Projet Mogpl
Optimisation robuste dans l'incertain total

Partie 1

1.1. On traite la linéarisation du critère maxmin 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 solution 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 linéaire en variables mixtes, on introduit 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 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 encore 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 :

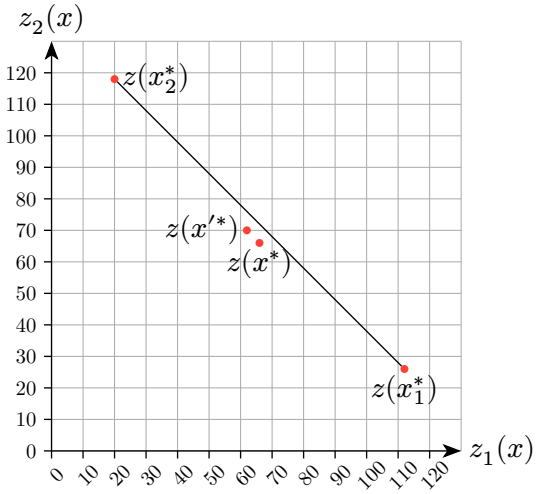

Fig. 4. – Représentation des différentes solutions dans chaque scénarios

Le segment ci-dessus 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 obtenu 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. Nous avons décidé d'augmenter la taille et le nombre de répétitions de chaque instance pour avoir des résultats plus significatif et surtout pour pouvoir mieux observer l'évolution des tendances des courbes.

L'implémentation de ce programme linéaire a été réalisée en Python (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.

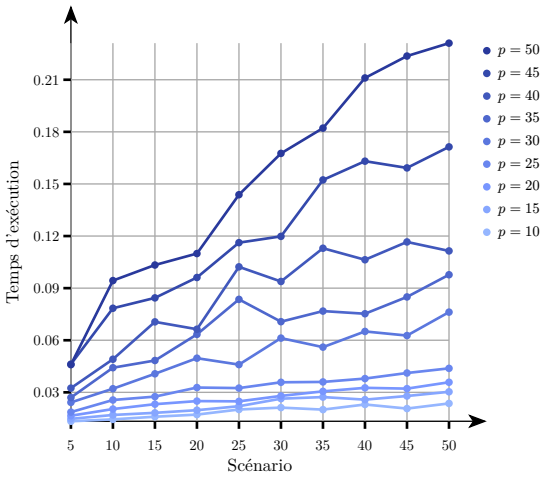

Fig. 5. – Maxmin par scénarios

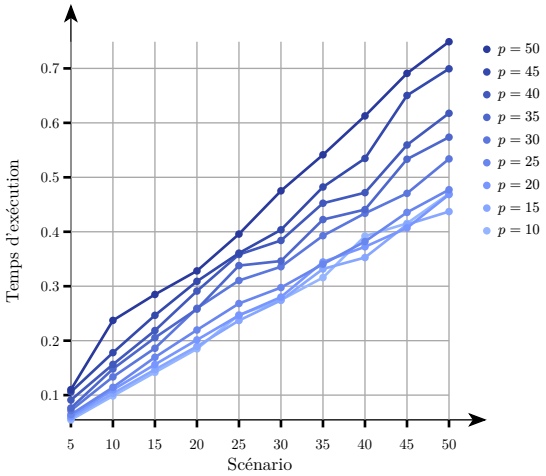

Fig. 6. – Minmax regret par scénarios

En revanche, 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 SacÀDos, 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 nom-

breux 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.

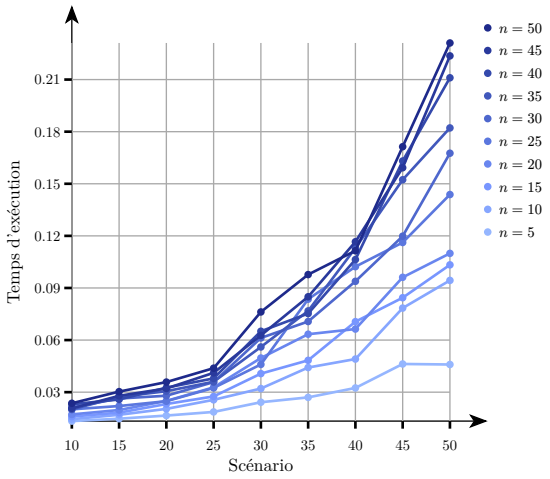

Fig. 7. – Maxmin par projets

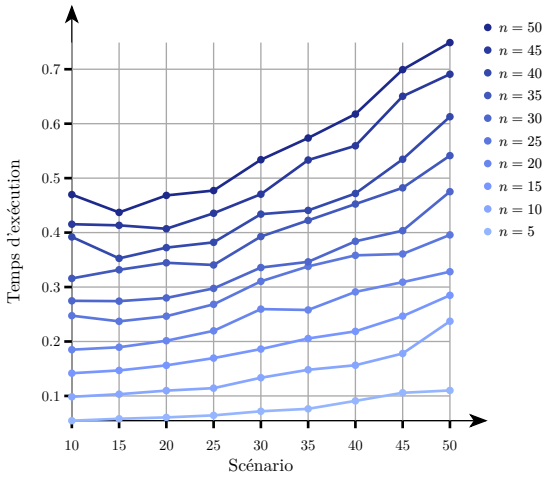

Fig. 8. – Minmax regret par projets

Partie 2

2.1. Soit le vecteur $L(z)$ défini pour tout $z \in \mathbb{R}^n$ comme $(L_1(z), ..., L_n(z))$ où, pour chaque composante, on a $L_k(z) = \sum_{i=1}^{k} z_{(i)}$. 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 faibles utilités obtenues dans le problème initial.

On considère maintenant le programme linéaire suivant.

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

On cherche à trouver la solution de ce programme linéaire.

On a introduit, pour représenter ce programme une nouvelle variable binaire a_{ik} avec pour 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 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 partie 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 $|I| = |J| = k$, 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 infé-

rieure 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 manuellement 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 résultats suivants.

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

Les deux façons de calculer la valeur de $L(z)$ renvoient 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 :

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

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

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

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 des 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 :

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

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

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

L'implémentation de ce programme linéaire a été 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 de nouveau 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.

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 $[0, n]$ 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.

On remarque tout d'abord que le critère maxOWA et minOWA regret sont plus lents en temps d'exécution. Par exemple une résolution $n = 50, p = 50$ prend $2.1s$ et $3.9s$ contre $0.23s$ et $0.8s$ pour les critère maxmin et minmax regret. Ces critères permettent donc une description plus précise des besoins de l'utilisateur mais cela a un prix en termes de performance. De plus, on remarque que l'évolution de la complexité qui étaient précédemment linéaire ne l'est plus. L'évolution est désormais devenue quadratique. On émet l'hypothèse que cela est dû au fait que le critères de type OWA introduisent un tri. La complexité naïve d'un tri est de l'ordre $O(n^2)$. Cela explique probablement le passage d'une complexité linéaire à une complexité quadratique sur l'évolution du nombre de projet.

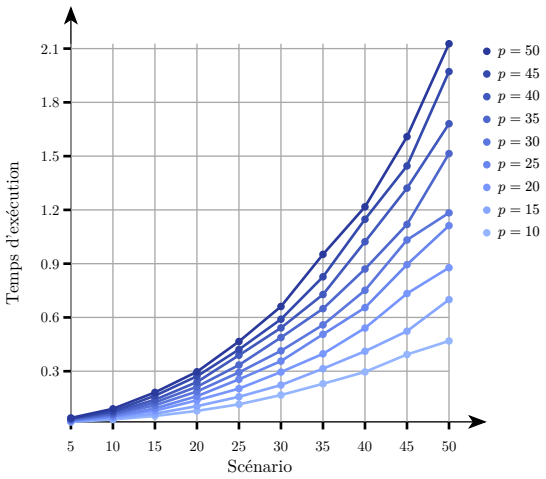

Fig. 11. – MaxOWA par scénarios

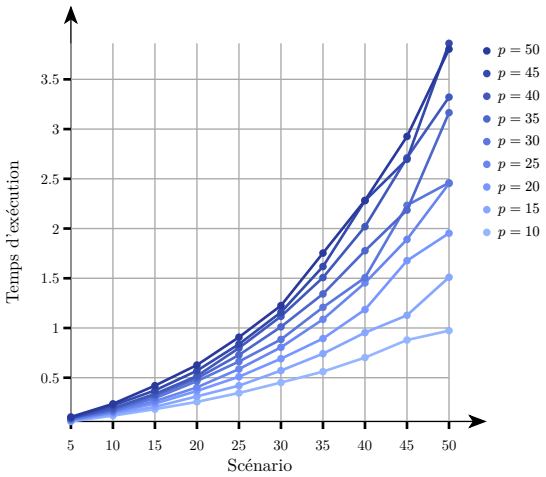

Fig. 12. – MinOWA regret par scénarios

On remarque également que les résolutions par projets semblent être devenues linéaires, alors qu'elles étaient en temps exponentiel. Cela est particulièrement étonnant, compte-tenu du fait que, comme on l'a vu en partie 1, ce problème est une instance du problème du SacÀDos, qui est un problème NP-complet. On emet l'hypothèse que ces courbes sont en fait bien exponentielles, mais que l'on n'a simplement pas encore pu voir l'évolution exponentielle, dû au fait que l'on observe que des instances de $[10, 50]$. On retrouvera

bien ce résultat sur des plus grosses instances, dans la partie 3.

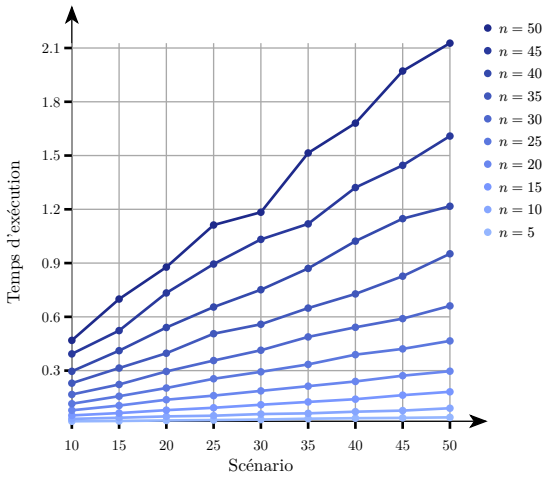

Fig. 13. – MaxOWA par projets

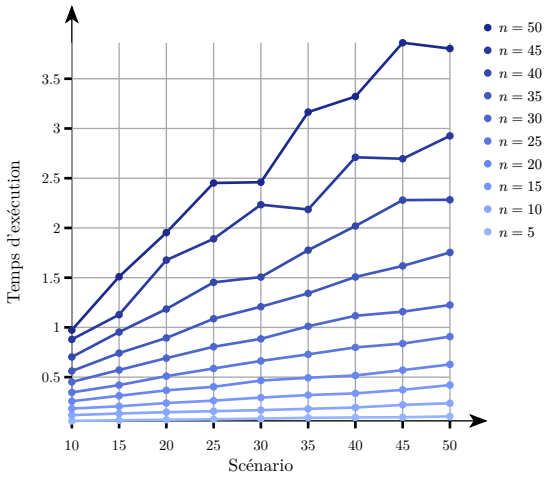

Fig. 14. – MinOWA regret par projets

Partie 3

3.1. On cherche à représenter le problème de recherche du chemin le plus rapide entre les sommets source et destination dans un graphe comme un problème linéaire. Dans ce programme linéaire, la fonction objective que l'on cherche à minimiser est la longueur du chemin. On propose de représenter le graphe sous la forme d'une matrice d'adjacence, en introduisant les variables binaires x_{ij} qui représentent si on prend ou non l'arc (i, j). L'ensemble des $x_{ij} = 1$ ainsi que les sommets s et t peuvent alors former un chemin. On peut représenter la longueur de ce chemin, avec s le scénario et p le nombre de nœuds présents dans le graphe par :

$$\sum_{i=0}^{n}\sum_{j=0}^{n} t_{ij}^{s} x_{ij}$$

Soit X l'ensemble des solutions admissibles, nous devons alors ajouter la contrainte $x \in X$, c'est à dire que les $x_{ij} = 1$, c'est à dire les arcs effectivement sélectionnés, doivent représenter un chemin effectivement existant dans le graphe. Pour trouver des contraintes linéaires qui modélisent correctement ce besoin, on transforme le problème en un problème de flot maximal. En particulier, on réfléchit à ce que l'ajout de capacité de 1 à tous les arcs existants permet. Si la source a un flot entrant, et la destination un flot sortant infini, alors de nœuds à nœuds, le flot va se propager dans différents chemins. Dans notre cas, on cherche cependant un unique chemin, pour ce faire, on peut fixer le flot entrant dans la source et le flot sortant de la source à 1. Le flot va alors partir de la source et se propager dans le graphe jusqu'à atteindre la destination (si c'est possible). Les arcs dont les flots est 1 vont alors correspondre à un chemin dans le graphe. Du fait, par analogie au problème de flot, on arrive au fait que les x_{ij} peuvent aussi être vus comme la valeur des flots dans le graphe. De plus, le problème de flots se modélise avec les contraintes linéaires suivantes. On veut tout d'abord que la somme des flots sortants de la source et la somme des flots entrants dans la destination valent 1. De plus, on souhaite avoir la contrainte de conservation de flot, c'est à dire que pour tout nœud, la somme de ses flots entrant soit égale à la somme de ses flots sortants.

$$\sum_{i=0}^{p} x_{si} = 1$$

$$\sum_{i=0}^{p} x_{it} = 1$$

$$\sum_{i=0}^{p} x_{iv} - x_{vi} = 0 \quad \forall v \neq s \text{ et } v \neq t$$

Ces observations nous amènent aux contraintes suivantes qui sont effectivement linéaires. On note aussi que l'on devrait aussi ajouter la contraintes $x_{ij} \leq 1$, pour introduire le fait que

les capacités des arcs valent toutes 1, mais cette information est déjà contenue dans le fait que les x_{ij} sont des variables binaires.

Ce qui nous donne le programme linéaire suivant :

$$\min \sum_{i=0}^{n} \sum_{j=0}^{n} t_{ij}^{s} x_{ij}$$

$$s.c. \begin{cases} \sum_{i=0}^{p} x_{si} = 1 \\ \sum_{i=0}^{p} x_{it} = 1 \\ \sum_{i=0}^{p} x_{iv} - x_{vi} = 0 \quad \forall v \neq s \text{ et } v \neq t \end{cases}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in \{1, ..., p\}$$

Un point final sur lequel il nous faut commenter est la structure de la matrice d'adjacence. Il est commun de représenter, dans les matrices d'adjacences, les arcs inexistants par des 0. Réfléchissons à l'impact de cette décision sur le programme linéaire ci-dessus. Si l'on cherche à minimiser, alors pour prendre un exemple minimal simple. S'il n'y a pas d'arc entre la source et la destination, alors la solution qui contient uniquement $x_{st} = 1$ et $x_{ij} = 0 \forall i \neq s, j \neq t$ est bien une solution admissible, dans le sens où tous les critères du programme linéaire sont respectés, et la valeur de sa fonction objective est 0. Le programme linéaire peut donc renvoyer au moins cette solution, alors que, puisqu'un arc n'existe pas entre ces deux points, cela ne devrait pas être le cas. On pourrait rajouter une autre contrainte, mais une autre solution plus simple est d'introduire un pré traitement à la matrice d'adjacence. Soit M grand, si on remplace toutes les valeurs 0 dans la matrice, c'est à dire les arcs inexistants, par M, alors dans le précédent scénario, la valeur de la fonction objective sera M. Si M est effectivement supérieure à la longueur du plus court chemin, alors le programme linéaire trouvera une meilleure solution. On introduit donc ce pré traitement à la matrice avant le programme linéaire afin de pénaliser les arcs inexistants.

3.2. L'implémentation de ce programme linéaire a été réalisé en Python (voir le fichier `src/q32.py`). La résolution nous fournit les résultats suivants.

On observe bien qu'en pratique, la pénalisation choisie précédemment s'avère utile car comme prévu la minimisation favorise les arcs inexistants.

Matrice x^*: $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Arcs sélectionnés : $\{(a, b), (b, d), (d, f)\}$

Valeur optimale: $g(x^*) = 8$

Fig. 15. – Instance gauche, scénario 1

Matrice x^*: $\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Arcs sélectionnés : $\{(a, c), (c, d), (d, f)\}$

Valeur optimale: $g(x^*) = 4$

Fig. 16. – Instance gauche, scénario 2

Matrice x^*: $\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Arcs sélectionnés : $\{(a, d), (d, c), (c, f), (f, g)\}$

Valeur optimale: $g(x^*) = 5$

Fig. 17. – Instance droite, scénario 1

Matrice x^*: $\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Arcs sélectionnés : $\{(a, c), (c, e), (e, g)\}$

Valeur optimale: $g(x^*) = 6$

Fig. 18. – Instance droite, scénario 2

On a donné ici les valeurs des matrices x^*, cependant, par souci de simplicité visuelle, on ne présentera, dans le reste de ce rapport, que l'ensemble des arcs sélectionnés. La valeur de

la matrice peut être obtenue directement par l'exécution du programme.

3.3. On cherche à généraliser les observations que l'on a fait en partie 1 et 2 pour l'appliquer à la partie 3 sur ce nouveau problème. En fait, le problème de la partie 1 dans un scénario s donné, peut-être décrit de la façon minimale suivante.

$$\begin{array}{ccc} \max z_s(x) & \Leftrightarrow & \max \sum_{i=0}^{p} s_i^s x_i \\ s.c. \{ x \in X & \Leftrightarrow & s.c. \left\{ \sum_{i=0}^{p} c_i x_i \right. \end{array}$$

Cette description est minimale, mais les quatre critères que l'on a vus ne dépendent pas de l'implémentation spécifique de la fonction objective et des contraintes d'admissibilité.

On a tout d'abord maxmin et minmax regret, pour lesquels on a simplement à introduire une nouvelle variable et ajouter une contrainte qui utilise la fonction objective et les optimaux dans le cas de minmax regret.

$$\max \alpha$$
$$s.c \begin{cases} \alpha \leq z_s(x) & \forall s \in \{1, ..., n\} \\ x \in X \end{cases}$$
$$x \in \{0,1\}^p$$
$$\alpha \in \mathbb{R}$$

$$\min \beta$$
$$s.c \begin{cases} \beta \geq z_s^* - z_s(x) & \forall s \in \{1, ..., n\} \\ x \in X \end{cases}$$
$$x \in \{0,1\}^p$$
$$\beta \in \mathbb{R}$$

Pour les critères maxOWA et minOWA regret, la description des programmes est plus longue, mais une fois de plus on n'utilise que la fonction objective et les contraintes d'admissibilité. De plus, dans la partie 2, nous n'avons pas utilisé la structure du problème de SacÀDos directement, donc les preuves que nous avons données sont toujours vraies peu importe le détail du problème. De façon plus générale, si un problème peut être modélisé par un programme linéaire de maximisation avec des variables binaires, dans un scénario, alors on peut utiliser ces formulations génériques pour orienter une prise de décision dans l'incertain complet sur les critères.

$$\max \sum_{k=1}^{n} w'_k \left(k r_k - \sum_{s=1}^{n} b_{sk} \right)$$
$$s.c \begin{cases} r_k - b_{sk} \leq z_s(x) & \forall s \in \{1, ..., n\} \\ x \in X \end{cases}$$
$$x \in \{0,1\}^p$$
$$b_{sk} \geq 0 \quad \forall s, k \in \{1, ..., n\}$$
$$r_k \in \mathbb{R} \quad \forall k \in \{1, ..., n\}$$

$$\min \sum_{k=1}^{n} w'_k \left(k r_k + \sum_{s=1}^{n} b_{sk} \right)$$
$$s.c \begin{cases} r_k - b_{sk} \geq z_s^* - z_s(x) & \forall s \in \{1, ..., n\} \\ x \in X \end{cases}$$
$$x \in \{0,1\}^p$$
$$b_{sk} \geq 0 \quad \forall s, k \in \{1, ..., n\}$$
$$r_k \in \mathbb{R} \quad \forall k \in \{1, ..., n\}$$

On note p la dimension du vecteur binaire x. Dans la partie 1 et 2, on avait une dimension p (avec p le nombre de projets), dans la partie 3, on va voir une dimension $p \times p$ (avec p le nombre de nœuds dans le graphe). Cette dimension, la fonction objective et les contraintes d'admissibilité décrivent alors entièrement un problème et peuvent être utilisées pour calculer les quatre critères dans l'incertain complet.

C'est pourquoi on choisit de réécrire le problème de recherche de plus court chemin dans un graphe de la façon suivante :

$$\max z_s(x) = -\sum_{i=0}^{n} \sum_{j=0}^{n} t_{ij}^s x_{ij}$$
$$s.c. \begin{cases} \sum_{i=0}^{p} x_{si} = 1 \\ \sum_{i=0}^{p} x_{it} = 1 \\ \sum_{i=0}^{p} x_{iv} - x_{vi} = 0 \quad \forall v \neq s \text{ et } v \neq t \end{cases}$$
$$x_{ij} \in \{0,1\} \quad \forall i, j \in \{1, ..., p\}$$

On cherche à obtenir un problème de maximisation, ce qui est nécessaire pour appliquer les

critères de façon générique. On observe ici que ce résultat reste simple à conserver si on passe d'une minimisation à une maximisation et qu'on inverse le signe de la fonction objective. On aura une valeur opposée à la fonction objective précédente, mais on choisira bien les mêmes valeurs du vecteur binaire x. Dans notre exemple, cela correspond à avoir des chemins de coûts négatifs (que l'on peut trivialement rendre positifs pour avoir le vrai coût du chemin) et le même ensemble d'arcs qui caractérise le chemin.

On va désormais pouvoir appliquer nos critères sur le problème de recherche de chemin le plus court dans l'incertain complet.

$$\max \alpha$$
$$s.c \begin{cases} \alpha \leq -\sum_{i=0}^{n} \sum_{j=0}^{n} t_{ij}^{s} x_{ij} \quad \forall s \in \{1, ..., n\} \\ \sum_{i=0}^{p} x_{si} = 1 \\ \sum_{i=0}^{p} x_{it} = 1 \\ \sum_{i=0}^{p} x_{iv} - x_{vi} = 0 \quad \forall v \neq s \text{ et } v \neq t \end{cases}$$
$$x_{ij} \in \{0, 1\} \quad \forall i, j \in \{1, ..., p\}$$
$$\alpha \in \mathbb{R}$$

Pour le problème maxmin dans le cas de la recherche du chemin de plus faible coût, on obtient la formulation suivante. L'implémentation de ce programme linéaire a été réalisée en Python (voir le fichier `src/q33.py`). La solution nous fournit les résultats suivants.

Arcs sélectionnés : $\{(a, b), (b, d), (d, f)\}$
Vecteur image $z(x^*) = (-8, -9)$
Valeur optimale: $g(x^*) = -9$

Arcs sélectionnés : $\{(a, b), (b, e), (e, g)\}$
Vecteur image $z(x^*) = (-10, -10)$
Valeur optimale: $g(x^*) = -10$

On observe que la valeur optimale a une valeur négative mais on peut simplement prendre son opposée pour obtenir le coût réel du chemin. Ici, le critère maxmin nous informe que dans les deux graphes, si on est le plus pessimiste possible sur les scénarios, alors on aura les valeurs obtenues dans le vecteur image comme potentielle action. Dans le pire cas, on sélectionne la plus petite valeur des deux (c'est à dire le plus long chemin) qui correspond à être dans le mauvais scénario. Finalement, on remarque que le vecteur est plutôt équilibré, ce qui reflète nos observations de la partie 1.

$$\min \beta$$
$$s.c \begin{cases} \beta \geq z_s^* + \sum_{i=0}^{n} \sum_{j=0}^{n} t_{ij}^{s} x_{ij} \quad \forall s \in \{1, ..., n\} \\ \sum_{i=0}^{p} x_{si} = 1 \\ \sum_{i=0}^{p} x_{it} = 1 \\ \sum_{i=0}^{p} x_{iv} - x_{vi} = 0 \quad \forall v \neq s \text{ et } v \neq t \end{cases}$$
$$x_{ij} \in \{0, 1\} \quad \forall i, j \in \{1, ..., p\}$$
$$\beta \in \mathbb{R}$$

Pour le problème minmax regret dans le cas de la recherche du chemin de plus faible coût, on obtient la formulation suivante. L'implémentation de ce programme linéaire a été réalisée en Python (voir le fichier `src/q33.py`). La solution nous fournit les résultats suivants.

Arcs sélectionnés : $\{(a, b), (b, e), (e, f)\}$
Vecteur image $z(x^*) = (3, 3)$
Valeur optimale: $g(x^*) = 3$

Arcs sélectionnés : $\{(a, b), (b, e), (e, g)\}$
Vecteur image $z(x^*) = (5, 4)$
Valeur optimale: $g(x^*) = 5$

Cette fois-ci, les regrets ont une valeur positive, ce n'est pas le coût d'un chemin, mais bien un regret de prendre une décision. Dans le meilleur cas, on prend bien le plus grand regret parmi ceux du vecteur image.

$$\max \sum_{k=1}^{n} w_k' \left(k r_k - \sum_{s=1}^{n} b_{sk} \right)$$
$$s.c \begin{cases} r_k - b_{sk} \leq -\sum_{i=0}^{n} \sum_{j=0}^{n} t_{ij}^{s} x_{ij} \quad \forall s \in \{1, ..., n\} \\ \sum_{i=0}^{p} x_{si} = 1 \\ \sum_{i=0}^{p} x_{it} = 1 \\ \sum_{i=0}^{p} x_{iv} - x_{vi} = 0 \quad \forall v \neq s \text{ et } v \neq t \end{cases}$$
$$x_{ij} \in \{0, 1\} \quad \forall i, j \in \{1, ..., p\}$$
$$b_{sk} \geq 0 \quad \forall s, k \in \{1, ..., n\}$$
$$r_k \in \mathbb{R} \quad \forall k \in \{1, ..., n\}$$

Pour le problème maxOWA dans le cas de la recherche du chemin de plus faible coût, on obtient la formulation suivante. L'implémentation de ce programme linéaire a été réalisée en Python (voir le fichier `src/q33.py`). La solution nous fournit les résultats suivants avec le vecteur poids $w = (2, 1)$.

Arcs sélectionnés : $\{(a, b), (b, d), (d, f)\}$
Vecteur image $z(x^*) = (-8, -9)$
Valeur optimale: $g(x^*) = -26$

Arcs sélectionnés : $\{(a, d), (d, f), (f, g)\}$
Vecteur image $z(x^*) = (-6, -12)$
Valeur optimale: $g(x^*) = -30$

Comme on l'a vu, l'introduction des poids dans maxOWA permet plus de nuance face au pessimisme de maxmin. Ici on n'anticipe pas le pire scénario uniquement, mais on caractérise l'importance qu'on porte à chaque scénario, du pire au meilleur dans notre décision. L'idée est de se prémunir du risque de façon moins pessimiste que maxmin. Les valeurs optimales ici ne correspondent bien pas non plus à la longueur des chemins, et c'est donc bien la valeur du chemin qui nous intéresse.

$$\min \sum_{k=1}^{n} w'_k \left(k r_k + \sum_{s=1}^{n} b_{sk} \right)$$

$$s.c \begin{cases} r_k - b_{sk} \geq z^*_s + \sum_{i=0}^{n} \sum_{j=0}^{n} t^s_{ij} x_{ij} \forall s \in \{1, ..., n\} \\ \sum_{i=0}^{p} x_{si} = 1 \\ \sum_{i=0}^{p} x_{it} = 1 \\ \sum_{i=0}^{p} x_{iv} - x_{vi} = 0 \quad \forall v \neq s \text{ et } v \neq t \end{cases}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in \{1, ..., p\}$$

$$b_{sk} \geq 0 \quad \forall s, k \in \{1, ..., n\}$$

$$r_k \in \mathbb{R} \quad \forall k \in \{1, ..., n\}$$

Pour le problème minOWA regret dans le cas de la recherche du chemin de plus faible coût, on obtient la formulation suivante. L'implémentation de ce programme linéaire a été réalisée en Python (voir le fichier `src/q33.py`). La solution nous fournit les résultats suivants avec le vecteur poids $w = (2, 1)$.

Arcs sélectionnés : $\{(a, b), (b, e), (e, f)\}$
Vecteur image $z(x^*) = (3, 3)$
Valeur optimale: $g(x^*) = 9$

Arcs sélectionnés : $\{(a, d), (d, f), (f, g)\}$
Vecteur image $z(x^*) = (1, 6)$
Valeur optimale: $g(x^*) = 13$

Finalement, le critère minOWA regret combine les idées précédentes pour obtenir un critère qui minimise le regret tout en étant pas aussi pessimiste que minmax regret en prenant en compte dans son analyse les scénarios plus optimistes, mais avec un poids plus faible.

Pour le moment, on a utilisé le vecteur de poids $w = (2, 1)$. On s'intéresse désormais à l'évolution de la solution si on choisi d'utiliser un vecteur $w = (k, 1)$ avec $k \in \{2, 4, 8, 16\}$. Tout d'abord on réfléchit de façon théorique à ce à quoi on devrait s'attendre. Dans les critère maxOWA et minOWA regret, les poids correspondent à l'importance qu'on donne à chaque scénario une fois trié du pire au meilleur (ou du meilleur au pire pour les regrets). Dans le cas de deux scénarios, si on a $k = 1$, alors cela revient à dire que l'on considère les deux scénarios, le meilleur et le pire, comme aussi importants pour la prise de décision. C'est en quelque sorte faire une sorte de moyenne dans notre décision. Mais de façon plus intéressante, si on monte k, alors dans le cas de deux scénarios, cela revient à dire qu'on donne une importance de plus en plus grande au pire scénario, quitte à négliger le meilleur. En fait, cela revient à se ramener au critère maxmin et minmax, qui est très pessimiste et ne considère que le pire. On émet donc la conjecture que à mesure que k grandit, maxOWA converge vers maxmin et minOWA regret converge vers minmax regret. C'est à dire que pour un graphe donné G, on peut trouver N tel que $\forall k > N$, la solution renvoyée par le programme linéaire maxOWA (minOWA regret) avec les poids $(k, 1)$ est exactement la même que la solution du programme linéaire maxmin (minmax regret).

Graphe de gauche

Maxmin: $\{(a,b),(b,d),(d,f)\}$
MaxOWA:
$k=2$: $\{(a,b),(b,d),(d,f)\}$
$k=4$: $\{(a,b),(b,d),(d,f)\}$
$k=8$: $\{(a,b),(b,d),(d,f)\}$
$k=16$: $\{(a,b),(b,d),(d,f)\}$

Minmax regret: $\{(a,b),(b,e),(e,f)\}$
MinOWA regret:
$k=2$: $\{(a,b),(b,e),(e,f)\}$
$k=4$: $\{(a,b),(b,e),(e,f)\}$
$k=8$: $\{(a,b),(b,e),(e,f)\}$
$k=16$: $\{(a,b),(b,e),(e,f)\}$

Graphe de droite

Maxmin: $\{(a,b),(b,e),(e,g)\}$
MaxOWA:
$k=2$: $\{(a,d),(d,f),(f,g)\}$
$k=4$: $\{(a,b),(b,e),(e,g)\}$
$k=8$: $\{(a,b),(b,e),(e,g)\}$
$k=16$: $\{(a,b),(b,e),(e,g)\}$

Minmax regret: $\{(a,b),(b,e),(e,g)\}$
MinOWA regret:
$k=2$: $\{(a,d),(d,f),(f,g)\}$
$k=4$: $\{(a,b),(b,e),(e,g)\}$
$k=8$: $\{(a,b),(b,e),(e,g)\}$
$k=16$: $\{(a,b),(b,e),(e,g)\}$

Fig. 27. – Évolution des chemins par k

On observe bien en pratique sur nos graphes les résultats attendus théoriquement.

3.4. On s'intéresse finalement à l'évolution des temps de résolution de tous les critères reformulés en fonction du nombre de scénarios ($n = \{2,5,10,...,50\}$) et du nombre de projets ($p = \{10,15,20,40,80,120,160,200,240\}$). Pour évaluer systématiquement leurs performances respectives, on génère pour chaque couple (n,p) un ensemble de 50 instances aléatoires.

Pour chaque instance, on doit générer des graphes aléatoires avec une densité entre 30% et 50%. Pour ce faire, on engendre un vecteur contenant l'ensemble arcs (i,j). On mélange au hasard ce vecteur et on tire un nombre au hasard a entre 30% et 50% de la taille de ce vecteur. On peut ensuite prendre les a premiers éléments de ce vecteur pour obtenir nos arcs. On réalise le même procédé, mais sur les sommets pour réaliser le tirage aléatoire de la source et de la destination. Les coûts des arcs dans chaque scénarios sont tirés uniformément dans l'intervalle $[1,100]$. Les poids sont choisis dans l'intervalle $[0,n]$ puis triés pour pouvoir respecter la conditions de poids décroissants.

L'implémentation de ce programme linéaire a été réalisée en Python (voir le fichier `src/q34.py`). La résolution nous fournit les résultats suivants.

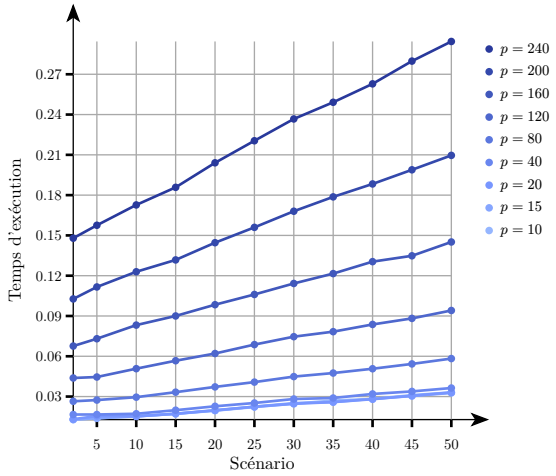

Fig. 28. – Maxmin par scénarios

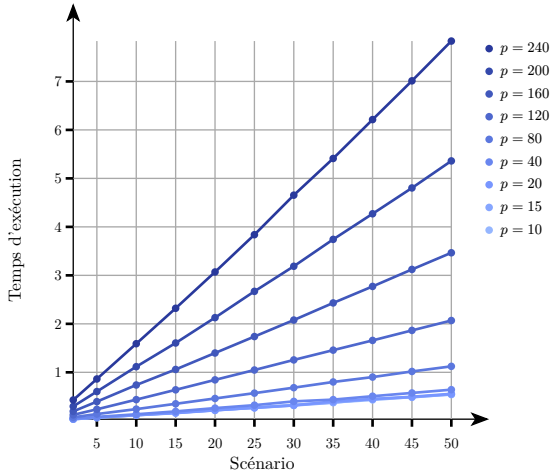

Fig. 29. – Minmax regret par scénarios

Tout d'abord commençons par commenter l'évolution des scénarios pour les critères maxmin et minmax regret. On observe que, comme dans la partie 1, considérer des nouveaux scénarios est peu cher, puisque l'évolution de la com-

plexité suit une courbe linéaire. Comme dans le première partie, cette propriété est intéressante, puisqu'elle nous permet de pouvoir considérer un grand nombre de scénarios à moindre coût.

Pour l'évolution des scénarios pour les critères maxOWA et minOWA regret, on observe une fois de plus des courbes qui semblent être quadratiques. Cela semble être aussi du au fait que le critère requiert un tri, qui suit une complexité quadratique.

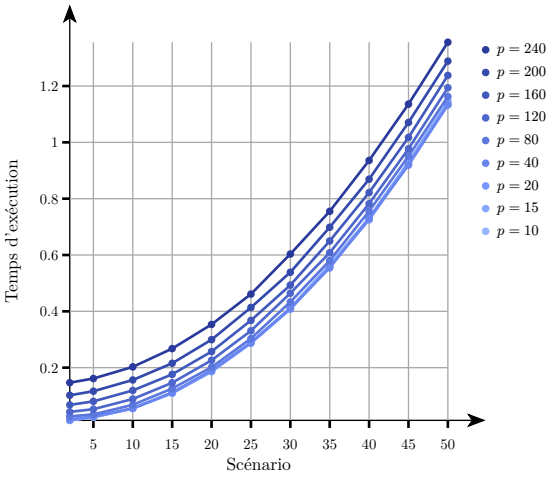

Fig. 30. – MaxOWA par scénarios

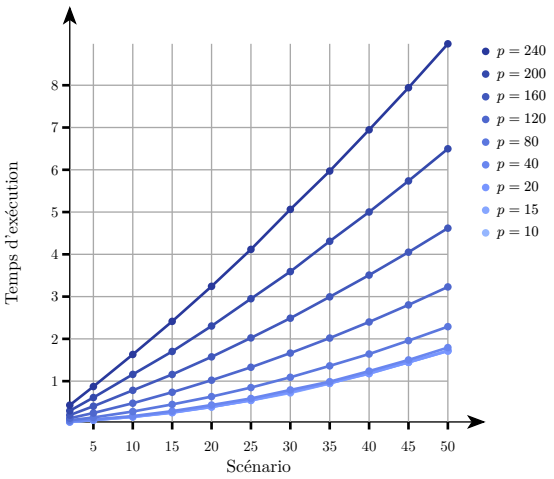

Fig. 31. – MinOWA regret par scénarios

Finalement, on s'intéresse à l'évolution de la courbe en fonction du nombre de nœuds pour les quatre critères. On remarque que la courbe a l'air d'être une fois de plus exponentielle.

Il est important de commenter que nous avons fait le choix de tirer jusqu'à des instances de taille $p = 240$ parce qu'après avoir fait des observations jusqu'à $p = 50$, la courbe était pratiquement constante. La recherche d'un plus court chemin dans un graphe de taille $p = 10$ étant évidement plus simple que dans un graphe de taille $p = 50$, on a eu l'intuition que ce plateau était probablement simplement le début d'une courbe exponentielle. C'est pourquoi nous avons pris la décision d'observer sur de plus grandes instances, et qu'on a pu valider le fait que la courbe suit bien une évolution exponentielle.

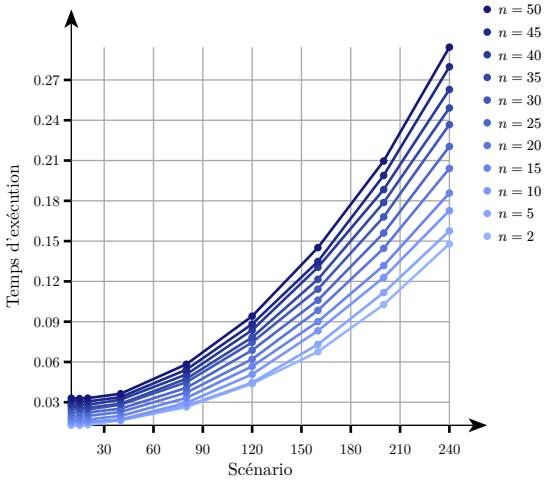

Fig. 32. – Maxmin par nœuds

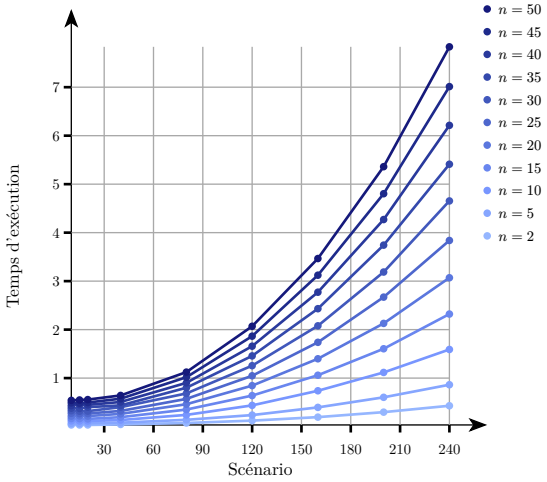

Fig. 33. – Minmax regret par nœuds

Cela étant dit, la recherche de plus court chemin dans un graphe n'est pas NP-complet. On avait justifié dans la partie 1 que cette évolution était exponentielle dû au fait que SacÀDos était NP-complet, mais cela ne le justifie plus le fait que la courbe soit exponentielle ici. Il est alors intéressant de se demander si la courbe est effectivement

exponentielle ou bien si elle n'est en fait que quadratique et que la taille réduite des tirages nous porte à croire à tort qu'elle l'est. Cela étant dit, si l'on s'intéresse au variables que l'on utilise, alors on remarque que les variables sont binaires. Cela nous renvoie à un résultat de théorie de la complexité, celui que le problème de décision 0-1IntegerProgramming est NP-complet. De fait, par le choix du domaine de définition des variables, on a contraint une garantie exponentielle sur l'évolution de la complexité. Il est particulièrement intéressant de se demander si on aurait pu éviter cette complexité en utilisant des variables continues, en introduisant simplement une des variables définies sur $\mathbb{R}^+$ et en ajoutant la contrainte $x_{ij} \leq 1$, c'est à dire d'introduire la contrainte du flot de façon explicite. Il faut noter qu'une solution entière aurait alors été possible (si un chemin existe réellement). En effet le théorème d'intégralité sur les flots maximums nous dit que si les capacités sont entières (ce qui est le cas), alors il existe un flot maximum entier. Cette nouvelle opération de benchmarking n'a pas pu être faite pour ce projet, mais on aurait probablement obtenu un résultat différent et intéressant.

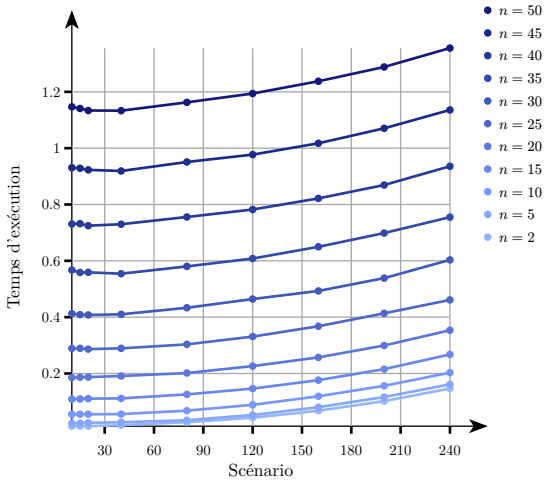

Fig. 34. – MaxOWA par nœuds

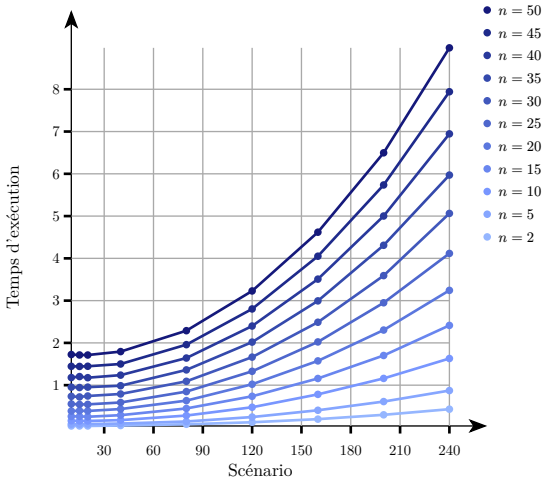

Fig. 35. – MinOWA regret par nœuds

On a pu voir dans ce projet que si un problème peut être modélisé comme un programme linéaire à variables binaires dans un scénario donné, alors on peut le passer à l'incertain complet, dans lequel on ne sait pas quel scénario il faut privilégier. Cette propriété est particulièrement intéressante pour des applications industrielles, on a pu voir comment le problème théorique du SacÀDos peut être perçu comme un problème d'investissement pour une entreprise. Pour le problème des graphes de la partie 3, on peut imaginer une application industrielle de système GPS, avec la prise en compte d'aléas météo ou de ralentissement de la circulation. Sans savoir ce qui va se passer, lorsque les modèles statistiques sont dépassés pour prévoir la probabilité des scénarios, on peut quand même utiliser ces critères pour prendre des décisions bien plus robustes qu'une simple pondération des solutions obtenues dans chaque scénario. On peut aussi imaginer une application du fameux problème de voyageur du commerce, important dans les réseaux télécoms, qui peut être alors étendu d'un scénario à un ensemble de scénarios incertains. La décision est alors capable de s'y adapter et de proposer des solutions plus robuste.

Dans la vie courante, on peut facilement trouver de nombreux exemples de problèmes où les modèles statistiques ne permettent pas une conclusion satisfaisante, comme les élections en politique, les fusions acquisitions d'entreprises

ou le résultat d'une décision de justice. Dans ce type d'évènements, qui peuvent avoir un impact très important sur le déroulé d'opérations industrielles, on peut, grâce à ces critères, prendre des décisions bien plus robustes, et en particulier bien plus robustes dans les pire cas.

Finalement, on aimerait pouvoir combiner l'aspect robuste de la décision dans l'incertain et de la prémunition face au risque et l'aspect probabiliste. En pratique, on a quand même parfois des informations qui permettent d'orienter notre jugement dans un sens ou dans l'autre. Il serait particulièrement intéressant d'introduire un aspect probabiliste à ces critères, en particulier pour minOWA regret qui est le critère le plus complexe que l'on a vu dans ce projet.