x_{be} + 1x_{ce} + 2x_{cf} + 4x_{dc} + 5x_{df} + 1x_{eg} + 1x_{fg}$$

$$x_{ab} + x_{ac} + x_{ad} = 1$$

$$x_{eg} + x_{fg} = 1$$

$$x_{ab} = x_{bc} + x_{bd} + x_{be}$$

$$x_{ac} + x_{bc} + x_{dc} = x_{ce} + x_{cf}$$

$$x_{ad} + x_{bd} = x_{dc} + x_{df}$$

$$x_{be} + x_{ce} = x_{eg}$$

$$x_{cf} + x_{df} = x_{fg}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

On peut réécrire notre programme linéaire tel que :

$$\min z$$

$$-5x_{ab} - 10x_{ac} - 2x_{ad} - 4x_{bc} - 1x_{bd} - 4x_{be} - 3x_{ce} - 1x_{cf} - 1x_{dc} - 3x_{df} - 1x_{eg} - 1x_{fg} + z \geq 5$$

$$-3x_{ab} - 4x_{ac} - 6x_{ad} - 2x_{bc} - 3x_{bd} - 6x_{be} - 1x_{ce} - 2x_{cf} - 4x_{dc} - 5x_{df} - 1x_{eg} - 1x_{fg} + z \geq 6$$

$$x_{ab} + x_{ac} + x_{ad} = 1$$

$$x_{eg} + x_{fg} = 1$$

$$x_{ab} - x_{bc} - x_{bd} - x_{be} = 0$$

$$x_{ac} + x_{bc} + x_{dc} - x_{ce} - x_{cf} = 0$$

$$x_{ad} + x_{bd} - x_{dc} - x_{df} = 0$$

$$x_{be} + x_{ce} - x_{eg} = 0$$

$$x_{cf} + x_{df} - x_{fg} = 0$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

Grâce au solveur : Soit P le chemin de a à g :

$$x^{'*}_d = (1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1) \quad \text{correspondant au chemin } (a, b), (b, c), (c, f), (f, g)$$

Avec $x^{'*}_d$, $t^1(P) = 7$ et $t^2(P) = 12$ Donc :

$$t(x^{'*}_d) = (7, 12)$$

Vous pouvez exécuter le script **minmax_regret_d.py** pour retrouver cette solution.

3.3.3 Résulats

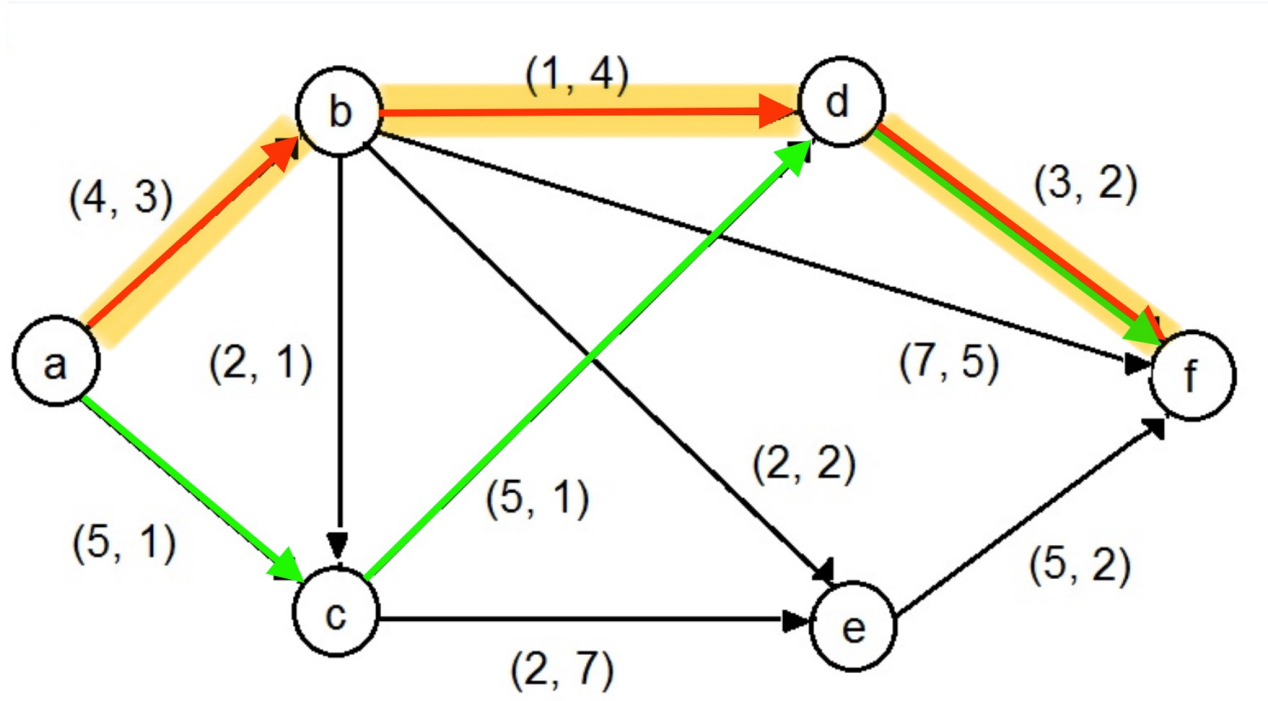


Figure 5: Jaune : solution trouvée avec maxmin et minmax regret

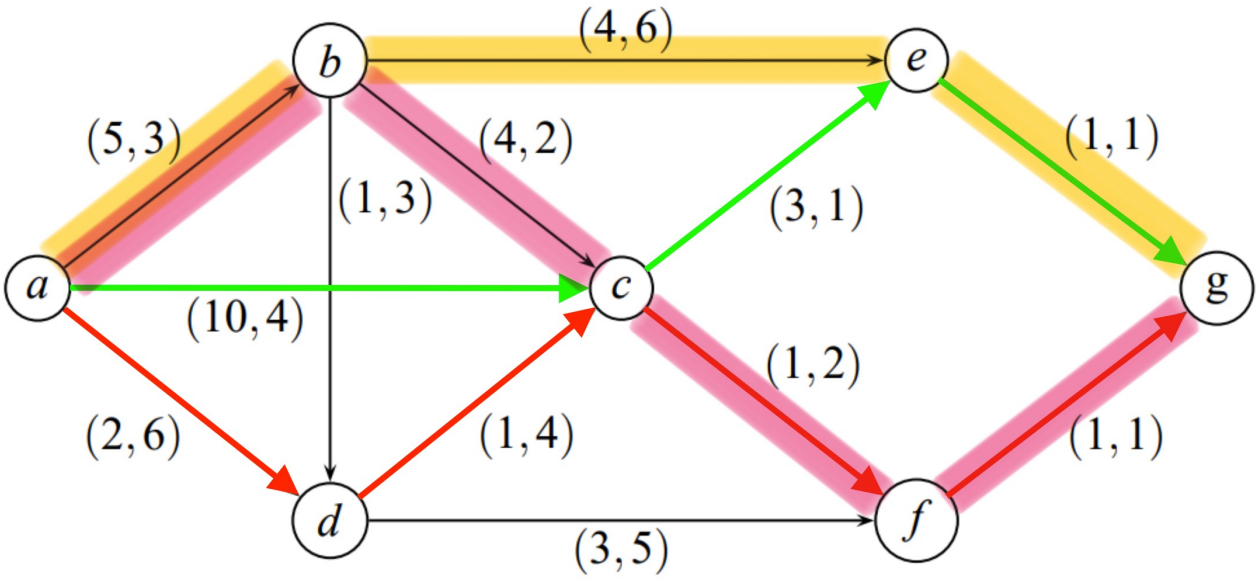


Figure 6: Jaune : solution trouvée avec maxmin, Rose : solution trouvée avec min max regret

3.3.4 maxOWA

Formulation du problème :

Le problème peut être formulé comme suit :

$$\max\left(w_1 \cdot t_1(x) + w_2 \cdot t_2(x)\right)$$

où $t_1(x)$ et $t_2(x)$ sont les coûts associés à chaque arc dans les deux scénarios s_1 et s_2, respectivement.
On cherche à maximiser la fonction g, qui est une combinaison des coûts $t_1(x)$ et $t_2(x)$ pondérée par les coefficients w_1 et w_2.

Afin de linéariser le problème, on introduit des variables de décision et on définit les contraintes sur le flot dans le graphe.

$$\max z$$
$$z \leq -t_1(x) = -4x_{ab} - 5x_{ac} - 2x_{bc} - 1x_{bd} - 2x_{be} - 7x_{bf} - 5x_{cd} - 2x_{ce} - 3x_{df} - 5x_{ef}$$
$$z \leq -t_2(x) = -3x_{ab} - 1x_{ac} - 1x_{bc} - 4x_{bd} - 2x_{be} - 5x_{bf} - 1x_{cd} - 7x_{ce} - 2x_{df} - 2x_{ef}$$

Les contraintes sur le flot entre les noeuds sont les suivantes :

$$x_{ab} + x_{ac} = 1$$
$$x_{bf} + x_{df} + x_{ef} = 1$$
$$x_{ab} = x_{bc} + x_{bd} + x_{be} + x_{bf}$$
$$x_{ac} + x_{bc} = x_{cd} + x_{ce}$$
$$x_{bd} + x_{cd} = x_{df}$$
$$x_{be} + x_{ce} = x_{ef}$$
$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

Reformulation linéaire :

La fonction objectif devient la suivante :

$$\max z$$
$$4x_{ab} + 5x_{ac} + 2x_{bc} + 1x_{bd} + 2x_{be} + 7x_{bf} + 5x_{cd} + 2x_{ce} + 3x_{df} + 5x_{ef} + z \leq 0$$
$$3x_{ab} + 1x_{ac} + 1x_{bc} + 4x_{bd} + 2x_{be} + 5x_{bf} + 1x_{cd} + 7x_{ce} + 2x_{df} + 2x_{ef} + z \leq 0$$

Les contraintes de flot sont réécrites comme suit :

$$x_{ab} + x_{ac} = 1$$
$$x_{bf} + x_{df} + x_{ef} = 1$$
$$x_{ab} - x_{bc} - x_{bd} - x_{be} - x_{bf} = 0$$
$$x_{ac} + x_{bc} - x_{cd} - x_{ce} = 0$$
$$x_{bd} + x_{cd} - x_{df} = 0$$
$$x_{be} + x_{ce} - x_{ef} = 0$$

Résolution avec le solveur Gurobi :

Le chemin optimal P du graphe de gauche correspond aux variables de décision suivantes :

$$x_g^* = (1, 0, 0, 1, 0, 0, 0, 0, 1, 0)$$

Cela correspond au chemin (a, b), (b, d), (d, f) du graphe. Le coût associé pour les deux scénarios est $t_1(P) = 8$ et $t_2(P) = 9$, ce qui donne un vecteur de coûts $t(x_g^*) = (8, 9)$.
La solution optimale du problème est donnée par la valeur de la fonction objectif, que nous avons calculée avec le solveur Gurobi.

$$t(x_g^*) = (8, 9)$$

Vous pouvez exécuter le script **maxOWA_g.py** pour retrouver cette solution.

minOWA

3.4 Etude de l'évolution du temps de résolution pour la recherche d'un chemin optimal au sens de chacun des critères (maxmin, minmax regret, maxOWA et minOWA des regrets) en fonction de n et p

Dans cette section, nous cherchons à généraliser l'exemple 2 en considérant p sommets et n scénarios. L'objectif est d'analyser l'évolution du temps de résolution en fonction de n et p. Plus précisément, pour n = 2, 5, 10 et p = 10, 15, 20, nous générerons 10 instances du problème pour chaque couple (n, p). Les coûts des arcs seront tirés aléatoirement dans l'intervalle [0, 100] et la densité des arcs sera d'environ 50

3.4.1 minmax et maxmin regret

Et on procède ainsi pour les 2 méthodes : maxmin et minmax regret.
On remarque que résoudre le PL maxmin revient à avoir les paramètres :
Matrice des contraintes :

$$a = \begin{bmatrix} \text{les coûts dans [-100; 0]} & \dots & 1 \\ & \dots & \dots & 1 \\ \text{les contraintes sur les arcs} & \dots & 0 \end{bmatrix}$$

Second membre :

$$b = \begin{bmatrix} 0 \\ \dots \\ 0 \\ 1 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

Fonction objectif :

$$c = \begin{bmatrix} 0 & \dots & -1 \end{bmatrix}$$

On a alors écrit la fonction **generator_maxmin(n,p)** qui prend en paramètre le nombre de scénario n et sommets p et retourne les 3 matrices a, b et c correspondant à notre PL. En ayant généré aléatoirement les coûts et arcs tout en respectant nos conditions.
Vous pouvez consulter son fonctionnement et le tester dans le script **generator.py**.

Pour minmax regret on remarque également que résoudre le PL revient à avoir les paramètres :
Matrice des contraintes :

$$a = \begin{bmatrix} \text{les coûts dans [-100; 0]} & \dots & 1 \\ & \dots & \dots & 1 \\ \text{les contraintes sur les arcs} & \dots & 0 \end{bmatrix}$$

Second membre :

$$b = \begin{bmatrix} \text{solution optimale scénario 1} \\ \dots \\ \text{solution optimale scénario n-1} \\ 1 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

Fonction objectif :

$$c = \begin{bmatrix} 0 & \dots & 1 \end{bmatrix}$$

De même on a écrit la fonction **generator_minmaxregret(n,p)** que vous pouvez également consulter et tester dans les script **generator.py**.

Maintenant qu'on peut générer autant de PL qu'on veut.
On peut calcluer le temps d'éxecution en fonction de n et p.
Pour cela vous pouvez consulter le script **main.py**

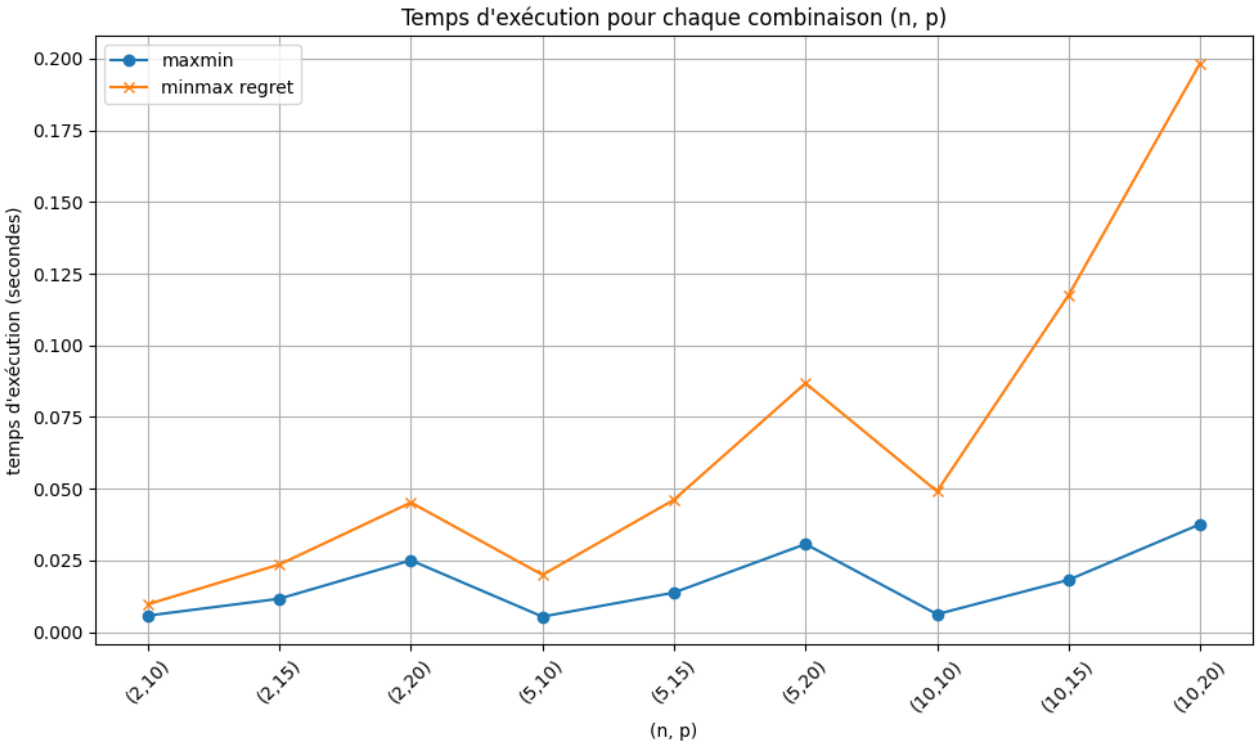


Figure 7: Temps d'execution en fonction de n et p

Le graphe est très similaire à celui de la question **1.4**.