Deux approches complémentaires pour un problème d'arbre couvrant robuste

Lucie Galand, Olivier Spanjaard

Deux approches complémentaires pour un problème d'arbre couvrant robuste

L. Galand et O. Spanjaard

LIP6, 4 place Jussieu, 75252 Paris cedex 05
{lucie.galand,olivier.spanjaard}@lip6.fr

Résumé Le problème de l'arbre couvrant minimum se résout en temps polynomial par les algorithmes de Kruskal (1956) et de Prim (1957). Nous nous intéressons ici à une variante plus difficile de ce problème, où l'on recherche un arbre couvrant robuste en présence d'incertitude sur le coût des arêtes. Plus précisément, on suppose que l'incertitude est modélisée par la prise en compte explicite de plusieurs scénarios, autrement dit plusieurs jeux de valuations possibles. Il s'agit alors de trouver un arbre couvrant qui reste satisfaisant dans tous les scénarios. Nous adoptons comme mesure de robustesse la moyenne ordonnée pondérée (OWA, *Ordered Weighted Average*), dont l'utilisation en optimisation robuste a été justifiée dans [7] et [8]. Après avoir discuté la complexité du problème ainsi posé, nous présentons un algorithme approché par k-optimisation, puis un algorithme exact par séparation et évaluation, qui s'appuie sur le précédent dans sa phase d'initialisation. Des résultats numériques sont présentés, montrant l'efficacité de notre algorithme.

Mots-Clefs. Optimisation robuste ; Arbre couvrant minimum ; Séparation et évaluation.

1 Introduction

L'optimisation combinatoire robuste connaît un intérêt croissant depuis l'ouvrage [5]. Elle vise à revisiter les problèmes classiques d'optimisation combinatoire en prenant en compte explicitement l'incertitude qu'il peut y avoir sur les paramètres du problème, en particulier sur les valuations utilisées. Deux approches de la robustesse peuvent être distinguées selon la façon dont est défini l'ensemble des scénarios : le modèle par intervalles où chaque valuation est un intervalle et où l'ensemble des scénarios est défini en compréhension comme le produit cartésien de ces intervalles ; le modèle par scénarios où les valuations sont des vecteurs-coûts, dont chaque composante correspond à un scénario donné.

Nous nous intéressons ici plus spécifiquement au problème de l'arbre couvrant robuste dans le modèle par scénarios. Ce problème a été principalement étudié jusqu'à maintenant en utilisant le critère min-max comme mesure de robustesse (i.e., on recherche l'arbre couvrant dont le vecteur-coût a la composante maximale la plus petite possible). Les auteurs de [3] ont montré que ce problème est NP-difficile (une preuve alternative est présentée dans [10]) et ont proposé un algorithme exact pour le résoudre, fondé sur la k-optimisation. Plus récemment, les auteurs de [1] ont proposé un schéma complet d'approximation polynomiale pour ce problème. Nous étudions ici un problème plus général, où l'on utilise le critère de la moyenne ordonnée pondérée (OWA, *Ordered Weighted Average* [9]) comme mesure de robustesse. Ce critère, justifié dans [7] et [8] pour l'optimisation robuste, permet de rendre compte d'attitudes moins pessimistes face à l'incertitude que le critère min-max, en prenant en compte les valeurs des solutions sur l'ensemble des scénarios plutôt que sur le pire seulement.

Après avoir présenté formellement le problème et discuté sa complexité (section 2), nous proposons une méthode fondée sur la k-optimisation (section 3) dont la variante approchée s'avère beaucoup plus performante que la variante exacte, comme le montrent les expérimentations numériques présentées. Nous proposons ensuite un algorithme exact fondé sur une méthode de séparation et évaluation (section 4), dont l'initialisation est effectuée à partir de la solution renvoyée par l'algorithme approché de la section précédente. Des résultats numériques sont également fournis pour cette méthode, qui montrent son intérêt comparativement à la version exacte de l'algorithme par k-optimisation.

2 Définition du problème

Etant donné un ensemble $\{1, \ldots, q\}$ de scénarios, on peut associer un vecteur de $\mathbb{N}^q$ à chaque arbre couvrant du graphe, correspondant à la somme vectorielle des coûts des arêtes le composant. La comparaison d'arbres couvrants se réduit alors à la comparaison des vecteurs correspondants. A la suite de [7] et [8], nous proposons de comparer les vecteurs selon leur valeur OWA [9] définie comme suit :

Définition 1 *Etant donné un vecteur $x \in \mathbb{N}^q$, sa moyenne ordonnée pondérée est* $\text{OWA}(x) = \sum_{i=1}^{q} w_i x_{(i)}$, *où $x_{(1)} \geq \ldots \geq x_{(q)}$ représentent les composantes de x triées en ordre décroissant et* $\sum_{i=1}^{q} w_i = 1$.

Nous considérons ici la sous-famille des opérateurs OWA où les poids sont décroissants (i.e., $w_1 \geq \ldots \geq w_q$), ce qui conduit à accorder moins d'importance aux scénarios pour lesquels le coût est peu élevé. Cette préoccupation est naturelle pour modéliser la notion de robustesse. Ce critère est cependant moins pessimiste que le critère MAX puisqu'il accorde des poids non nuls aux scénarios pour lesquels le pire ne se réalise pas. Remarquons toutefois que l'opérateur OWA englobe le critère MAX : il suffit pour cela de prendre les poids $w_1 = 1$, $w_2 = 0$, ..., $w_q = 0$. Un autre cas particulier intéressant est obtenu lorsque l'on a de grands écarts de poids ($w_1 \gg \ldots \gg w_q$), puisque l'on retrouve alors l'ordre induit par l'opérateur LEXIMAX, qui consiste à comparer deux vecteurs sur la base de leur plus grande composante, puis de leur seconde plus grande en cas d'égalité sur la première, et ainsi de suite... Ce critère raffine donc l'opérateur MAX en départageant les vecteurs ayant la même valeur sur la plus grande composante en fonction des valeurs sur les composantes suivantes.

Dans ce cadre, la recherche d'un arbre couvrant robuste revient à résoudre le problème suivant :

ARBRE COUVRANT ROBUSTE (ACR)
Instance : un graphe non-orienté $G = (V, E)$, q fonctions de valuation $c_i : E \to \mathbb{N}$ pour $i = 1, \ldots, q$.
Objectif : déterminer un arbre couvrant T minimisant $\text{OWA}(c(T))$, où $c(T) = (c_1(T), \ldots, c_q(T))$ avec $c_i(T) = \sum_{e \in T} c_i(e)$.

Comme indiqué précédemment, lorsque $w_1 = 1$ et tous les autres poids w_i sont nuls, le critère OWA se réduit au critère MAX. Or, il a été prouvé dans [3] et [10] que le problème de la recherche d'un arbre couvrant min-max est NP-difficile. Le problème défini ici l'est donc également dans le cas général. Remarquons cependant que le problème peut devenir polynomial pour certaines familles d'instances :

- lorsque $w_1 = w_2 = \ldots = w_q$, il suffit de valuer chaque arête e par $\sum_i c_i(e)$, puis d'appliquer un algorithme classique d'arbre couvrant minimum pour obtenir la solution optimale ;
- lorsqu'il existe une permutation π des scénarios telle que $c_{\pi(1)}(e) \geq \ldots \geq c_{\pi(q)}(e)$ pour toute arête e, il suffit de valuer chaque arête e par $\sum_i w_i c_{\pi(i)}(e)$ puis d'appliquer un algorithme classique d'arbre couvrant minimum pour obtenir la solution optimale.

Précisons que ces deux cas de figure se rencontrent néanmoins assez peu fréquemment.

Enfin, il est facile de construire un schéma complet d'approximation polynomiale pour ce problème. Pour cela, à la manière de [1], on peut s'appuyer sur le schéma complet existant pour approximer la frontière de Pareto de la version multiobjectif du problème de l'arbre couvrant minimum [6]. Cet algorithme renvoie un ensemble $\mathcal{T}_\varepsilon$ d'arbres couvrants de taille polynomiale tel que pour tout arbre couvrant T du graphe il existe $T_\varepsilon \in \mathcal{T}_\varepsilon$ pour lequel $c_i(T_\varepsilon) \leq (1+\varepsilon)c_i(T)$ pour $i = 1, \ldots, q$. En remarquant que $[\forall i \quad x_i \leq y_i] \Rightarrow \text{OWA}(x) \leq \text{OWA}(y)$ et que $\text{OWA}((1+\varepsilon)x) = (1+\varepsilon)\ \text{OWA}(x)$, on peut en déduire que l'arbre couvrant T_ε^* tel que $\text{OWA}(c(T_\varepsilon^*)) = \min_{T_\varepsilon \in \mathcal{T}_\varepsilon} \text{OWA}(c(T_\varepsilon))$ vérifie $\text{OWA}(c(T_\varepsilon^*)) \leq (1+\varepsilon)\text{OWA}(c(T^*))$, où T^* désigne l'arbre couvrant optimal au sens de l'opérateur OWA. On dispose donc ainsi d'un schéma complet d'approximation polynomiale pour le problème ACR défini précédemment (l'algorithme est bien polynomial puisque la recherche de l'arbre couvrant T_ε^* se fait dans un ensemble de taille polynomiale). Cependant, la portée de ce résultat est essentiellement théorique. C'est pourquoi nous proposons dans la section suivante un algorithme (de complexité exponentielle) plus opérationnel pour approcher la solution optimale.

3 Un algorithme approché

3.1 Une approche par k-optimisation

En pratique, les solutions robustes sont souvent de bonne qualité en terme de coût moyen dans les différents scénarios. Ainsi, à la manière de [3], il peut être intéressant d'énumérer les arbres couvrants dans l'ordre croissant de leur coût moyen respectif (voir aussi [2] pour la mise en œuvre de ce type d'approche pour la recherche d'un arbre couvrant de meilleur compromis en optimisation multicritère). La question que l'on se propose d'étudier dans cette section est de savoir quand arrêter l'énumération afin obtenir une solution approchée avec un rapport d'approximation à $(1+\varepsilon)$ (avec $\varepsilon \geq 0$). A ce titre, on établit le résultat préliminaire suivant, qui lie la valeur moyenne d'un vecteur et sa valeur OWA :

Proposition 1 *Pour tout vecteur $x \in \mathbb{N}^q$ et tout jeu de poids décroissants $w_1, \ldots, w_q$ tel que $\sum_{i=1}^q w_i = 1$, on vérifie :* $\text{AVG}(x) \leq \text{OWA}\ (x)$, *où* $\text{AVG}(x) = \frac{1}{q}\sum_{i=1}^q x_i$.

Preuve. Considérons la différence $D = \text{OWA}(x) - \text{AVG}(x)$.
On a : $D = \sum_{i=1}^q w_i x_{(i)} - \sum_{i=1}^q \frac{1}{q} x_i = \sum_{i=1}^q (w_i - \frac{1}{q}) x_{(i)}$. Soit $k \in \{1, \ldots, q\}$ tel que $\forall i \in \{1, \ldots, k\}$, $w_i > \frac{1}{q}$ et $\forall i \in \{k+1, \ldots, q\}$, $\frac{1}{q} \geq w_i$. On peut décomposer D en : $\sum_{i=1}^k (w_i - \frac{1}{q}) x_{(i)} + \sum_{i=k+1}^q (w_i - \frac{1}{q}) x_{(i)}$. Comme $x_{(1)} \geq x_{(2)} \geq \ldots \geq x_{(q)}$, on a $\sum_{i=1}^k (w_i - \frac{1}{q}) x_{(i)} \geq x_{(k)} \sum_{i=1}^k (w_i - \frac{1}{q})$ et $x_{(k+1)} \sum_{i=k+1}^q (w_i - \frac{1}{q}) \geq \sum_{i=k+1}^q (w_i - \frac{1}{q}) x_{(i)}$. Ainsi, $D \geq x_{(k)} \sum_{i=1}^k (w_i - \frac{1}{q}) - x_{(k+1)} \sum_{i=k+1}^q (\frac{1}{q} - w_i)$ (1). On peut remarquer que $\sum_{i=1}^q (w_i - \frac{1}{q}) = 0$ puisque $\sum_{i=1}^q w_i = 1 = \sum_{i=1}^q \frac{1}{q}$, et donc $\sum_{i=1}^k (w_i - \frac{1}{q}) = \sum_{i=k+1}^q (\frac{1}{q} - w_i)$. Posons $W = \sum_{i=1}^k (w_i - \frac{1}{q}) = \sum_{i=k+1}^q (\frac{1}{q} - w_i)$. D'après (1), on a donc $D \geq W(x_{(k)} - x_{(k+1)})$. On en conclut que $\text{OWA}(x) \geq \text{AVG}(x)$ puisqu'il est facile de vérifier que $W \geq 0$.

Soit $\{T^1, \ldots, T^r\}$ l'ensemble des arbres couvrants du graphe, avec des vecteurs coûts $x^1, \ldots, x^r$, indicés de telle manière que $\text{AVG}(x^1) \leq \text{AVG}(x^2) \leq \ldots \leq \text{AVG}(x^r)$. La suite $(T^j)_{j=1,\ldots,r}$ peut être engendrée en implémentant un algorithme de k-optimisation (i.e., un algorithme qui énumère les k meilleures solutions dans l'ordre croissant) sur le graphe $G = (V, E)$ valué selon la fonction de valuation scalaire $c' : E \to \mathbb{R}_+$ définie par $c'(e) = \text{AVG}(c(e))$. En effet, la valeur d'un arbre couvrant T^j du graphe est $c'(T^j) = \sum_{e \in T^j} c'(e) = \sum_{e \in T^j} \text{AVG}(c(e)) = \sum_{e \in T^j} \frac{1}{q} \sum_{i=1}^q c_i(e) = \sum_{i=1}^q \frac{1}{q} \sum_{e \in T^j} c_i(e) = \sum_{i=1}^q \frac{1}{q} c_i(T^j) = \sum_{i=1}^q \frac{1}{q} x_i^j = \text{AVG}(x^j)$. L'énumération des arbres couvrants dans l'ordre croissant de la moyenne peut ainsi être réalisée à l'aide de l'algorithme de k-optimisation proposé par [4] pour le problème de l'arbre couvrant.

Supposons que durant l'énumération, on atteigne à l'étape k un arbre couvrant T^k tel que $(1+\varepsilon)\text{AVG}(x^k) \geq \text{OWA}(x^{\beta(k)})$, où $\beta(k) = \arg \min_{j \in \{1,\ldots,k\}} \text{OWA}(x^j)$ ($\beta(k)$ est l'indice de l'arbre couvrant optimal au sens de OWA parmi $\{T^1, \ldots, T^k\}$). L'énumération peut alors être arrêtée grâce à la proposition suivante :

Proposition 2 *S'il existe $k \in \{1, \ldots, r\}$ pour lequel $(1+\varepsilon)\text{AVG}(x^k) \geq \text{OWA}(x^{\beta(k)})$, alors* $\text{OWA}(x^{\beta(k)}) \leq (1+\varepsilon) \min_{j=1,\ldots,r} \text{OWA}(x^j)$.

Preuve. $T^{\beta(k)}$ est un arbre couvrant vérifiant $\text{OWA}(x^{\beta(k)}) = \min_{j=1,\ldots,k} \text{OWA}(x^j)$. Or, pour tout $j \in \{k+1, \ldots, r\}$, on a : $\text{OWA}(x^j) \geq \text{AVG}(x^j)$ (d'après la proposition 1) et $\text{AVG}(x^j) \geq \text{AVG}(x^k)$. Ainsi, $(1+\varepsilon)\text{AVG}(x^j) \geq (1+\varepsilon)\text{AVG}(x^k) \geq \text{OWA}(x^{\beta(k)})$, d'où $(1+\varepsilon)\text{OWA}(x^j) \geq \text{OWA}(x^{\beta(k)})$ pour tout $j \in \{k+1, \ldots, r\}$. Ainsi $\text{OWA}(x^{\beta(k)}) \leq \text{OWA}(x^j)$ pour $j = 1, \ldots, k$ et $\text{OWA}(x^{\beta(k)}) \leq (1+\varepsilon)\text{OWA}(x^j)$ pour $j = k+1, \ldots, r$, donc $\text{OWA}(x^{\beta(k)}) \leq (1+\varepsilon)\text{OWA}(x^j)$ pour $j = 1, \ldots, r$.

Les propositions 1 et 2 montrent qu'un arbre couvrant approchant la valeur optimale à $(1+\varepsilon)$ près peut être obtenu par l'exécution d'un algorithme de k-optimisation sur le graphe dont les arêtes sont valuées par la fonction scalaire c'. Dans le pire des cas, cela reviendrait à engendrer tous les arbres couvrants du graphe sans activer la condition d'arrêt de la proposition 2. Cependant, en pratique, la séquence croissante des $((1+\varepsilon)\text{AVG}(x^j))_{j=1,\ldots,k}$ croise la séquence décroissante des $(\text{OWA}(x^{\beta(j)}))_{j=1,\ldots,k}$ à un k très petit par rapport au nombre total d'arbres couvrants du graphe,

comme le montrent les expérimentations numériques présentées ci-dessous.

Remarque : la proposition 1 permet de montrer facilement qu'un problème d'optimisation robuste est q-approximable (i.e., on peut déterminer en temps polynomial une solution dont la valeur est au plus q fois la valeur de la solution optimale) dès lors que sa version classique (i.e., à un seul scénario) est résoluble en temps polynomial. En effet, appelons x^*_{OWA} la solution optimale du problème pour le critère OWA et x^*_{AVG} la solution optimale du problème pour le critère AVG. Remarquons tout d'abord, comme indiqué précédemment, que la solution x^*_{AVG} peut être obtenue en temps polynomial en résolvant le problème valué selon c'. De plus, on a $\text{OWA}(x) = \sum_{i=1}^{q} w_i x_{(i)} \leq \sum_{i=1}^{q} x_i = q \sum_{i=1}^{q} \frac{1}{q} x_i = q.\text{AVG}(x)$ pour tout vecteur $x \in \mathbb{N}^q$ et tout jeu de poids $w_1, \ldots, w_q$ tel que $w_i \leq 1\ \forall i$. En particulier, on vérifie donc $\text{OWA}(x^*_{\text{AVG}}) \leq q.\text{AVG}(x^*_{\text{AVG}})$. Comme $\text{AVG}(x^*_{\text{AVG}}) \leq \text{AVG}(x^*_{\text{OWA}})$, on en déduit que $\text{OWA}(x^*_{\text{AVG}}) \leq q.\text{AVG}(x^*_{\text{OWA}})$. D'après la proposition 1, on a $\text{AVG}(x^*_{\text{OWA}}) \leq \text{OWA}(x^*_{\text{OWA}})$ dès lors que les poids sont décroissants et somment à 1, et par conséquent $\text{OWA}(x^*_{\text{AVG}}) \leq q.\text{OWA}(x^*_{\text{OWA}})$.

3.2 Expérimentations numériques

L'algorithme que nous proposons a été implémenté en C++ et les tests ont été menés sur un ordinateur muni d'un processeur Pentium IV à 3.6 Ghz avec 2Go de mémoire vive. Les tableaux ci-dessous (Tab. 1) montrent les résultats de tests effectués sur des graphes complets à 3 et 5 scénarios respectivement, en fonction de la valeur de ε et du nombre de sommets du graphe. Les temps indiqués dans les tableaux correspondent au temps moyen réalisé par l'algorithme sur 30 instances aléatoires. Le symbole " - " signifie que la mémoire vive de l'ordinateur a été insuffisante pour la majorité des instances. Lorsque la mémoire a été insuffisante pour une minorité d'instances, nous faisons figurer le temps moyen obtenu sur les instances pour lesquelles l'exécution de l'algorithme a pu être menée à terme. Ceci est particulièrement le cas pour $\varepsilon = 0$. Les temps figurant sur cette ligne doivent donc être vus comme des indicateurs de performance sur des instances *non critiques*.

Tab. 1. Temps d'exécution pour 3 et 5 scénarios.

Temps (en ms) pour 3 scénarios ($w = (0.6, 0.3, 0.1)$)

$\varepsilon \backslash \lvert V \rvert$	5	10	15	20	30	40
0%	0	5	443	563	2051	4356
3%	0	2	12	380	654	973
5%	0	0	22	28	63	228
10%	0	0	0	2	2	5

Temps (en s) pour 5 scénarios ($w = (0.5, 0.3, 0.1, 0.06, 0.04)$)

$\varepsilon \backslash \lvert V \rvert$	5	10	15	20	30	40
0%	0	0,09	2,89	8,42	20,2	-
3%	0	0,02	0,57	2,1	2,91	27,64
5%	0	0,01	0,01	0,36	2	3,5
10%	0	0	0,01	0,02	0,04	2,11

Remarquons qu'on obtient un algorithme exact pour $\varepsilon = 0$. Le principal enseignement de ces tests est l'écart de performance grandissant entre les versions approchées et la version exacte lorsque le nombre de scénarios passe de 3 à 5 (par exemple, pour 30 sommets le temps est multiplié par 10 pour la version exacte, alors qu'il reste de l'ordre de quelques secondes pour la version approchée à 3%). Autrement dit, lorsque le nombre de scénarios augmente, vérifier la condition d'arrêt pour $\varepsilon > 0$ est beaucoup plus rapide que pour $\varepsilon = 0$. En effet, d'une part il existe pratiquement toujours une solution de faible OWA parmi les solutions de faible moyenne, et d'autre part il existe de nombreux arbres couvrants de coût moyen équivalent. On obtient donc rapidement une solution de bonne qualité au sens de l'OWA, mais pour parvenir à prouver l'optimalité d'une solution il

faut énumérer tous les arbres couvrants à moyenne faible, ce qui demande un gros effort de calcul et un espace mémoire important (l'algorithme de k-optimisation nécessite le stockage de données intermédiaires importantes, ce qui devient rédhibitoire lorsque la taille de l'instance augmente, comme l'indique la présence du " - " en bout de première ligne du tableau pour 5 scénarios). Pour faire face à cette difficulté et obtenir un algorithme exact praticable pour ce problème, nous présentons dans la section suivante un algorithme par séparation et évaluation. En effet, les grands ensembles de solutions de moyenne équivalente ne présentent plus la même difficulté de traitement pour ce type d'algorithme.

4 Un algorithme exact

4.1 Une approche par séparation et évaluation

Nous décrivons ci-dessous une méthode par séparation et évaluation pour le problème ACR. La méthode que nous proposons explore en profondeur d'abord l'arborescence de recherche dont chaque noeud n est caractérisé par les éléments suivants :

- $in(n)$ l'ensemble des arêtes qui doivent figurer dans tous les arbres couvrants associés à n ;
- $out(n)$ l'ensemble des arêtes qui sont interdites dans tous les arbres couvrants associés à n ;
- $ev(n)$ la valeur de la fonction d'évaluation en n, qui représente une borne inférieure de $\min_{T \in \mathcal{T}(n)} \text{OWA}(c(T))$, où $\mathcal{T}(n)$ désigne le sous-ensemble d'arbres couvrants défini implicitement par $in(n)$ et $out(n)$.

Nous détaillons maintenant plus précisément l'initialisation, le principe de séparation et la fonction d'évaluation de notre méthode :

Initialisation. Une méthode de séparation et évaluation est notoirement plus efficace quand une bonne solution est connue avant de démarrer la recherche. Dans notre méthode, la borne supérieure est initialisée par l'algorithme approché décrit dans la section 3. En effet, comme indiqué précédemment, une solution approchée de bonne qualité peut ainsi être obtenue très rapidement, ce qui permettra ensuite d'éviter une exploration trop approfondie de sous-espaces ne comportant pas de bonne solution.

Principe de séparation. En chaque noeud n, on décide de placer dans $in(n)$ ou dans $out(n)$ l'arête e minimisant sa moyenne AVG parmi les arêtes de l'ensemble $E - (in(n) \cup out(n))$. Cela revient à diviser l'espace de recherche en créant deux noeuds n' et n'', successeurs de n dans l'arborescence, tels que :

- $in(n') = in(n) \cup \{e\}$ et $out(n') = out(n)$,
- $in(n'') = in(n)$ et $out(n'') = out(n) \cup \{e\}$.

Remarquons que si ajouter l'arête e à $in(n)$ dans le noeud n' crée un cycle, alors seul le noeud n'' sera créé comme successeur de n.

Fonction d'évaluation. En chaque noeud n, on doit disposer d'une fonction d'évaluation représentant une borne inférieure de la moyenne ordonnée OWA de tout arbre couvrant issu du noeud n. Pour cela, nous proposons de considérer le coût de l'ensemble des arbres couvrants issus de n (i.e. incluant les arêtes de $in(n)$ et excluant les arêtes de $out(n)$) pour chacun des scénarios séparément ainsi que pour la moyenne arithmétique des différents scénarios. Soit $f(n) \in \mathbb{N}^q$ le vecteur coût tel que $f_i(n)$ représente la valeur de l'arbre couvrant minimum au nœud n pour le scénario i (cette valeur est obtenue par l'application de l'algorithme de Kruskal pour le scénario i en tenant compte des contraintes imposées par $in(n)$ et $out(n)$). Soit $f_{c'}(n) \in \mathbb{R}$ le coût du meilleur arbre couvrant issu de n pour la moyenne arithmétique des scénarios (cette valeur est obtenue par l'application de l'algorithme de Kruskal en tenant compte des contraintes imposées par $in(n)$ et $out(n)$, cette fois-ci sur le graphe valué par le scalaire c'). La fonction d'évaluation $ev(n)$ que nous proposons d'utiliser est la valeur optimale du programme suivant :

$$(P_n) \begin{cases} \min \text{OWA}(x) \\ \qquad x_i \geq f_i(n) \quad \forall i = 1, \ldots, q \\ \sum_{i=1}^{q} \frac{1}{q} x_i \geq f_{c'}(n) \\ \qquad x \in \mathbb{N}^q \end{cases}$$

La résolution de (P_n) demande la prise en compte de l'ordre des composantes du vecteur x. On peut donc décomposer ce problème en sous-problèmes définis chacun dans un sous-espace de $\mathbb{N}^q$ dans lequel tous les vecteurs sont comonotones (i.e. pour toute paire x, y de vecteurs, il existe une permutation π de $(1,\ldots,q)$ telle que $x_{\pi(1)} \geq \ldots \geq x_{\pi(m)}$ et $y_{\pi(1)} \geq \ldots \geq y_{\pi(m)}$). Dans chacun de ces sous-espaces, l'opérateur OWA se ramène alors à une simple somme pondérée des coûts. La résolution de P_n revient donc à résoudre chacun des programmes linéaires définis pour une permutation π de $(1,\ldots,q)$ par :

$$(P_{n,\pi}) \begin{cases} \min \sum_{i=1}^{q} w_i x_{\pi(i)} & \\ \quad x_{\pi(i)} \geq x_{\pi(i+1)} \quad \forall i = 1,\ldots,q-1 & (1.1) \\ \quad x_i \geq f_i(n) \quad \forall i = 1,\ldots,q & (1.2) \\ \sum_{i=1}^{q} \frac{1}{q} x_i \geq f_{c'}(n) & (1.3) \\ \quad x \in \mathbb{N}^q & \end{cases}$$

L'évaluation inférieure en n se définit donc comme $ev(n) = \min_{\pi \in \Pi} \text{OWA}(x^*_{n,\pi})$ où $x^*_{n,\pi}$ désigne la solution optimale du programme linéaire $P_{n,\pi}$ et Π l'ensemble des permutations possibles. Remarquons que pour q scénarios, il y a $|\Pi| = q!$ programmes linéaires à résoudre. Cependant, en pratique la résolution de ces $q!$ programmes linéaires n'est pas nécessaire. En effet, on peut montrer qu'il existe une permutation π^* aisément calculable telle que $ev(n) = \text{OWA}(x^*_{n,\pi^*})$:

Proposition 3 *Soit π^* la permutation telle que $f_{\pi^*(1)}(n) \geq f_{\pi^*(2)}(n) \geq \ldots \geq f_{\pi^*(q)}(n)$. Pour toute solution réalisable x de $P_{n,\pi}$, il existe une solution réalisable y de P_{n,π^*} vérifiant $\text{OWA}(y) = \text{OWA}(x)$.*

Preuve. L'idée est de déterminer une solution réalisable y de P_{n,π^*} telle que $y_{(i)} = x_{(i)}$ $\forall i$. En effet, on a alors $\text{OWA}(x) = \text{OWA}(y)$ et on peut immédiatement conclure. Pour cela, on construit une séquence $(x^j)_{j=1,\ldots,k}$ de solutions et une séquence $(\pi^j)_{j=1,\ldots,k}$ de permutations telles que x^j est réalisable pour P_{n,π^j} (pour $j = 1,\ldots,k$), avec $x^1 = x$, $\pi^1 = \pi$, $\pi^k = \pi^*$ et $x^1_{(i)} = x^2_{(i)} = \ldots = x^k_{(i)}$ $\forall i$. Considérons qu'il existe $i_0, i_1 \in \{1,\ldots,q\}$ tels que $i_0 < i_1$ et $f_{\pi^1(i_0)}(n) < f_{\pi^1(i_1)}(n)$. Soit π^2 la permutation définie par $\pi^2(i_0) = \pi^1(i_1)$, $\pi^2(i_1) = \pi^1(i_0)$, et $\pi^2(i) = \pi^1(i)$ $\forall i \neq i_0, i_1$. Soit x_2 la solution définie par $x^2_{\pi^2(i)} = x^1_{\pi^1(i)}$ pour $i = 1,\ldots,q$. Montrons que x_2 est bien une solution réalisable de P_{n,π^2}. En remarquant que $x^1_{\pi^1(i_0)} \geq f_{\pi^1(i_0)}(n)$, $x^1_{\pi^1(i_1)} \geq f_{\pi^1(i_1)}(n)$, $x^1_{\pi^1(i_0)} \geq x^1_{\pi^1(i_1)}$ et $f_{\pi^1(i_1)}(n) > f_{\pi^1(i_0)}(n)$, on vérifie bien que les contraintes (1.2) sont satisfaites :

- $x^2_{\pi^2(i_0)} = x^1_{\pi^1(i_0)} \geq x^1_{\pi^1(i_1)} \geq f_{\pi^1(i_1)}(n) = f_{\pi^2(i_0)}(n)$,
- $x^2_{\pi^2(i_1)} = x^1_{\pi^1(i_1)} \geq f_{\pi^1(i_1)}(n) > f_{\pi^1(i_0)}(n) = f_{\pi^2(i_1)}(n)$,
- $x^2_{\pi^2(i)} = x^1_{\pi^1(i)} \geq f_{\pi^1(i)}(n) = f_{\pi^2(i)}(n)$ pour $i \neq i_0, i_1$.

Les contraintes (1.1) sont également satisfaites car $[x^1_{\pi^1(i)} \geq x^1_{\pi^1(i+1)}\ \forall i] \Rightarrow [x^2_{\pi^2(i)} \geq x^2_{\pi^2(i+1)}\ \forall i]$ puisque $x^1_{\pi^1(i)} = x^2_{\pi^2(i)}$ $\forall i$. De ces mêmes égalités, on déduit enfin que la contrainte (1.3) est satisfaite et que $x_{(i)} = y_{(i)}$ $\forall i$. La solution x^2 est donc bien une solution réalisable de P_{n,π^2} avec $x_{(i)} = y_{(i)}$ $\forall i$. Comme toute permutation est le produit de permutations élémentaires, on peut toujours construire ainsi progressivement une séquence de permutations qui mène à π^* (et les solutions réalisables correspondantes). En posant $y = x^k$, on obtient alors la solution réalisable recherchée de P_{n,π^*}.

Une conséquence immédiate de ce résultat est que $ev(n) = \text{OWA}(x^*_{n,\pi^*})$. Ainsi le calcul de la fonction d'évaluation $ev(n)$ revient à résoudre le programme linéaire P_{n,π^*}. De plus, nous montrons maintenant que la résolution de ce programme peut être effectuée en temps linéaire du nombre de scénarios sans recourir à un solveur de programme linéaire.

Le principe de cette procédure (algorithme 1) est de fixer $x^*_i = f_i(n)$ pour tout $i = 1,\ldots,q$, de sorte que les contraintes (1.1) et (1.2) de P_{n,π^*} soient vérifiées. Cependant, si la somme des coûts de x^* est inférieure à $qf_{c'}(n)$, la contrainte (1.3) n'est pas vérifiée. Afin de la satisfaire, il est alors nécessaire de répartir le surplus $\Delta = qf_{c'}(n) - \sum_{j=1}^{q} f_j(n)$ entre les différentes composantes de x^*. Pour cela, l'idée est d'augmenter les coûts des composantes de x^* en modifiant en priorité les performances des composantes de poids les plus faibles (dans l'ordre $x^*_{\pi^*(q)}$, $x^*_{\pi^*(q-1)}$, ...),

en veillant bien sûr à respecter la contrainte (1). Ce calcul peut être accéléré en déterminant directement la valeur de la composante minimale du vecteur ainsi modifié (quantité $\lfloor r \rfloor$ dans l'algorithme 1).

Algorithme 1 : Résolution de P_{n,π^*}

$\Delta \leftarrow q f_{c'}(n) - \sum_{j=1}^{q} f_j(n)$
$s \leftarrow 0$; $a \leftarrow 0$; $i \leftarrow q$

/* Calcul du nombre $q - i$ de composantes à modifier */
/* a : quantité max pouvant être ajoutée aux $q - i$ dernières composantes */
tant que $a < \Delta$ **faire**
  si $i = 1$ **alors**
    $a \leftarrow \infty$
  sinon
    $a \leftarrow a + (f_{i-1}(n) - f_i(n))(q - i + 1)$
  fin
  $s \leftarrow s + f_i(n)$
  $i \leftarrow i - 1$
fin

/* Construction du vecteur $x^*_{(n,\pi^*)}$ */
$r \leftarrow \dfrac{s + \Delta}{q - i}$
$k \leftarrow (r - \lfloor r \rfloor) \times (q - i)$
pour j allant de 1 à i **faire**
  $x^*_{n,\pi^*(j)} \leftarrow f_j(n)$
fin
pour j allant de $(i+1)$ à $(i+k)$ **faire**
  $x^*_{n,\pi^*(j)} \leftarrow \lfloor r \rfloor + 1$
fin
pour j allant de $(i+k+1)$ à q **faire**
  $x^*_{n,\pi^*(j)} \leftarrow \lfloor r \rfloor$
fin
Sortie : $(x^*_{(n,\pi^*)})$

Afin d'illustrer le principe de notre algorithme, considérons maintenant un exemple de calcul de borne inférieure sur un problème à 3 scénarios en un noeud n de l'arborescence de recherche tel que l'on a $f(n) = (5, 10, 3)$ et $f_{c'}(n) = 7$. Le programme (P_n) à résoudre est alors le suivant (peu importe les valeurs $w_1 \geq w_2 \geq w_3$) :

$$(P_n) \begin{cases} \min & w_1 x_{(1)} + w_2 x_{(2)} + w_3 x_{(3)} \\ & x_1 \geq 5 \\ & x_2 \geq 10 \\ & x_3 \geq 3 \\ & \frac{1}{3}(x_1 + x_2 + x_3) \geq 7 \\ & x \in \mathbb{N}^3 \end{cases}$$

Selon la proposition 3, la résolution de ce programme se ramène à la résolution du programme linéaire (P_{n,π^*}) suivant, pour lequel $\pi^*(1) = 2$, $\pi^*(2) = 1$ et $\pi^*(3) = 3$ (puisque l'on a $f_2(n) \geq f_1(n) \geq f_3(n)$).

$$(P_{n,\pi^*}) \begin{cases} \min & w_1 x_2 + w_2 x_1 + w_3 x_3 \\ & x_2 \geq x_1 \geq x_3 & (2.1) \\ & x_1 \geq 5 & (2.2) \\ & x_2 \geq 10 & (2.3) \\ & x_3 \geq 3 & (2.4) \\ & x_1 + x_2 + x_3 \geq 21 & (2.5) \\ & x \in \mathbb{N}^3 \end{cases}$$

L'idée à partir de laquelle l'algorithme 1 a été conçu est la suivante :

- on pose $x = f(n) = (5, 10, 3)$: les contraintes (2.1), (2.2), (2.3) et (2.4) sont ainsi vérifiées. Cependant, la contrainte (2.5) ne peut être vérifiée que si l'on ajoute la quantité $\Delta = 3$ sur une ou plusieurs composantes de x.
- On augmente alors la composante de poids minimum (i.e. x_3) autant que l'on peut en tenant compte de la contrainte (2.1) : cela revient à poser $x_3 = \min\{x_1; x_3 + \Delta\}$. Ici on obtient $x_3 = 5$, ce qui n'est pas suffisant pour satisfaire la contrainte (2.5).
- On augmente alors simultanément les deux composantes de poids les plus faibles x_1 et x_3. Puisqu'il ne reste plus que la quantité 1 à rajouter et que la contrainte (2.1) impose $x_1 \geq x_3$, on pose $x_1 = x_1 + 1$ et l'on n'ajoute rien sur la composante x_3.
- La solution $x = (6, 10, 5)$ ainsi obtenue est réalisable pour P_{n,π^*} : on est à l'optimum.

Notons que l'exécution de l'algorithme 1 sur cet exemple permet d'obtenir directement $\lfloor r \rfloor = 5$, $i = 1$ et $k = 1$, ce qui bien sûr conduit à la solution $x^*_{n,\pi^*} = (6, 10, 5)$.

4.2 Résultats expérimentaux

Afin de tester l'efficacité de la méthode par séparation et évaluation, nous avons réalisé des tests dans les mêmes conditions que pour l'algorithme de la section 3. Le tableau 2 montre les temps obtenus sur des graphes complets en fonction du nombre de scénarios et du nombre de sommets du graphe. Les temps indiqués dans les tableaux correspondent au temps moyen total (initialisation + exploration) sur 50 instances aléatoires, la durée moyenne spécifiquement consacrée à la phase d'initialisation étant précisée entre parenthèses. Cette phase d'initialisation est réalisée à l'aide de l'algorithme de k-optimisation pour $\varepsilon = 0,05$. En effet, cette valeur de ε permet d'obtenir très rapidement une solution de bonne qualité, comme on l'a vu dans la section précédente.

Tab. 2. Temps d'exécution (en s)

$q \backslash \lvert V \rvert$	5	10	15	20	30	40
2	0	0	0	0,01	0,09	2,2
	(0)	(0)	(0)	(0)	(0)	(0,01)
3	0	0	0,02	0,07	1,2	3,4
	(0)	(0)	(0)	(0)	(0,02)	(0,02)
5	0	0,06	0,9	15	229	261
	(0)	(0,01)	(0,07)	(0,41)	(1,55)	(3,3)

Ces résultats montrent que cette approche permet d'obtenir une solution optimale en un temps inférieur à l'approche exacte de la section 3 pour 3 scénarios. Concernant les instances avec 5 scénarios, les temps obtenus deviennent plus importants à partir de 20 sommets, mais cela ne peut être interprété comme une supériorité de l'approche exacte par k-optimisation sur la méthode par séparation et évaluation car les temps figurant dans le tableau 1 pour $\varepsilon = 0$ ne tiennent compte que des instances non critiques. De plus, la méthode par séparation et évaluation ne souffre pas des problèmes de mémoire vive de la précédente car l'exploration de l'arborescence est réalisée en profondeur : toutes les exécutions sont menées à leur terme.

5 Conclusion

Dans ce papier, nous avons présenté deux approches pour un problème d'arbre couvrant robuste lorsque l'on souhaite prendre en compte plusieurs scénarios sur les coûts des arêtes. Plus précisément, le problème consiste à déterminer un arbre couvrant minimisant la moyenne ordonnée pondérée de ses coûts dans les différents scénarios. La première approche s'appuie sur un algorithme de k-optimisation et fournit rapidement une solution approchée selon un rapport d'approximation contrôlé. La seconde approche procède par séparation et évaluation pour déterminer une solution optimale. Sa principale originalité réside dans la fonction d'évaluation utilisée pour élaguer la recherche. Cette fonction d'évaluation est calculable en temps constant dès lors que le

nombre de scénarios est considéré comme une constante. Les résultats numériques obtenus montrent d'ailleurs que le nombre de scénarios est un paramètre crucial dans la difficulté du problème. Enfin, soulignons que les deux approches que nous avons développées sont assez générales (ni la proposition 1 que nous utilisons pour établir la validité du premier algorithme, ni la proposition 3 qui justifie la fonction d'évaluation du second algorithme, ne dépendent de la structure combinatoire du problème traité ici), et devraient donc pouvoir être adaptées efficacement à d'autres problèmes d'optimisation combinatoire robuste. C'est un sujet d'étude intéressant pour des travaux futurs.

Remerciements

Nous remercions Patrice Perny et Francis Sourd pour de multiples échanges qui ont contribué à cet article, ainsi que les relecteurs anonymes pour leurs suggestions pertinentes.

Références

1. Aissi, H., Bazgan, C. et Vanderpooten, D. : Approximation of min-max and min-max regret versions of some combinatorial optimization problems. European Journal of Operational Research (à paraître)
2. Galand, L. : Interactive search for compromise solutions in multicriteria graph problems. 9th IFAC Symposium on Automated Systems Based on Human Skill And Knowledge, 22-25 mai, Nancy (2006)
3. Hamacher, H.W. et Ruhe., G. : On spanning tree problems with multiple objectives. Annals of Operations Research 52, p. 209–230 (1994)
4. Katoh, N., Ibaraki, T. et Mine, H. : An Algorithm for Finding k Minimum Spanning Trees. SIAM J. Comput. 10, p. 247–255 (1981)
5. Kouvelis, P. et Yu, G. : Robust discrete optimization and its applications. Kluwer Academic Publisher (1997)
6. Papadimitriou, C.H. et Yannakakis, M. : The complexity of tradeoffs, and optimal access of web sources. 41st Annual IEEE Symposium on Foundations of Computer Science FOCS, p. 86–92 (2000)
7. Perny P. et Spanjaard, O. : An Axiomatic Approach to Robustness in Search Problems with Multiple Scenarios. Proceedings of the 19th conference on Uncertainty in Artificial Intelligence, p. 469–476, Mexico (2003)
8. Perny, P., Spanjaard, O. et Storme, L.-X. : A decision-theoretic approach to robust optimization in multivalued graphs. Annals of Operations Research (à paraître)
9. Yager, R.R. : On ordered weighted averaging aggregation operators in multicriteria decision making. IEEE Trans. Systems, Man and Cybern., volume 18, p. 183–190 (1998)
10. Yu, G. : Min-max optimization of several classical discrete optimization problems. Journal of Optimization Theory and Applications, 98(1), p.221–242 (1998)