Sorbonne Université le 18 Octobre 2024
MASTER ANDROIDE M1 – MOGPL

Projet MOGPL : optimisation robuste dans l'incertain total

Introduction

Les problèmes d'optimisation combinatoires classiques se formulent comme la maximisation ou la minimisation d'une fonction objectif f sur un domaine combinatoire X de solutions défini implicitement, par exemple par un système de contraintes à satisfaire en programmation linéaire en nombres entiers ou par une propriété structurelle requise (chemin, arbre, affectation, cycle hamiltonien, etc) dans les graphes. La fonction objectif f attribue une valeur à toute solution réalisable et permet de les comparer pour déterminer la ou les solutions optimales. Cependant dans certains contextes cette évaluation est incertaine et dépend de scénarios sur l'environnement du problème susceptibles d'impacter différemment les évaluations. Si l'on note S l'ensemble des scénarios possibles, l'évaluation de x dans le scénario $s \in S$ sera notée $f(x, s)$. Dans le cas où l'ensemble S des scénarios considérés est fini et contient n éléments $\{s_1, \ldots, s_n\}$, chaque solution réalisable $x \in X$ est alors caractérisée par un vecteur de conséquences possibles $z(x) = (z_1(x), \ldots, z_n(x))$ où $z_i(x) = f(x, s_i)$ représente l'évaluation de la solution x dans le scénario s_i. Lorqu'on ne dispose d'aucune information sur la probabilité des scénarios considérés on dit qu'on est dans l'incertain total. Il faut néanmoins trouver une solution la plus intéressante possible dans le contexte incertain ainsi défini. A titre d'illustration considérons le problème de sac-à-dos suivant :

Exemple 1 *On considère un problème de sélection de projets parmi une liste de 10 projets disponibles, sachant qu'on dispose d'une enveloppe budgétaire de 100 Keuros. Le coût de chaque projet est connu de manière certaine mais l'utilité du projet varie selon le scénario considéré. On distingue en effet ici 2 scénarios s_1, s_2 concernant l'évolution de l'environment socio-économique. L'utilité des 10 projets dans ces 2 scénarios ainsi que les coûts des projets sont donnés dans le tableau ci-dessous et on cherche le meilleur sous-ensemble de projets à sélectionner en tenant compte des 2 scénarios (les utilités sont supposées additives sur les projets).*

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

Dans une telle situation, plutôt que de parier sur un scénario particulier et de faire de l'optimisation classique, l'optimisation robuste consiste à rechercher une solution qui reste attractive dans la plupart des scénarios, voire dans tous les scénarios. Plus généralement on optimise en accordant une importance prépondérante aux mauvais scénarios pour rechercher une solution robuste. A cette fin on peut envisager plusieurs critères. Nous en considérerons ici 4 que nous présentons dans le cas d'un problème où la fonction f initiale est à maximiser.

- *maxmim :* il s'agit de maximiser la fonction $g(x) = \min_{i=1,\ldots,n} z_i(x)$. Cela revient à chercher la solution dont l'évaluation dans le pire cas est la meilleure possible. Par exemple si l'on considère la solution $x = (1, 0, 1, 0, 0, 0, 1, 1, 0, 0)$ dans l'exemple 1 on a $z(x) = (38, 100)$ et $g(x) = 38$ ce qui n'est pas optimal.
- *minmax regret :* il s'agit de minimiser la fonction $g(x) = \max_{i=1,\ldots,n} r(x, s_i)$ où $r(x, s_i) = z_i^* - z_i(x)$ avec $z_i^* = \max_{y \in X} z_i(y)$. Ici z_i^* représente la valeur de la meilleure décision qu'on peut prendre dans le scénario s_i et $r(x, s_i)$ représente le regret d'avoir choisi x dans le scénario s_i (l'amplitude du manque à gagner par rapport à la meilleure décision dans ce scénario). Il s'agit donc de chercher la solution telle que le regret maximum pris sur tous les scénarios est le plus petit possible. Pour revenir à l'exemple 1 on peut vérifier que $z_1^* = 118$ et $z_2^* = 112$. Donc la solution $x = (1, 0, 1, 0, 0, 0, 1, 1, 0, 0)$ avec $z(x) = (38, 100)$ donne les regrets $r(x, s_1) = 118 - 38 = 80$ et $r(x, s_2) = 112 - 100 = 12$ donc $g(x) = 80$ ce qui n'est pas optimal.

— *maxOWA :* il s'agit de maximiser la fonction

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

où w_i sont des poids positifs et décroissants lorsque i augmente ($w_i \geq w_{i+1}, i = 1, \ldots, n-1$) qui sont fixés par l'utilisateur et $(z_{(1)}(x), \ldots, z_{(n)}(x))$ représente le résultat d'un tri des composantes de $(z_1(x), \ldots, z_n(x))$ par ordre croissant (ainsi $z_{(i)}(x) \leq z_{(i+1)}(x), i = 1, \ldots, n-1$). On voit donc qu'on applique un poids plus important aux composantes faibles du vecteur des $z_i(x)$. Ce critère est connu sous le nom d'OWA (*Ordered Weighted Average*) soit encore "moyenne pondérée ordonnée". Pour revenir à l'exemple 1, si l'on considère que les poids de l'OWA sont $w_1 = 2$ et $w_2 = 1$ la solution $x = (1, 0, 1, 0, 0, 0, 1, 1, 0, 0)$ avec $z(x) = (38, 100)$ vaut $g(x) = 2 \times 38 + 100 = 176$. La solution $y = (0, 0, 0, 0, 0, 0, 1, 1, 1, 1)$ avec $z(y) = (118, 20)$ vaudrait quant à elle $g(y) = 2 \times 20 + 118 = 158$ et serait donc pire que x.

— *minOWA des regrets :* il s'agit d'une combinaison des deux idées précédentes qui consiste à minimiser la fonction $g(x) = \sum_{i=1}^{n} w_i\, r(x, s_{(i)})$ où $r(x, s_{(i)})$ représente le regret arrivant en $i$ème place lorsqu'on a trié les regrets $r(x, s_i)$ par ordre *décroissant*. Ici encore, les poids de l'OWA sont définis par l'utilisateur ici de telle sorte que $w_i \geq w_{i+1}, i = 1, \ldots, n-1$ ce qui permet d'attribuer un poids plus important aux regrets les plus forts. Pour revenir à l'exemple 1, si l'on considère que les poids de l'OWA sont $w_1 = 2$ et $w_2 = 1$ la solution $x = (1, 0, 1, 0, 0, 0, 1, 1, 0, 0)$ avec $z(x) = (38, 100)$ qui engendre les regrets $r(x, s_1) = 80$ et $r(x, s_2) = 12$ coûte alors : $g(x) = 2 \times 80 + 12 = 172$. La solution $y = (0, 0, 0, 0, 0, 0, 1, 1, 1, 1)$ avec $z(y) = (118, 20)$ qui engendrerait les regrets $r(y, s_1) = 0$ et $r(y, s_2) = 92$ coûterait alors : $g(y) = 2 \times 92 + 0 = 184$ et serait donc pire que x.

Ces critères s'adaptent facilement au cas d'une minimisation en gardant à l'esprit que minimiser f revient à maximiser l'opposé de la fonction initiale f. On constate qu'aucun des 4 critères envisagés ci-dessus pour l'optimisation robuste n'est linéaire en fonction des valeurs $z_i(x)$. Il va donc falloir trouver, pour chaque cas, une linéarisation, c'est-à-dire une reformulation linéaire du problème d'optimisation de $g(x)$ pour pouvoir le résoudre par la programmation linéaire en variables continues ou mixtes.

1) Linéarisation des critères maxmin et minmax regret

1.1) Formuler le problème de la recherche d'une solution optimale au sens du maxmin par un programme linéaire en variables continues ou mixtes et l'appliquer pour résoudre l'exemple 1. Déterminer alors à l'aide de Gurobi une solution x^* optimale au sens du maxmin et son vecteur image $z(x^*) = (z_1(x^*), z_2(x^*))$.

1.2) Formuler le problème de la recherche d'une solution optimale au sens du minmax regret par un programme linéaire en variables continues ou mixtes et l'appliquer pour résoudre l'exemple 1. En utilisant Gurobi, on déterminera préalablement deux solutions x_1^* et x_2^* qui maximisent $z_1(x)$ et $z_2(x)$ respectivement et on en déduira les quantités z_1^* et z_2^*. On pourra alors utiliser Gurobi de nouveau pour déterminer une solution x'^* optimale au sens du regret minmax ainsi que son vecteur image $z(x'^*) = (z_1(x'^*), z_2(x'^*))$.

1.3) Toujours dans le cas de l'exemple 1, représenter dans le plan $z_1(x), z_2(x)$ les 4 points $z(x_1^*)$, $z(x_2^*)$, $z(x^*)$ et $z(x'^*)$. En déduire qu'aucune des solutions x^* et x'^* n'aurait pu être trouvée en maximisant une moyenne pondérée (à coefficients positifs) entre $z_1(x)$ et $z_2(x)$.

1.4) On généralise maintenant l'exemple 1 en considérant un problème de sac-à-dos robuste en présence de p projets et n scénarios. On souhaite étudier l'évolution du temps de résolution en fonction de n et p. Pour $n = 5, 10, 15$ et $p = 10, 15, 20$ tirer aléatoirement 10 instances de taille (n, p) à coefficients positifs (le budget disponible pour la contrainte de poids sera fixé à 50% du coût total des p projets considérés, les coûts des projets et les utilités seront choisis entiers dans l'intervalle [1, 100]) et calculer le temps moyen de résolution des 10 instances pour chaque couple (n, p). Commenter les résultats obtenus.

2) Linéarisation du critère maxOWA

2.1) Pour tout vecteur $z \in \mathbb{R}$, on note $L(z)$ le vecteur $(L_1(z), \ldots, L_n(z))$ dont la composante $L_k(z)$ est définie par $L_k(z) = \sum_{i=1}^{k} z_{(i)}$. Expliquer pourquoi $L_k(z)$ est la valeur optimale du programme linéaire en variables 0-1 suivant :

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

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

2.2) On admettra que le programme mathématique ci-dessus peut être relaxé en variables continues (les solutions optimales du problème relaxé sont naturellement entières) ce qui fait que $L_k(z)$ est aussi la valeur à l'optimum du programme linéaire suivant :

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

Ecrire le dual D_k du programme linéaire ci-dessus, pour $k \in \{1, \ldots, n\}$ (on notera r_k la variables duale associée à la première contrainte du primal et b_{ik} les variables duales associées aux contraintes $a_{ik} \leq 1, i = 1, \ldots, n$. Utiliser cette formulation duale pour calculer par programmation linéaire les composantes du vecteur $L(2, 9, 6, 8, 5, 4)$.

2.3) Montrer que l'OWA défini par l'équation (1) peut se réécrire $g(x) = \sum_{k=1}^{n} w'_k L_k(z(x))$ avec $w'_k = (w_k - w_{k+1})$ pour $k = 1, \ldots, n-1$ et $w'_n = w_n$. On notera au passage que le vecteur $w' = (w'_1, \ldots, w'_n)$ à toutes ses composantes positives ou nulles puisque w a ses composantes décroissantes.

2.4) En utilisant les résultats des questions précédentes la maximisation d'un OWA sur un ensemble X décrit par des contraintes linéaires peut s'écrire sous la forme du programme linéaire suivant :

$$\max \sum_{k=1}^{n} w'_k (k\ r_k - \sum_{i=1}^{n} b_{ik})$$
$$s.c. \begin{cases} r_k - b_{ik} \leq z_i(x),\ i = 1, \ldots, n \\ x \in X \end{cases}$$
$$b_{ik} \geq 0, i = 1, \ldots, n$$

Cette formulation linéaire utilise n^2 variables réelles positives b_{ik} pour $i, k \in \{1, \ldots, n\}$ et n variables réelles non-signées r_k, $k \in \{1, \ldots, n\}$. En utilisant cette linéarisation de l'OWA, formuler le problème de l'optimisation d'un OWA dans l'exemple 1 comme un programme linéaire. Calculer la solution optimale du problème avec Gurobi.

2.5) En vous aidant des questions précédentes, proposer une approche similaire pour linéariser le critère minOWA des regrets et l'appliquer sur l'exemple 1 pour déterminer la solution optimale au sens du minOWA des regrets.

2.6) Etudier l'évolution du temps de résolution de problèmes de sac-à-dos robustes pour l'optimisation maxOWA et minOWA des regrets en fonction de n (nombre d'objets ou de projets) et p (nombre de scénarios). Pour $n = 5, 10, 15$ et $p = 10, 15, 20$ tirer aléatoirement 10 instances de taille (n, p) à coefficients positifs et calculer le temps moyen de résolution des 10 instances pour chaque couple (n, p) (les coûts et les utilités seront choisis entiers dans l'intervalle [1, 100], le budget disponible pour la contrainte de poids sera fixé à 50% du coût total des p objets/projets considérés). Commenter les résultats obtenus.

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

Dans un réseau modélisé par un graphe orienté on recherche un chemin rapide pour aller d'un sommet initial à un sommet destination. Plusieurs scénarios sont envisagés sur les temps de transport dans ce réseau. Plus précisément on considère un ensemble $S = \{1, \ldots, n\}$ de scénarios possibles et le temps pour parcourir chaque arc (i, j) du graphe dans les différents scénarios est donné par le vecteur $(t^1_{ij}, \ldots, t^n_{ij})$ où t^s_{ij} représente le temps nécessaire pour parcourir l'arc (i, j) dans le scénario s pour tout $s \in S$. Les temps sont additifs le long d'un chemin ce qui signifie que le temps pour parcourir un chemin P dans le scénario s est défini par $t^s(P) = \sum_{(i,j) \in P} t^s_{ij}$.

3.1) Etant donné un graphe orienté G, formuler un programme linéaire qui permette de calculer le chemin le plus rapide du sommet initial au sommet destination dans un scénario s donné, $s \in S$ (indication : on pensera à reformuler le problème comme celui de la recherche d'un flot max à coût minimum).

Exemple 2 *On considère un problème à deux scénarios $S = \{s_1, s_2\}$ et deux instances données ci-dessous dans lesquelles les vecteurs coûts étiquetant les arcs représentent les temps de parcours dans les scénarios s_1 et s_2.*

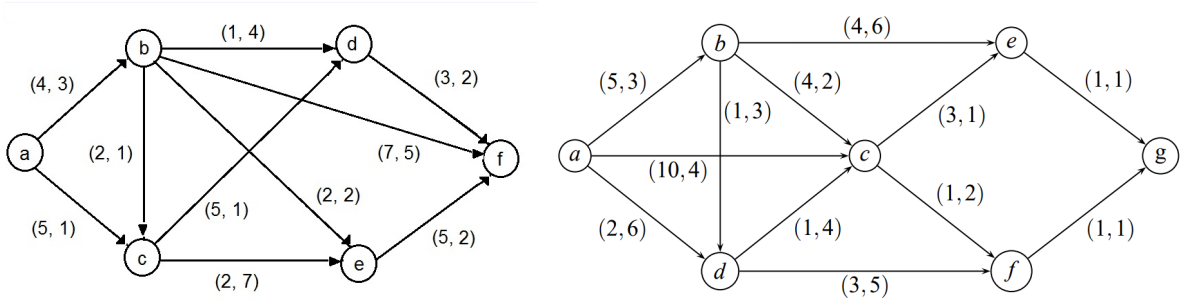

FIGURE 1 – Deux instance du problème de plus court chemin robuste à 2 scénarios

3.2) Pour chaque instance de l'exemple 2, déterminer par programmation linéaire (en variables continues ou mixtes selon les besoins) les chemins les plus rapides dans chacun des deux scénarios (de a à f pour le graphe de gauche et de a à g pour celui de droite).

3.3) Le problème de chemin robuste peut alors se formuler que celui de la recherche d'un chemin P qui optimise $v_g(P) = g(-t^1(P), \ldots, -t^n(P))$ où g est une des 4 fonctions considérées dans l'introduction. Pour chacune de ces 4 fonctions, proposer un programme linéaire (en variables continues ou mixtes selon les besoins) pour calculer un chemin robuste. Appliquer le aux graphes de la figure 2 pour obtenir, pour chaque instance, un chemin robuste de a à f (graphe de gauche) ou de a à g (graphe de droite). On cherchera un chemin P qui optimise $v_g(P)$ avec le vecteur de pondération $w = (k, 1)$ pour $k = 2, 4, 8, 16$ et on commentera l'évolution des résultats obtenus en fonction de k.

3.4) Etudier 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 (nombre de scénarios) et p (nombre de noeuds du graphe). Pour $n = 2, 5, 10$ et $p = 10, 15, 20$ tirer aléatoirement 10 instances de taille (n, p) à coefficients positifs (on tirera aléatoirement le coût des arcs dans les différents scénarios dans l'intervalle [1, 100], on recommande d'engendrer des graphes avec une densité d'arcs entre 30 et 50%) et calculer les temps moyen de résolution des 10 instances pour chaque couple (n, p). Commenter les résultats obtenus.

Modalités de travail et délivrables

Le travail est à effectuer en binôme constitué de deux personnes inscrites dans le même groupe de TD. Le binôme devra être déclaré par mail à votre chargé de TD (objet du mail : binome MOGPL-24) au plus tard le lundi 4 Novembre 2024. Les projets seront déposés au plus tard le lundi 2 décembre 2024 à minuit sur le site moodle de MOGPL. Votre livraison sera constituée d'une archive zip (pas de .tar ou .targz etc) nommée GRnumgroupe_nom1_nom2.zip qui comportera les sources du programme, un fichier README détaillant comment exécuter le programme, et un rapport rédigé (un fichier au format pdf nommé GRnumgroupe_nom1_nom2.pdf) qui présentera le travail effectué et les réponses aux différentes questions. Le plan du rapport suivra le plan du sujet. Il est fortement recommandé de rédiger son rapport en LaTeX. Les projets rendus feront l'objet d'une brève soutenance sur machine en salle tme lors de la semaine du 9 décembre 2024.

Accès au solveur Gurobi et implémentation en python

Il est fortement conseillé d'implanter vos programmes de test en python en faisant appel à *Gurobi solver* pour les résolutions de programmes linéaires (http ://www.gurobi.com/). Ce solveur est installé dans les salles de tme mais peut aussi être installé sur des machines personnelles en téléchargeant depuis une adresse de Sorbonne Université avec une licence étudiant (gratuite).