Projet Mogpl
Optimisation robuste dans l'incertain total

Partie 1

1.1. On traite la linéarisation du critère maximin dans le contexte de la sélection de projets sous incertitude. On cherche la solution dont l'évaluation dans le pire scénario est la meilleure possible.

Considérons l'ensemble de $p = 10$ projets, caractérisés comme suit par des coût (c) et deux variables d'utilités (s^1 et s^2) :

$$c = (60, 10, 15, 20, 25, 20, 5, 15, 20, 60)$$
$$s^1 = (70, 18, 16, 14, 12, 10, 8, 6, 4, 2)$$
$$s^2 = (2, 4, 6, 8, 10, 12, 14, 16, 18, 70)$$

Dans ce contexte, une solution réalisable est caractérisée par un vecteur $x \in \{0,1\}^p$ satisfaisant la contrainte budgétaire $\sum_{j=1}^p c_j x_j \leq B$ avec $B = 100$. Pour toute solutoin x, on note $z(x) = (z_1(x), z_2(x))$ son vecteur image où l'utilité dans chaque scénario est donnée par :

$$\begin{cases} z_1(x) = \sum_{j=1}^p s_j^1 x_j \\ z_2(x) = \sum_{j=1}^p s_j^2 x_j \end{cases}$$

Le problème d'optimisation initial s'écrit alors :

$$\max_{x \in X} g(x) = \max_{x \in X} \min\{z_1(x), z_2(x)\}$$

où X représente l'ensemble des solutions réalisables défini par :

$$X = \left\{ x \in \{0,1\}^p : \sum_{j=1}^p c_j x_j \leq B \right\}$$

Pour obtenir un programme linéraire en variables mixtes, nous introduisons une variable α représentant le minimum des utilités. Le problème se reformule alors :

$$\max \alpha$$
$$s.c. \begin{cases} \alpha \leq \sum_{j=1}^p s_j^1 x_j \\ \alpha \leq \sum_{j=1}^p s_j^2 x_j \\ \sum_{j=1}^p c_j x_j \leq B \end{cases}$$
$$x_j \in \{0,1\} \quad \forall j \in \{1, ..., p\}, \alpha \in \mathbb{R}$$

L'implémentation de ce programme linéaire a été réalisée en Python à l'aide de la librairie `pulp` et du solveur `gurobi` (voir le fichier `src/q11.py`). La résolution nous fournit les résultats suivants.

La solution optimale x^* est un vecteur binaire où seuls les projets 2, 3, 4, 7, 8 et 9 sont sélectionnés, ce qui s'écrit :

Vecteur x^* : $(0, 1, 1, 1, 0, 0, 1, 1, 1, 0)$
Coût total : $85K€$
Valeur image : $z(x^*) = (66, 66)$
Valeur optimale : $g(x^*) = 66$

Il est intéressant de noter que cette solution atteint exactement la même utilité dans les deux scénarios ($z_1(x^*) = z_2(x^*) = 66$), ce qui suggère un bon équilibre entre les deux scénarios.

Il est intéressant de noter que l'utilisation d'autres solveurs nous conduit à une solution alternative :

Vecteur x^* : $(0, 0, 1, 1, 1, 1, 1, 1, 0, 0)$
Coût total : $100K€$
Valeur image : $z(x^*) = (66, 66)$
Valeur optimale : $g(x^*) = 66$

Bien que ces deux solutions soient équivalentes du point de vue de notre critère maxmin, atteignant la même valeur optimale $g(x^*) = 66$, elles diffèrent par leur efficience économique. En effet, la seconde solution mobilise l'intégralité du budget pour atteindre le même niveau d'utilité que la première qui n'en utilise que 85%. La

minimisation des coûts n'étant pas un objectif de notre programme linéaire, ces deux solutions sont mathématiquement équivalentes, bien que la première apparaisse plus avantageuse d'un point de vue pratique.

1.2. Dans cette partie, nous traitons la linéarisation du critère minmax regret, qui est très similaire au problème précédent mais avec une approche différente de l'évaluation des solutions. En effet, plutôt que de considérer directement les utilités dans chaque scénario, nous nous intéressons maintenant au « regret » - c'est-à-dire à la différence entre l'utilité obtenue et la meilleure utilité possible dans chaque scénario.

Reprenons les données de la partie 1.1 avec les mêmes vecteurs de coûts et d'utilités.

La première étape consiste à déterminer les utilités optimales z_1^* et z_2^* pour chaque scénario, qui sont obtenues en résolvant deux problèmes d'optimisations distincts :

$$z_i^* = \max_{x \in X} z_{i(x)} = \max_{x \in X} \sum_{j=1}^{p} s_j^i x_j \quad \forall i \in \{1, 2\}$$

où X représente toujours l'ensemble des solutions réalisables défini par :

$$X = \left\{ x \in \{0,1\}^p : \sum_{j=1}^{p} c_j x_j \leq B \right\}$$

Le critère minmax regret cherche alors à minimiser le regret maximum sur l'ensemble des scénarios. Pour toute solution x, le regret dans le scénario i est donné par $r(x, s_i) = z_1^* - z_{i(x)}$. Le problème se formule alors :

$$\min_{x \in X} g(x) = \min_{x \in X} \max\{r(x, s_1), r(x, s_2)\}$$

Pour obtenir un programme linéaire en variables mixtes, nous introduisons une variables β représentant le regret maximum. Le problème se reformule alors :

$$\min \beta$$
$$s.c. \begin{cases} \beta \geq z_1^* - \sum_{j=1}^{p} s_j^1 x_j \\ \beta \geq z_2^* - \sum_{j=1}^{p} s_j^2 x_j \\ \sum_{j=1}^{p} c_j x_j \leq B \end{cases}$$
$$x_j \in \{0,1\} \quad \forall j \in \{1, ..., p\}, \beta \in \mathbb{R}$$

L'implémentation de ce programme linéaire a été réalisée en Python (voir le fichier `src/q12.py`). La résolution nous fournit les résultats suivants.

Vecteur x^* : $(0, 1, 1, 0, 0, 1, 1, 1, 1, 0)$
Coût total : $85K€$
Regrets : $r(x^*) = (50, 48)$
Valeur optimale : $g(x^*) = 50$

Le regret maximum est presque équilibré entre les deux scénarios (50 et 48), ce qui suggère que la solution est bien équilibrée au sens de ce nouveau critère. On note finalement que cette solution est différente de la solution proposée à la question 1.1, un nouveau critère résulte bien ici en une solution différente.

1.3. On cherche à représenter l'utilité de chacune des solutions x_1^*, x_2^*, x^* et x'^* dans chacun des scénarios. L'implémentation du calcul des points a été réalisée en Python (voir le fichier `src/q13.py`). On obtient les points $x_1^* = (112, 26)$, $x_2^* = (20, 118)$, $x^* = (66, 66)$ et $x'^* = (62, 70)$ illustrés dans le graphe suivant :

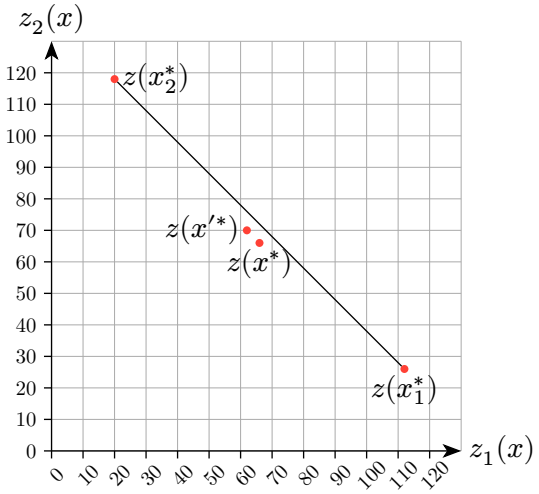

Fig. 4. – Représentation des différentes solutions dans chaque scénarios

Le segment ici représenté illustre le fait que ces nouveaux critères ne sont pas une simple pondération des solutions x_1^* et x_2^*. En effet, si on avait introduit une nouvelle variable $\lambda \in [0, 1]$ et que l'on avait calculé par simple pondération des deux solutions une nouvelle solution en faisant :

$$(1 - \lambda)x_1^* + \lambda x_2^*$$

Alors, on aurait obtenue une solution sur le segment. Cependant, graphiquement, on peut voir que nos deux solutions x^* et x'^* ne sont pas sur ce segment. Les critère maxmin et minmax regret sont donc bien différents et représentent une nouvelle façon d'envisager le problème.

1.4. On s'intéresse maintenant à l'évolution des temps de résolution des deux critères (maxmin et minmax regret) en fonction du nombre de scénarios ($n = \{5, 10, 15, ..., 50\}$) et du nombre de projets ($p = \{10, 15, 20, ..., 50\}$). Pour évaluer systématiquement leurs performances respectives, nous générons pour chaque couple (n, p) un ensemble de 50 instances aléatoires. Les coûts et utilités de chaque instance sont tirés uniformément dans l'intervalle $[1, 100]$, avec un budget fixé à 50% de la somme totale des coûts des projets. Cette approche nous permet d'obtenir une distribution statistiquement significative des temps de résolution pour différentes tailles de problèmes.

L'implémentation de ce programme linéaire a été réalisée en Python à l'aide de la librairie `pulp` et du solveur `gurobi` (voir le fichier `src/q14.py`). La résolution nous fournit les résultats suivants.

L'analyse des temps de résolution révèle un comportement différent entre l'augmentation du nombre de scénarios et celle du nombre de projets. La croissance linéaire observée avec le nombre de scénarios s'explique par la structure même des programmes linéaires : chaque nouveau scénario ajoute simplement un nouvel ensemble de contraintes linéaires au problème, sans modifier la nature combinatoire du problème sous-jacent.

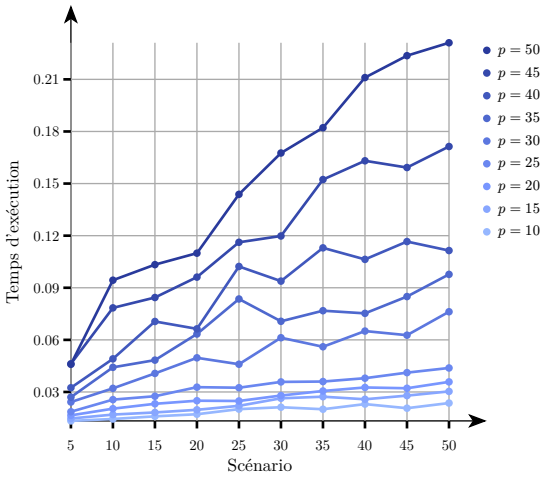

Fig. 5. – Maxmin par scénarios

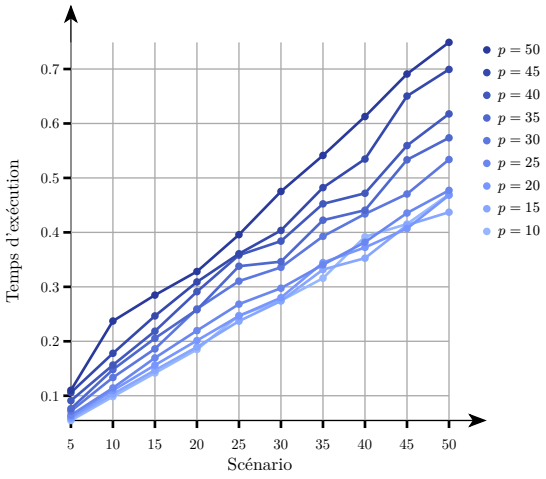

Fig. 6. – Minmax regret par scénarios

En revance, l'ajout de nouveaux projets impacte directement la complexité combinatoire du problème. Le problème de sélection de projets sous contrainte budgétaire est une variante du problème du sac à dos, connu pour être NP-complet. Chaque nouveau projet double potentiellement l'espace des solutions à explorer, ce qui explique la croissance exponentielle observée des temps de calcul. En effet, avec p projets, l'espace des solutions possibles est de taille 2^p, et même les algorithmes les plus sophistiqués ne peuvent échapper à cette complexité fondamentale dans les pires cas.

Finalement, cette analyse suggère qu'il est relativement peu coûteux d'envisager de nombreux scénarios différents, tandis que l'ajout de nouveaux projets complexifie rapidement le problème. Cette propriété est particulièrement intéressante

dans un contexte d'optimisation robuste, où l'on cherche à se prémunir contre différents scénarios possibles : on peut explorer un large éventail de futurs possibles sans que cela n'impacte drastiquement la complexité de résolution du problème.

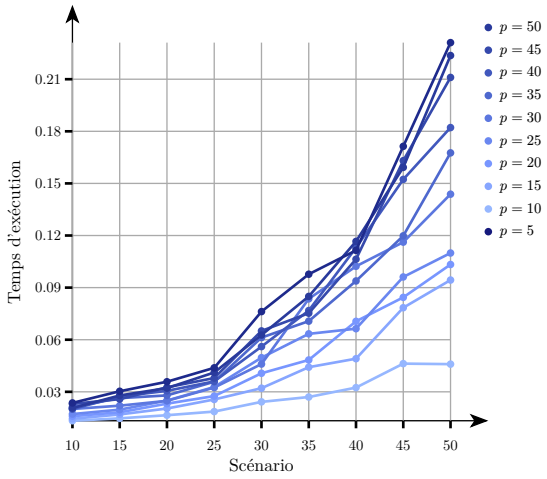

Fig. 7. – Maxmin par projets

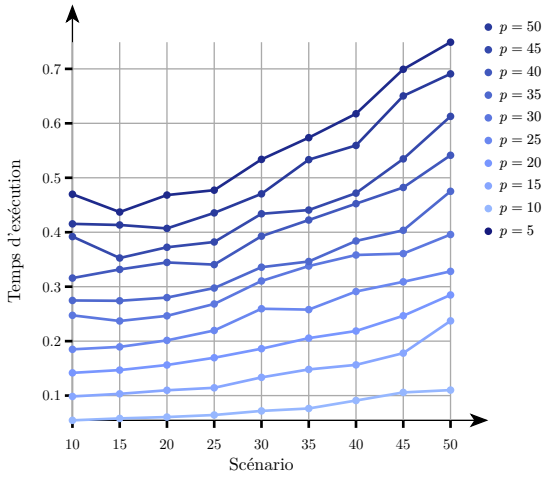

Fig. 8. – Minmax regret par projets

Partie 2

2.1. Soit le vecteur $L(z)$ défini pour tout $z \in \mathbb{R}^n$ comme $\big(L_1(z), ..., L_{n(z)}\big)$ où chaque composante $L_k(z) = \sum_{i=1}^{k} z_{(i)}$. Pour tout vecteur z, nous notons $z_{(i)}$ la i-ème composante du vecteur z trié par ordre croissant, représentant ainsi la i-ème plus faible valeur de z. D'où $L_k(z)$ représente la somme des k-ème plus faible utilités obtenues dans le problème initial.

On considère maintenant le programme linéaire suivant.

$$\min \sum_{i=1}^{n} a_{ik} z_i$$
$$s.c. \Big\{ \sum_{i=1}^{n} a_{ik} = k$$
$$a_{ik} \in \{0, 1\} \quad \forall i \in \{1, ..., n\}$$

On cherche à trouver la solution de ce programme linéaire.

On a introduit, pour représenter ce programme, une nouvelle variable a_{ik}, une variable binaire avec la seule contrainte que la somme de ces variables doit être égale à k. Comme a_{ik} est une variable binaire, la somme du min revient à sommer des termes z_i en les prenant si $a_{ik} = 1$ et en ne les prenant pas si $a_{ik} = 0$. On cherche ensuite à minimiser cette valeur.

On va maintenant montrer ce pourquoi $a_k = (a_{1k}, ..., a_{nk})$ de valeur $L_k(z)$ est la solution optimale de notre programme linéaire. Tout d'abord, montrons que a_k est bien une solution réalisable.

$L_k(z)$ est une somme de exactement k z_i, ils sont simplement triés par ordre croissant. On peut donc bien avoir $L_k(z) = a_{ik} z_i$. Cela veut simplement dire que $a_{ik} = 1$ si z_i fait parti des k plus petites valeurs de z et $a_{ik} = 0$ sinon. D'où $\sum_{i=1}^{n} a_{ik} = k$, la solution a_k est bien une résolution réalisable.

Supposons maintenant, par l'absurde, qu'il existe a'_k une solution optimale, donc de valeur inférieure à $L_k(z)$. On note I l'ensemble des indices i tels que $a'_{ik} = 1$ et J l'ensemble des indices des k plus petites composantes de z. Si $I \neq J$, et comme $|I| = |J|$, alors il existe $i_1 \in I \setminus J$ et $i_2 \in J \setminus I$. On construit alors une nouvelle solution a''_k, tel que $a''_{i_1 k} = 0$ et $a''_{i_2 k} = 1$ et $a''_{ik} = a'_{ik}$ pour toutes les autres valeurs de i. Cette solution est toujours réalisable, et de plus, par définition de j, $z_{i_2} < z_{i_1}$, donc $\sum_{i \in J} a''_{ik} z_i < \sum_{i \in I} a'_{ik} z_i$. On a donc trouvé une solution avec une valeur inférieure à celle de a'_k, donc a'_k n'est pas la solution optimale, ce qui est une contradiction.

Par conséquent, $L_k(z)$ est bien la solution optimale de notre programme linéaire.

2.2. On part du programme linéaire relaxé:

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

Pour construire le dual, on suit la méthode standard. Les contraintes du primal deviennent des variables dans le dual, les variables du primal deviennent des contraintes dans le dual et le sens de l'optimisation s'inverse. D'où le programme dual D_k suivant :

$$\max k r_k - \sum_{i=1}^{n} b_{ik}$$
$$s.c. \{ r_k - b_{ik} \leq z_i \quad \forall i \in \{1, ..., n\}$$
$$b_{ik} \geq 0 \quad \forall i \in \{1, ..., n\}, r_k \in \mathbb{R}$$

Soit $z = (2, 9, 6, 8, 5, 4)$, on cherche désormais à calculer la valeur de $L(z)$. On peut le faire de deux façons différentes, tout d'abord, en triant à la main le vecteur z, pour obtenir $z' = (2, 4, 5, 6, 8, 9)$ et calculant le vecteur cumulé, ce qui correspond à $L(z)$.

$$L_1(z) = 2$$
$$L_2(z) = 2 + 4 = 6$$
$$L_3(z) = 6 + 5 = 11$$
$$L_4(z) = 11 + 6 = 17$$
$$L_5(z) = 17 + 8 = 25$$
$$L_6(z) = 25 + 9 = 34$$

Ce qui nous permet de conclure :

$$L(z) = (2, 6, 11, 17, 25, 34)$$

On peut aussi écrire un programme linéaire pour résoudre le même dual, ce qui a été réalisé en Python (voir le fichier `src/q22.py`. La résolution nous fournit les résolutats suivants.

$$L(z) = (2, 6, 11, 17, 25, 34)$$

Les deux façons de calculer la valeur de $L(z)$ renvoie bien exactement les mêmes résultats, comme attendu.

2.3. On cherche à montrer le résultat suivant.

$$g = \sum_{k=1}^{n} w'_k L_k(z)$$
$$= \sum_{k=1}^{n-1} (w_k - w_{k+1}) L_k(z) + w_n L_n(z)$$
$$= \sum_{k=1}^{n-1} w_k L_k(z) - \sum_{k=1}^{n-1} w_{k+1} L_k(z) + w_n L_n(z)$$
$$= [w_1 L_1(z) + w_2 L_2(z) + ... + w_{n-1} L_{n-1}(z)]$$
$$-[w_2 L_1(z) + ... + w_{n-1} L_{n-2}(z) + w_n L_{n-1}(z)]$$
$$+ w_n L_n(z)$$
$$= w_1 L_1(z) + w_2 (L_2(z) - L_1(z)) + ... + w_{n-1}$$
$$(L_{n-1}(z) - L_{n-2}(z)) + w_n (L_n(z) - L_{n-1}(z))$$

Avant de continuer le calcul de cette somme, on remarque que :

$$L_1(z) = \sum_{i=1}^{1} z_{(i)} = z_{(1)}$$

On s'intéresse aussi aux termes récurrents, ce qui nous permet d'observer que :

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

D'où, si on reprend le calcul de g :

$$g = w_1 z_{(1)} + w_2 z_{(2)} + ... + w_{n-1} z_{(n-1)} + w_n z_{(n)}$$
$$= \sum_{i=1}^{n} w_i z_{(i)}$$

Ceci montre bien que:

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

2.4. On va désormais passer de la formulation du dual à une formulation finale. On sait que $g(x) =$

$\sum_{i=1}^{n} w'_k L_k(z(x))$ et que $L_k(z(x)) = \max kr_k - \sum_{i=1}^{n} b_{ik}$ s.c. On va donc pouvoir combiner ces deux résultats pour arriver à un programme linéaire.

$$g(x) = \sum_{k=1}^{n} w'_k L_k(z(x)) = \sum_{k=1}^{n} w'_k \max kr_k - \sum_{i=1}^{n} b_{ik}$$

Comme les poids w'_k sont positifs, on peut fusionner les max :

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

On obtient donc le programme linéaire suivant :

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

$$s.c. \begin{cases} r_k - b_{ik} \leq \sum_{j=1}^{p} s^i_j x_j & \forall i \in \{1, ..., n\} \\ \sum_{j=1}^{p} c_j x_j \leq B \end{cases}$$

$$b_{ik} \geq 0 \quad \forall i, k \in \{1, ..., n\}$$
$$r_k \in \mathbb{R} \quad \forall k \in \{1, ..., n\}$$
$$x_j \in \{0, 1\} \quad \forall j \in \{1, ..., p\}$$

L'implémentation de ce programme linéaire a été réalisé en Python (voir le fichier `src/q24.py`). La résolution nous fournit les résultats suivants.

Vecteur x^*: $(0, 1, 1, 1, 0, 0, 1, 1, 1, 0)$
Coût total: $85K€$
Valeur image: $z(x*) = (66, 66)$
Valeur optimale: $g(x^*) = 198$

Cette solution est identique dans notre exemple à la solution obtenue durant la partie 1.1, ce qui n'est pas surprenant car les deux critères favorisent les solutions équilibrées. On peut tout de même faire varier les valeurs des poids et observer de nouvelles solutions, ce qui démontre bien l'utilité et le développement d'un tel critère : il permet à l'utilisateur d'adapter ses perceptions par rapport à ce qu'il anticipe pour avoir un résultat plus conforme à ses attentes. On note aussi que mettre les poids à $w = (1, 0)$ correspond à ne considérer que le pire des deux scénarios et revient bien au maxmin, on observe bien ce résultat en testant le programme linéaire.

2.5. Pour le minOWA des regrets, on suit une approche similaire mais avec des modifications importantes. Comme pour le maxOWA, on commence par définir:

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

où $r\left(x, s_{(i)}\right)$ représente le i-ème plus grand regret (en ordre décroissant). On peut définir :

$$L'_k(r) = \sum_{i=1}^{k} r_{(n-i+1)}$$

qui représente la somme des k plus grands regrets. Cette fonction peut être obtenu comme solution du programme linéaire suivant :

$$L'_k(r) = \max \sum_{i=1}^{n} a_{ik}(z^*_i - z_i)$$

$$s.c. \begin{cases} \sum_{i=1}^{n} a_{ik} = k \\ a_{ik} \leq 1 \quad \forall i \in \{1, ..., n\} \end{cases}$$

$$a_{ik} \geq 0 \quad \forall i \in \{1, ..., n\}$$

Le dual de ce programme est :

$$\min kr_k + \sum_{i=1}^{n} b_{ik}$$

$$s.c. \{ r_k + b_{ik} \geq z^*_i - z_i$$
$$b_{ik} \geq 0 \quad \forall i, k \in \{1, ..., n\},$$
$$r_k \in \mathbb{R} \quad \forall k \in \{1, ..., n\}$$

Par un raisonnement similaire à celui de la question 2.4, on obtient le programme linéaire final :

$$\min \sum_{k=1}^{n} w'_k \left(kr_k + \sum_{i=1}^{n} b_{ik} \right)$$

$$s.c. \begin{cases} r_k + b_{ik} \geq z^*_i - \sum_{j=1}^{p} s^i_j x_j & \forall i \in \{1, ..., n\} \\ \sum_{j=1}^{p} c_j x_j \leq B \end{cases}$$

$$b_{ik} \geq 0 \quad \forall i, k \in \{1, ..., n\}$$
$$r_k \in \mathbb{R} \quad \forall k \in \{1, ..., n\}$$
$$x_j \in \{0, 1\} \quad \forall j \in \{1, ..., p\}$$

L'implémentation de ce programm linéaire a été réalisée en Python (voir le fichier `src/q25.py`). La résolution nous fournit les résultats suivants.

Vecteur x^*: $(0, 1, 1, 0, 0, 1, 1, 1, 1, 0)$
Coût total: $85K€$
Valeur image: $z(x*) = (50, 48)$
Valeur optimale: $g(x^*) = 148$

On note que cette solution est une fois encore la même que celle obtenue dans la partie 1.2. On peut, de la même façon qu'en 2.4, faire varier les poids pour obtenir de nouvelles solutions. Finalement, on note que prendre les poids $w = (1, 0)$ revient à faire exactement le minmax regret de la question 1.2.

même question qu'à la q 1.4

2.6. On s'intéresse maintenant à l'évolution des temps de résolution des deux critères (maxOWA et minOWA regret) en fonction du nombre de scénarios ($n = \{5, 10, 15, ..., 50\}$) et du nombre de projets ($p = \{10, 15, 20, ..., 50\}$). Pour évaluer systématiquement leurs performances respectives, nous générons pour chaque couple (n, p) un ensemble de 50 instances aléatoires. Les coûts et utilités de chaque instance sont tirés uniformément dans l'intervalle $[1, 100]$, avec un budget fixé à 50% de la somme totale des coûts des projets. Les poids sont choisis dans l'intervalle $[1, n]$ puis trié pour pouvoir respecter la conditions de poids décroissants. Cette approche nous permet d'obtenir une distribution statistiquement significative des temps de résolution pour différentes tailles de problèmes.

L'implémentation de ce programme linéaire a été réalisée en Python(voir le fichier `src/q26.py`). La résolution nous fournit les résultats suivants.

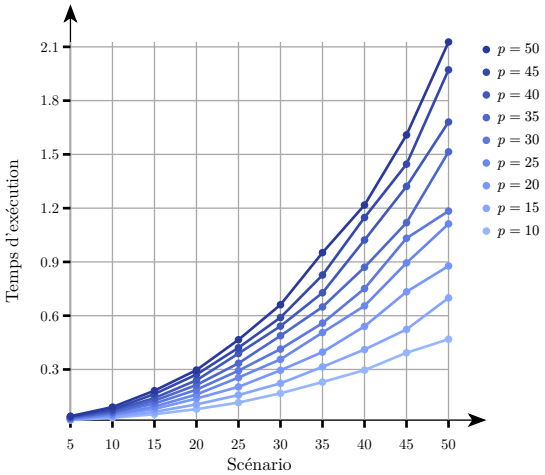

Fig. 11. – MaxOWA par scénarios

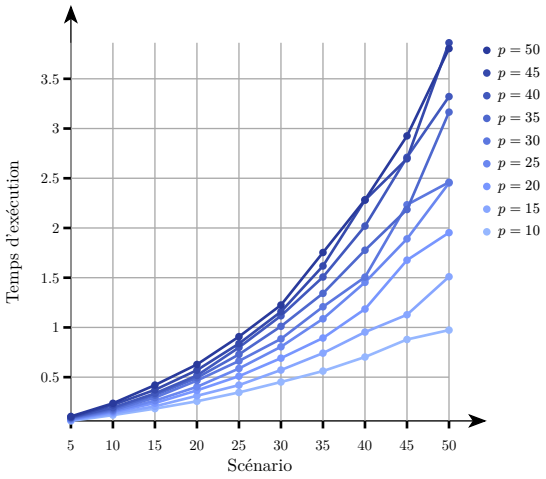

Fig. 12. – MinOWA regret par scénarios

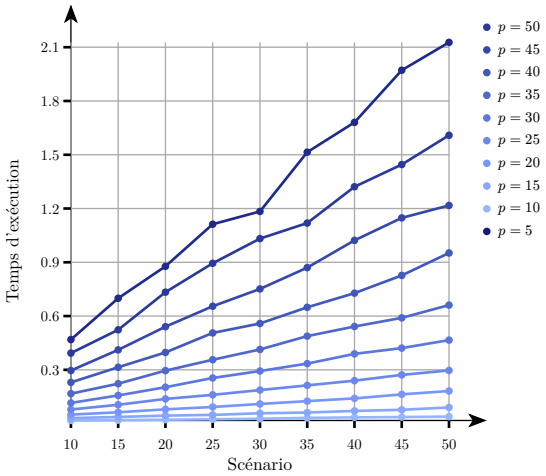

Fig. 13. – MaxOWA par projets

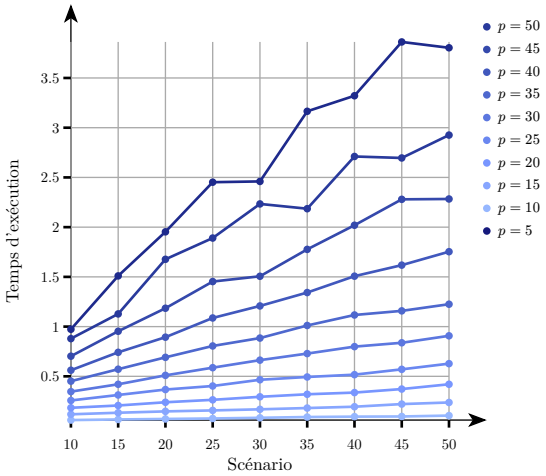

Fig. 14. – MinOWA regret par projets

Partie 3

3.1. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt

ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum.

3.2. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me.

3.3. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et.

3.4. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.