Techniques d'allocation de lots avec des préférences conflictuelles représentées par des graphes acycliques dirigés pondérés

Sara Maqrot, Stéphanie Roussel, Gauthier Picard, Cédric Pralet

ONERA/DTIS, Université de Toulouse

prenom.nom@onera.fr

Résumé

Nous introduisons des techniques d'allocation de ressources pour un problème où (i) les agents expriment des demandes d'obtention de lots d'objets sous forme de graphes acycliques dirigés compacts pondérés (chaque chemin dans ces graphes est un lot dont l'évaluation est la somme des poids des arêtes traversées), et (ii) les agents ne demandent pas exactement les mêmes articles mais les demandes peuvent porter sur des objets conflictuels, qui ne peuvent pas être tous deux assignés. Ce cadre est motivé par des applications réelles telles que l'allocation de créneaux d'observation de la Terre, la virtualisation de fonctions réseaux ou la recherche de chemins multi-agents. Nous étudions plusieurs techniques d'allocation et analysons leurs performances sur un problème d'allocation de portions d'orbite.

Mots-clés

Allocation de ressources, approche utilitariste, partage équitable, graphes.

Abstract

We introduce resource allocation techniques for a problem where (i) the agents express requests for obtaining item bundles as compact edge-weighted directed acyclic graphs (each path in such graphs is a bundle whose valuation is the sum of the weights of the traversed edges), and (ii) the agents do not bid on the exact same items but may bid on conflicting items, that cannot be both assigned. This setting is motivated by real applications such as Earth observation slot allocation, virtual network functions, or multi-agent path finding. We study several allocation techniques and analyze their performances on an orbit slot ownership allocation problem.

Keywords

Resource allocation, utilitarianism, fair division, graphs.

1 Introduction

Nous considérons un problème d'allocation de lots d'articles indivisibles contraint par le chaînage d'articles (pour allouer à chaque agent une chaîne d'articles successifs) et des articles conflictuels. La première contrainte est capturée par un graphe acyclique dirigé (DAG) avec des arêtes pondérées, un nœud source et un nœud puits. Ce graphe représente tous les lots (*i.e.* chemins) d'articles valides pour un agent, où la qualité d'un lot est représentée par les poids additifs des arêtes. La deuxième contrainte stipule que chaque agent doit obtenir un chemin sans conflit dans son graphe. On trouve ces problèmes d'allocations dans des domaines d'application tels que la virtualisation des fonctions de réseau (NFV), où les utilisateurs demandent l'allocation de graphes dirigés de services dans une infrastructure réseau partagée [20], ou dans l'observation de la Terre à l'aide d'une constellation de satellites, où les utilisateurs demandent l'exclusivité sur des portions d'orbite (sans chevauchement avec les portions d'autres utilisateurs) pour mettre en œuvre leurs demandes d'observation périodiques [11, 17]. Dans de tels contextes, outre les pondérations additives des arêtes, d'autres critères peuvent être pris en compte pour guider le processus d'allocation, en particulier lorsque les utilisateurs de la constellation sont des parties prenantes qui s'attendent à ce que les allocations soient équitables ou proportionnelles à leur investissement.

Travaux connexes. La littérature contient quelques travaux relatifs à l'allocation de biens structurés sous forme de graphes. Dans la division équitable de graphe, l'objectif est de diviser un graphe d'éléments entre plusieurs agents, avec des utilités additives attachées aux nœuds [1, 8]. Ces travaux fournissent des propriétés intéressantes pour trouver des allocations sans envie (*envy-free*) ou Pareto-optimales, de manière efficace dans certaines structures de graphes spécifiques, comme par exemple les chemins, les arbres, les étoiles. Cependant, dans notre problème, (i) les agents ne sont pas en compétition pour le même ensemble d'éléments, (ii) le graphe est dirigé pour composer des chemins d'un article de début à un article de fin, (iii) même en faisant correspondre notre problème à une division de graphe et en regroupant les éléments conflictuels en éléments composites, il est hautement improbable que le graphe résultant soit acyclique. Ici, les graphes sont utilisés pour exprimer les préférences, et non les biens à allouer. En bref, notre travail ne s'inscrit pas dans les cadres existants de partage équitable de graphes, ou ne peut pas bénéficier des résultats théoriques sur les graphes en forme de chemin ou d'étoile.

D'autres travaux connexes concernent les enchères de chemins [9, 6, 21], où les agents font des offres pour des chemins dans un graphe où chaque arête appartient à un

agent. L'objectif est d'attribuer des chemins aux agents par le biais d'enchères, et éventuellement de préserver une certaine confidentialité pour les propriétaires des arêtes. Dans le cas d'une fonction objectif utilitaire pour le problème de détermination du vainqueur, sans confidentialité des prix, cela entre dans le cadre de Vickrey-Clarke-Groves (VCG), et garantit donc des mécanismes efficaces et sans risque de confusion. Mais, là encore, les agents font des offres sur le même ensemble de nœuds et d'arêtes.

Dans le domaine du transport, les recherches sur des structures très similaires, à savoir les réseaux de flux, fournissent des techniques pour un flux maximal équitable dans les réseaux multi-sources et multi-puits [12]. Bien que les techniques utilisées soient très similaires aux nôtres (programmation linéaire), l'objectif de débit maximal est très différent de la maximisation de l'utilité du chemin. Par ailleurs, [7] a travaillé sur des problèmes de plus court chemin multiple basés sur des techniques de déconfliction. Bien que le problème présente des caractéristiques similaires, encore une fois les agents évoluent sur les mêmes graphes, et l'objectif est centré sur la minimisation de la longueur du chemin et la minimisation des chemins conflictuels, sans considérations d'équité.

Dans les jeux de congestion, les agents se voient attribuer des chemins de manière à minimiser les délais dûs au croisement des chemins. Si des agents se voient attribuer les mêmes nœuds, alors le retard lié à leurs chemins augmente : [14, 16]. Dans notre travail, nous ne considérons pas le retard mais des incompatibilités. Bien que ces dernières puissent être modélisées par des fonctions non linéaires $\{0, \infty\}$, certaines allocations de chemins sont irréalisables dans notre problème, contrairement aux jeux de congestion. En outre, l'utilisation de méthodes de résolution de jeux de congestion comme dans [16] peut aboutir à des équilibres de Nash injustes, en raison de nombreux chemins irréalisables.

Plus généralement, une autre approche classique de l'allocation équitable de biens indivisibles est le *round-robin*, qui est presque sans envie [3]. C'est notamment une technique privilégiée pour allouer des fonctions réseau virtuelles dans les infrastructures de virtualisation de fonctions réseau [19], ou pour ordonnancer des tâches. Nous comparons nos techniques à cette approche.

Contributions. Cet article introduit et présente un modèle pour de tels scénarios, sous le prisme de l'optimalité et de l'équité. Le modèle capture toute application dans lesquelles les agents expriment leurs préférences sous forme de DAGs avec arêtes pondérées et dans lesquelles il existe des conflits entre certains nœuds de ces graphes. Nous montrons que ce problème d'allocation est NP-difficile. Nous présentons et évaluons plusieurs algorithmes sur des données provenant de constellations de satellites et de demandes simulées, selon trois critères : l'optimalité utilitaire, le temps de calcul et l'équité.

2 Modèle du problème

Nous étudions des problèmes d'allocation où les évaluations des agents sur les lots d'articles sont représentées comme des DAGs avec arêtes pondérées, avec la présence de certains conflits entre les nœuds de ces graphes. Une allocation consiste à choisir un chemin complet dans chaque graphe, de sorte que les chemins sélectionnés ne soient pas en conflit les uns avec les autres.

2.1 Définitions et notations

Définition 1. Un problème d'*allocation de multiple chemins conflictuels dans des graphes acycliques dirigés avec arêtes pondérées* (PADAG) est un tuple $\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle$, où :

- $\mathcal{A} = \{1, \ldots, n\}$ est un ensemble d'*agents* ;
- $\mathcal{G} = \{g_1, \ldots, g_m\}$ est un ensemble de DAGs pondérés, où chaque $g \in \mathcal{G}$ est un tuple $\langle V_g, E_g, u_g \rangle$ qui représente des préférences sur certains éléments dans V_g, avec des connexions entre les éléments dans $E_g \subset V_g \times V_g$, pondéré en utilisant la fonction d'utilité $u_g : V_g \times V_g \to \mathbb{R}$; nous supposons également que V_g contient deux nœuds spécifiques, la source s_g et le puits t_g, et que E_g contient une arête de s_g à t_g pondérée par l'utilité 0 (afin de traiter les cas où aucun lot d'éléments ne peut être sélectionné dans g) ;
- $\mu : \mathcal{G} \to \mathcal{A}$ fait correspondre chaque graphe g dans $\mathcal{G}$ à son propriétaire a dans $\mathcal{A}$;
- $\mathcal{C} \subset \{(v, v') | (v, v') \in V_g \times V_{g'}, g, g' \in \mathcal{G}^2, \mu(g) \neq \mu(g')\}$ est un ensemble de conflits entre des paires d'éléments de deux graphes distincts de $\mathcal{G}$ provenant de deux agents distincts.

Pour chaque graphe g et chaque ensemble d'arêtes $X \subseteq E_g$, l'utilité de X pour g est définie par $u_g(X) = \sum_{e \in X} u_g(e)$, ce qui signifie que les évaluations des arêtes sont considérées comme additives dans cet article. Par conséquent, chaque chemin de s_g à t_g dans un graphe g est évalué en additionnant les utilités des arêtes traversées, et chaque DAG représente de manière compacte un ensemble d'évaluations pour des paquets d'éléments, comme dans les enchères combinatoires. De plus, nous désignons par $\mathcal{G}_a = \mu^{-1}(a)$ l'ensemble des graphes possédés par l'agent a, et l'utilité d'un ensemble d'arêtes X pour l'agent a est définie par $u_a(X) = \sum_{g \in \mu^{-1}(a)} u_g(X \cap E_g)$.

Définition 2. Une *allocation* est une fonction π qui associe, à chaque graphe $g \in \mathcal{G}$, un chemin $\pi(g)$ de s_g à t_g dans g. Formellement, $\pi(g)$ peut être représentée comme un ensemble de nœuds dans V_g. En effet, comme on manipule des DAGs, il est facile de reconstruire les arêtes successivement traversées par le chemin à partir de cet ensemble. Par extension, l'allocation pour l'agent a est donnée par $\pi(a) = \cup_{g \in \mu^{-1}(a)} \pi(g)$.

Par convention, nous désignons par $u(\pi(g)) = u_{\mu(g)}(\pi(g))$ (resp. $u(\pi(a)) = u_a(\pi(a))$), l'utilité du graphe g

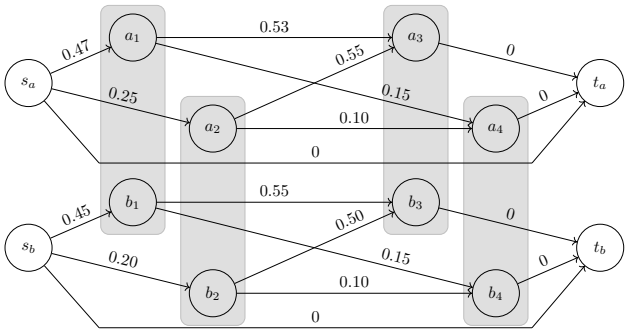

FIGURE 1 – Exemple d'évaluations (ou préférences) de lots par des agents représentées sous forme de DAGs. Les conflits sont représentés par des hypernœuds gris.

(resp. de l'agent a) pour l'allocation π. Enfin, l'utilité globale obtenue avec l'allocation π est donnée par $u(\pi) = \sum_{g \in \mathcal{G}} u(\pi(g))$ (ou de manière équivalente $u(\pi) = \sum_{a \in \mathcal{A}} u(\pi(a))$).

Définition 3. Une allocation π est *valide* si pour chaque paire de graphes distincts g et g' il n'y a pas de conflit entre les nœuds dans les chemins résultants, c'est-à-dire $(\pi(g) \times \pi(g')) \cap \mathcal{C} = \emptyset$.

Exemple 1. La figure 1 illustre un PADAG. Les meilleurs chemins pour les agents a et b sont respectivement $\{s_a, a_1, a_3, t_a\}$ et $\{s_b, b_1, b_3, t_b\}$, tous deux évalués à 1. En raison de conflits de nœuds (par exemple entre a_1 et b_1), ces deux chemins ne peuvent pas être assignés en même temps. Une allocation valide pourrait être $\pi_{\mathsf{ex}} = \{a \mapsto \{s_a, a_2, a_4, t_a\}, b \mapsto \{s_b, b_1, b_3, t_b\}\}$ avec une utilité globale $u(\pi_{\mathsf{ex}}) = u(\pi_{\mathsf{ex}}(a)) + u(\pi_{\mathsf{ex}}(b)) = 0.35 + 1.0 = 1.35$.

Les problèmes que nous considérons dans cet article sont (i) *comment calculer une allocation (utilitaire) optimale* π qui maximise $u(\pi)$, et (ii) *comment calculer une allocation équitable optimale* π, par le biais d'une optimisation leximin, *c.a.d.* maximisant lexicographiquement le vecteur d'utilité ordonné $(\Lambda_1, \ldots, \Lambda_n)$ contenant une composante par agent. Plus précisément, ce vecteur contient les utilités de tous les agents. Formellement, il existe une bijection entre les utilités $u(\pi(i))$ des agents i de $\mathcal{A}$ et les éléments Λ_j du vecteur. De plus, le vecteur est ordonné, *i.e.* $\forall i \leq j$, $\Lambda_i \leq \Lambda_j$.

2.2 Analyse de complexité

Les deux propositions suivantes fournissent des résultats de complexité sur les deux problèmes étudiés, relatifs à l'optimisation utilitaire et à l'optimisation leximin respectivement.

Proposition 1. *Déterminer s'il existe une allocation valide π telle que l'évaluation utilitaire $u(\pi)$ est supérieure ou égale à une valeur donnée est NP-complet.*

Démonstration. Tout d'abord, le problème est NP puisque $u(\pi)$ est calculable en temps polynomial. Ensuite, il existe une réduction polynomiale de 3-SAT (qui est NP-complet) à notre problème. Fondamentalement, dans une formule 3-SAT, chaque clause sur des variables propositionnelles x, y, z peut être représentée comme un DAG pondéré g où (1) l'ensemble des nœuds est $V_g = \{x, \neg x, y, \neg y, z, \neg z, s_g, t_g\}$, (2) l'ensemble des chemins de s_g à t_g dans g correspond à l'ensemble des valeurs de vérité pour x, y, z qui satisfont la clause (représentation du diagramme de décision), (3) le poids de chaque arête est fixé à 0 sauf pour les arêtes $s_g \to n$ où $n \neq t_g$ qui ont un poids de $1/m$, avec m le nombre de clauses dans la formule 3-SAT. Enfin, pour chaque variable propositionnelle x, nous pouvons ajouter un conflit (n, n') pour chaque paire de nœuds étiquetés par les littéraux x et $\neg x$ dans deux graphes distincts. Ensuite, comme un chemin est sélectionné dans chaque graphe et comme il y a m graphes, déterminer s'il existe une allocation valide π telle que $u(\pi) \geq 1$ est équivalent à trouver une solution qui satisfait toutes les clauses, d'où le résultat de NP-complétude étant donné que toutes les opérations utilisées dans la transformation sont polynomiales. □

Proposition 2. *Il est NP-complet de décider s'il existe une allocation valide dont l'évaluation leximin est supérieure ou égale à un vecteur d'utilité donné. La proposition est valable même s'il existe un graphe unique par agent.*

Démonstration. Dans le cas général, il suffit de considérer un problème impliquant un agent unique possédant tous les graphes, et d'utiliser le résultat de la proposition précédente. S'il y a un seul graphe par agent, il suffit d'utiliser exactement le même codage 3-SAT que précédemment mais de remplacer les poids $1/m$ par des poids 1. Il est alors possible de montrer qu'il existe une allocation valide dont l'évaluation leximin est supérieure ou égale à $(1, 1, \ldots, 1)$ s'il existe une solution au problème 3-SAT. De plus, l'évaluation leximin d'une allocation π peut être calculée en temps polynomial, d'où le résultat de NP-complétude. □

2.3 À propos des conflits

Dans les PADAGs, les agents sont en compétition pour acquérir les noeuds reliés par l'ensemble $\mathcal{C}$. Les PADAGs reviennent à allouer chaque clique maximale K_i dans le graphe de conflit $\langle \bigcup_{g \in \mathcal{G}} V_g, \mathcal{C} \rangle$ à un agent. L'évaluation d'un agent pour un lot $\mathcal{B} = \{K_1, \ldots, K_p\}$ de cliques maximales peut être définie comme l'utilité du meilleur chemin parmi tous les graphes de l'agent traversant un nœud de chaque K_i, mais ne traversant aucun nœud d'une clique en dehors de $\mathcal{B}$.

Pour résoudre ce problème, il est possible de s'appuyer sur des enchères combinatoires, où chaque agent enchérit sur l'ensemble des cliques maximales qu'il souhaite (lots de cliques), et un commissaire-priseur détermine le gagnant d'une manière utilitaire ou équitable, par exemple. Cependant, il est à noter que (i) trouver toutes les cliques maximales d'un graphe est $\mathcal{O}(3^{\frac{n}{3}})$ dans le pire des cas [2], et (ii) trouver une allocation maximisant le bien-être dans les enchères combinatoires est NP-difficile, en général [5]. De plus, l'évaluation de tels lots nécessite que les enchérisseurs connaissent les conflits, ce qui peut ne pas être une information publique. Par conséquent, nous considérons dans la

suite que les agents expriment leurs demandes (ensuite traduites en DAGs), sans savoir avec quels agents leurs offres sont en conflit. Seul le coordinateur (par exemple, l'opérateur de réseau ou l'opérateur de constellation) détermine ces conflits.

2.4 À propos des préférences et des comportements stratégiques

Alors que les approches leximin et utilitariste (dans un mécanisme VCG) sont à l'abri des stratégies (*strategyproof*), le fait que les agents définissent directement leur évaluation peut être problématique : un agent peut définir un graphe avec un seul chemin, le seul lot qu'il accepte. En utilisant une approche leximin, il est très probable qu'on lui attribue ce seul lot, pour éviter une utilité de 0 pour le pire agent. Cette situation pourrait être évitée dans certains contextes. Par exemple, les agents devraient seulement être autorisés à demander des articles, et ne pas exprimer directement les graphes. Le coordinateur pourrait ainsi générer des graphes connexes, qui ont de nombreux chemins possibles, en raison de la configuration du système. De plus, dans de nombreux contextes opérationnels, plusieurs utilisateurs peuvent avoir certaines priorités (par exemple, les agences de défense dans le cadre de l'observation par des satellites), ce qui pourrait nécessiter que certaines demandes aient des poids plus élevés et de désactiver la normalisation, ou d'adopter une approche d'allocation itérative, par niveau de priorités. Ainsi, dans la suite du document, nous ne considérerons que les agents ayant la même priorité et qui ne définissent pas les graphes par eux-mêmes.

3 Schémas d'allocation de chemins

Nous proposons ici plusieurs schémas d'allocation pour les PADAGs. Certains d'entre eux sont basés sur la programmation linéaire en nombres entiers (ILP) et la programmation linéaire en nombres entiers mixtes (MILP). Nous commençons par introduire les variables de décision et les contraintes pour ces modèles.

Pour tout DAG $g = \langle V_g, E_g, u_g \rangle$, nous définissons des variables binaires $x_e \in \{0, 1\}$, pour tout $e \in E_g$, indiquant si l'arête e est sélectionnée dans le chemin définissant le lot solution. Nous utilisons également des variables binaires auxiliaires β_v indiquant si le nœud v est sélectionné dans le chemin de solution $\pi(g)$, c'est-à-dire que $\beta_v = 1$ si $v \in \pi(g)$, 0 sinon. Pour tout nœud v dans V_g, nous désignons par $\mathsf{In}(v)$ (resp. $\mathsf{Out}(v)$) son ensemble d'arêtes entrantes (resp. sortantes). Dans tous les modèles ILP introduits par la suite, nous imposons les contraintes (1)–(3) pour définir tous les chemins possibles, les contraintes (4)–(5) pour tenir compte des conflits de sélection d'éléments, et la contrainte (6) pour s'assurer que les sources et les puits sont sélectionnés.

$$\sum_{e \in \mathsf{In}(v)} x_e = \sum_{e \in \mathsf{Out}(v)} x_e, \quad \forall g \in \mathcal{G}, \forall v \in V_g \setminus \{s_g, t_g\} \tag{1}$$

$$\sum_{e \in \mathsf{Out}(s_g)} x_e = 1, \quad \forall g \in \mathcal{G} \tag{2}$$

$$\sum_{e \in \mathsf{In}(t_g)} x_e = 1, \quad \forall g \in \mathcal{G} \tag{3}$$

$$\sum_{e \in \mathsf{In}(v)} x_e = \beta_v, \quad \forall g \in \mathcal{G}, \forall v \in V_g \setminus \{s_g, t_g\} \tag{4}$$

$$\sum_{v \in c} \beta_v \leq 1, \quad \forall c \in \mathcal{C} \tag{5}$$

$$\beta_{s_g} = \beta_{t_g} = 1, \quad \forall g \in \mathcal{G} \tag{6}$$

3.1 Allocation utilitariste (util)

L'approche classiquement utilisée en allocation de ressources est l'approche utilitaire. Elle consiste à trouver l'allocation qui maximise la somme des utilités de tous les chemins sélectionnés. Cela correspond à la résolution du programme linéaire en nombres entiers $P_{\mathsf{util}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle)$ donné ci-dessous :

$$\textbf{max} \quad \sum_{a \in \mathcal{A}} \sum_{g \in \mathcal{G}_a} \sum_{e \in E_g} u_g(e) \cdot x_e \tag{7}$$
$$\text{t.q.} \quad (1), (2), (3), (4), (5), (6)$$

L'allocation résultante π est décodée à partir des variables β_v. Formellement, pour tout $g \in \mathcal{G}$, $\pi(g) = \{v \in V_g \mid \beta_v = 1\}$.

Exemple 2. Dans la figure 1, l'allocation utilitaire optimale est $\pi_{\mathsf{util}} = \{a \mapsto \{s_a, a_2, a_3, t_a\}, b \mapsto \{s_b, b_1, b_4, t_b\}\}$, avec une utilité globale $u(\pi_{\mathsf{util}}) = u(\pi_{\mathsf{util}}(a)) + u(\pi_{\mathsf{util}}(b)) = 0.80 + 0.60 = 1.40$.

3.2 Allocation leximin (lex)

Au-delà de l'utilitarisme, une façon de mettre en œuvre une allocation équitable et Pareto-optimale est de considérer la règle *leximin* qui sélectionne, parmi toutes les allocations possibles, une allocation conduisant aux meilleurs profils d'utilité par rapport à l'ordre leximin [13]. Plus précisément, soit $z = [z_1, \ldots, z_n]$ le vecteur d'utilité où chaque composante $z_a \in [0, Z_a]$ représente l'utilité pour l'agent $a \in \mathcal{A}$. Z_a désigne ici la meilleure valeur d'utilité pour l'utilisateur a considéré seul, c'est-à-dire pour le problème mono-agent où le meilleur chemin peut être choisi pour chaque graphe $g \in \mathcal{G}_a$. En optimisation selon le leximin, l'objectif est de maximiser lexicographiquement le vecteur $\Lambda = [\Lambda_1, \ldots, \Lambda_n]$ obtenu après avoir ordonné $[z_1, \ldots, z_n]$ suivant un ordre croissant.

Cette règle leximin peut être mise en œuvre par une séquence de programmes linéaires [10]. Nous adaptons ici une telle procédure au cas spécifique des PADAGs. Supposons que nous ayons déjà optimisé sur les premières $K-1$ composantes $[\Lambda_1, \ldots, \Lambda_{K-1}]$ de Λ, pour $K \in [1..n]$. Ensuite, on peut utiliser le programme présenté par la suite pour optimiser la $K^{\text{ième}}$ composante Λ_K du profil leximin de z. Dans ce modèle, λ représente l'utilité obtenue au niveau K dans Λ, avec $\lambda \in [\Lambda_{K-1}, \max_{a \in \mathcal{A}} Z_a]$ et par convention $\Lambda_0 = 0$. y_{ak} est une variable binaire égale à

1 si l'agent $a \in \mathcal{A}$ joue le rôle de l'agent associé au niveau $k \in [1..K-1]$ dans $[\Lambda_1, \ldots, \Lambda_{K-1}]$, 0 sinon. L'optimisation de Λ_K peut être effectuée à l'aide du programme $P_{\mathsf{lex}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle, K, [\Lambda_1, \ldots, \Lambda_{K-1}])$ donné ci-dessous :

$$\textbf{max} \quad \lambda \tag{8}$$

$$\text{t.q.} \quad (1), (2), (3), (4), (5), (6)$$

$$z_a = \sum_{g \in \mathcal{G}_a} \sum_{e \in E_g} u_g(e) \cdot x_e, \quad \forall a \in \mathcal{A} \tag{9}$$

$$\sum_{a \in \mathcal{A}} y_{ak} = 1, \quad \forall k \in [1..K-1] \tag{10}$$

$$\sum_{k \in [1..K-1]} y_{ak} \leq 1, \quad \forall a \in \mathcal{A} \tag{11}$$

$$\lambda \leq z_a + M \sum_{k \in [1..K-1]} y_{ak}, \quad \forall a \in \mathcal{A} \tag{12}$$

$$z_a \geq \sum_{k \in [1..K-1]} \Lambda_k \cdot y_{ak}, \quad \forall a \in \mathcal{A} \tag{13}$$

Dans la contrainte (12), $M = \max_{a \in \mathcal{A}} Z_a$ est utilisé pour ignorer les agents associés à des niveaux strictement inférieurs à K lors de l'optimisation de λ (formulation big-M). La contrainte (13) garantit que l'utilité obtenue pour l'agent associé au niveau $k \in [1..K-1]$ ne soit pas être inférieure à Λ_k. Pour mettre en œuvre la règle du leximin, il suffit alors de résoudre une séquence de problèmes P_{lex} pour $K \in \mathcal{A}$ afin d'optimiser la valeur de chaque composante du profil d'utilité.

Algorithme 1 : Algorithme leximin (lex)

Données : Un problème PADAG $\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle$
Résultat : Une allocation leximin-optimale π

```
pour K = 1 à |A| faire
    (λ*, sol) ←
      résoudre P_lex(⟨A, G, μ, C⟩, K, [Λ_1, ..., Λ_{K-1}])
    Λ_K ← λ*
pour g ∈ G faire
    π(g) ← {v ∈ V_g | sol(β_v) = 1}
retourner π
```

Exemple 3. L'allocation leximin optimale pour l'exemple de la figure 1 est $\pi_{\mathsf{lex}} = \{a \mapsto \{s_a, a_1, a_4, t_a\}, b \mapsto \{s_b, b_2, b_3, t_b\}\}$, avec une utilité globale $u(\pi_{\mathsf{lex}}) = u(\pi_{\mathsf{lex}}(a)) + u(\pi_{\mathsf{lex}}(b)) = 0.62 + 0.70 = 1.32$ et le vecteur d'utilité $(0.62, 0.70)$.

3.3 Allocation leximin approchée (a-lex)

Le modèle précédent met en œuvre une règle leximin exacte et assure donc l'équité de l'allocation résultante, mais il peut difficilement passer à l'échelle avec l'augmentation du nombre d'agents et d'arêtes. Notons que la vérification des tests classiques de répartition équitable (proportionnalité, absence d'envie...) est impossible puisque les agents n'enchérissent pas sur le même ensemble d'articles. Nous proposons ainsi une version approximative du calcul du leximin, basée sur un schéma maximin itéré. Fondamentalement, cette approche considère à chaque étape une utilité minimale $\Delta_a \geq 0$ pour certains agents et maximise la pire utilité parmi les agents restants, pour lesquels nous supposons arbitrairement $\Delta_a = -1$. Le problème à résoudre, appelé $P_{\mathsf{a\text{-}lex}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle, \Delta)$, est le suivant :

$$\textbf{max} \quad \delta \tag{14}$$

$$\text{t.q.} \quad (1), (2), (3), (4), (5), (6)$$

$$\delta \leq \sum_{g \in \mathcal{G}_a} \sum_{e \in E_g} u_g(e) x_e, \quad \forall a \in \mathcal{A} \mid \Delta_a = -1 \tag{15}$$

$$\sum_{g \in \mathcal{G}_a} \sum_{e \in E_g} u_g(e) x_e \geq \Delta_a, \quad \forall a \in \mathcal{A} \mid \Delta_a \neq -1 \tag{16}$$

La méthode de résolution consiste alors à optimiser $P_{\mathsf{a\text{-}lex}}$ de manière itérative, comme pour leximin. Comme l'indique l'algorithme 2, à chaque itération (une par agent), $P_{\mathsf{a\text{-}lex}}$ est résolu, un pire agent $\hat{a}$ est déterminé, et son utilité minimale $\Delta_{\hat{a}}$ est fixée.

Algorithme 2 : Algorithme leximin approché (a-lex)

Données : Un problème PADAG $\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle$
Résultat : Une allocation maximin-optimale π

```
1 Δ ← [−1, ..., −1]
2 pour K = 1 à |A| faire
3     (δ*, sol) ← résoudre P_a-lex(⟨A, G, μ, C⟩, Δ)
4     S ← argmin_{a∈A | Δ_a=−1} Σ_{g∈G_a} Σ_{e∈E_g} u_g(e) sol(x_e)
5     â ← choisir un agent a ∈ S
6     Δ_â ← δ*
7 pour g ∈ G faire
8     π(g) ← {v ∈ V_g | sol(β_v) = 1}
9 retourner π
```

La principale différence avec P_{lex} est qu'à chaque itération, dans $P_{\mathsf{a\text{-}lex}}$, la position d'un agent dans l'ordre est déterminée une fois pour toutes, alors que dans P_{lex} l'ordre peut être révisé à chaque itération. De plus, si une égalité se produit à la ligne 5 pour déterminer le pire agent (cas $|S| > 1$), on peut se fier à une certaine heuristique ou à un choix arbitraire. Ainsi, $P_{\mathsf{a\text{-}lex}}$ est une approximation de P_{lex} qui contient moins de variables et de contraintes.

Exemple 4. L'allocation leximin approchée pour l'exemple de la figure 1 est $\pi_{\mathsf{a\text{-}lex}} = \{a \mapsto \{s_a, a_1, a_4, t_a\}, b \mapsto \{s_b, b_2, b_3, t_b\}\}$, avec le vecteur d'utilité $(u(\pi_{\mathsf{a\text{-}lex}}(a)), u(\pi_{\mathsf{a\text{-}lex}}(b))) = (0.62, 0.70)$ et une utilité globale $u(\pi_{\mathsf{a\text{-}lex}}) = 0.62 + 0.70 = 1.32$. C'est la même chose que π_{lex} puisqu'il n'y a que deux agents et aucune égalité entre les pires utilités.

3.4 Allocation gloutonne (greedy)

Pour les décisions très rapides, le maximin itéré peut encore être trop lent. Dans de tels cas, une approche gloutonne peut fournir des allocations valides très rapidement. L'idée principale de l'allocation de chemins par approche gloutonne est d'itérer sur l'ensemble des graphes. À chaque étape, un

graphe g^* qui a le meilleur chemin d'utilité est sélectionné, ce chemin est choisi comme $\pi(g^*)$, et tous les nœuds des autres graphes qui sont en conflit avec les nœuds de $\pi(g^*)$ sont désactivés. Le graphe g^* est alors supprimé, et le processus continue jusqu'à ce qu'il n'y ait plus de graphe à considérer. Ce processus garantit le respect des contraintes (1), (2), (3), (4), (5), (6).
Ce processus est décrit dans l'algorithme 3.

Algorithme 3 : Algorithme glouton (greedy)

Données : Un problème PADAG $\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle$
Résultat : Une allocation π

tant que $\mathcal{G} \neq \emptyset$ **faire**
  déterminer g^* le graphe d'utilité maximale avec le chemin p
  $\pi(g^*) \leftarrow p$
  pour $g \in \mathcal{G}$ **faire**
    $V_g \leftarrow \{v \in V_g \mid \forall w \in \pi(g^*), \{v, w\} \notin \mathcal{C}\}$
    $E_g \leftarrow \{(v, w) \in E_g \mid v \in V_g, w \in V_g\}$
  $\mathcal{G} \leftarrow \mathcal{G} \setminus \{g^*\}$
retourner π

Déterminer le meilleur chemin dans un DAG g est linéaire en temps : $\mathcal{O}(|E_g| + |V_g|)$ [4]. De toute évidence, la méthode gloutonne est équivalente à la méthode utilitaire lorsqu'il n'y a pas de conflit entre les graphes. En effet, greedy retournera le meilleur chemin pour chaque graphe, ce qui est la meilleure solution utilitaire dans un tel contexte. De plus, cette approche aboutit à un équilibre de Nash où aucun agent ne peut améliorer son utilité sans un impact négatif sur les autres agents. Ceci est équivalent à la procédure *Nashify* de [16] dans le contexte des jeux de congestion, avec un seul tour. Nous verrons dans les expériences que cet équilibre est loin d'être équitable.

Exemple 5. L'allocation gloutonne pour l'exemple de la figure 1 est $\pi_{\mathsf{greedy}} = \{a \mapsto \{s_a, a_1, a_3, t_a\}, b \mapsto \{s_b, b_2, b_4, t_b\}$, avec une utilité globale $u(\pi_{\mathsf{greedy}}) = u(\pi_{\mathsf{greedy}}(a)) + u(\pi_{\mathsf{greedy}}(b)) = 1.0 + 0.3 = 1.3$ et le vecteur d'utilité $(1.0, 0.3)$.

3.5 Allocations *round-robin* (p-rr et n-rr)

Une approche rapide pour l'allocation équitable de biens indivisibles est le *round-robin*. Elle consiste à faire en sorte que chaque agent choisisse à tour de rôle (dans un ordre fixe prédéfini) un élément (en fonction de ses préférences) jusqu'à ce qu'il n'y ait plus d'élément à allouer. Comme greedy, il est polynomial en nombre d'agents et d'articles. Dans notre cas, nous pouvons considérer deux types d'articles à allouer : les chemins (approches notée p-rr) ou les nœuds (approche notée n-rr). Dans le cas des chemins, chaque agent choisit à son tour le meilleur chemin possible, compte tenu des nœuds déjà alloués (pour éviter les conflits). Ce processus fonctionne de manière similaire à celui de greedy, mais alterne entre les utilisateurs pour équilibrer les utilités. Dans le cas des nœuds, chaque agent construit de manière incrémentale le chemin associé à chacun de ses graphes, en choisissant tour à tour son meilleur prochain nœud réalisable jusqu'à ce qu'il atteigne le puits ou qu'il n'y ait plus de nœud réalisable à choisir (un chemin sans issue). Dans ce dernier cas, l'agent se voit attribuer le chemin source-puits d'utilité 0 et perd les nœuds précédemment choisis. Dans les deux approches, les contraintes (1), (2), (3), (4), (5), (6) sont respectées puisque les chemins considérés sont tous faisables. Notons que p-rr résulte en un équilibre de Nash où chaque agent s'est vu allouer le meilleur chemin compte tenu des autres allocations. Ce n'est pas le cas pour n-rr, puisque certains nœuds laissés par un agent tombant dans une impasse peuvent avoir empêché d'autres agents de trouver une meilleure solution.

Exemple 6. L'allocation *round-robin* par chemin $\pi_{\mathsf{p\text{-}rr}}$ pour l'exemple de la figure 1 est équivalente à π_{greedy}, puisque a choisit $\{s_a, a_1, a_3, t_a\}$ et ensuite b choisit $\{s_b, b_2, b_4, t_b\}\}$. L'allocation *round-robin* par nœud $\pi_{\mathsf{n\text{-}rr}}$ est également équivalente à π_{greedy} car a choisit d'abord a_1, puis b choisit b_2 (seule option réalisable), puis a choisit a_3 (meilleure option), et enfin b choisit b_4 (seule option réalisable).

4 Evaluation expérimentale

Dans cette section, nous évaluons les performances des méthodes d'allocation proposées pour résoudre un problème d'allocation de portions d'orbite [1] codé en PADAGs. Nous présentons le dispositif expérimental et analysons quelques résultats obtenus sur des instances synthétiques.

4.1 Scénario d'évaluation

Comme décrit dans [18], les constellations de satellites d'observation de la Terre soulèvent de nombreux défis. Nous abordons ici celui dans lequel des utilisateurs peuvent demander a priori l'exclusivité sur des portions d'orbite, afin de disposer de temps satellite pour réaliser des prises de vue sans avoir à passer systématiquement par l'arbitrage d'un opérateur central. L'objectif de notre étude est d'allouer des portions d'orbite aux utilisateurs, sachant que chacun a des requêtes d'observation définies par un point d'intérêt (POI) à observer avec une fréquence de revisite donnée, par exemple observer la ville de Saint-Étienne toutes les 2 heures pendant plusieurs jours. Étant donné que plusieurs satellites peuvent capturer le même point sur la Terre autour des plots temporels définis, plusieurs lots sont spécifiés par chaque utilisateur, qui se valorisent différemment en fonction de la qualité de la séquence des portions d'orbite, par exemple la proximité des plots temporels ou les angles d'acquisition. En outre, plusieurs utilisateurs peuvent être intéressés par des points d'intérêt très proches, ce qui entraîne un chevauchement des portions d'orbite qui ne peuvent être attribués simultanément aux utilisateurs correspondants.

1. qui est un problème assez récent identifié mais non formalisé dans [17], à ne pas confondre avec les problèmes d'ordonnancement d'orbite [11].

FIGURE 2 – Un problème d'allocation de portions d'orbites impliquant 2 agents (a en rouge, b en bleu) demandant des portions d'orbite autour de deux plots temporels (t_1 et t_2), avec des fenêtres de tolérance autour de chaque plot (en gris) et deux portions candidates pour chaque plot ($a_1, \ldots, a_4, b_1, \ldots, b_4$).

4.2 Cadre expérimental

Nous considérons une constellation en orbite basse (500 km d'altitude) composée de n_p plans orbitaux régulièrement espacés ayant une inclinaison de 60 degrés, avec $n_p \in \{2, 4, 8, 16\}$ et 2 satellites régulièrement espacés sur chaque plan orbital. Pour générer des instances PADAG, nous générons aléatoirement des requêtes pour 4 agents souhaitant obtenir la propriété de portions d'orbite afin de réaliser des acquisitions au sol répétitives de points d'intérêt (POI) appartenant à la même zone. Les POIs sont choisis aléatoirement dans un sous-ensemble extrait de [15]. Tous les agents ont le même modèle pour une requête r : obtenir une observation tous les jours à 8 :00 + δ_r, 12 :00 + δ_r, et 16 :00 + δ_r, avec une tolérance de 1 heure autour de chaque plot temporel, et un décalage aléatoire uniforme $\delta_r \in [-2, 2]$ pour tous les plots temporels de la même requête. Pour chaque POI et chaque plot temporel, les portions d'orbite candidates sont déterminées grâce à une librairie logicielle de mécanique spatiale, en partant de l'hypothèse qu'un satellite est pertinent pour un POI dès que son élévation au-dessus de l'horizon est supérieure à 15 degrés. Les portions d'orbite incompatibles sont celles qui se chevauchent alors qu'elles appartiennent au même satellite.

Nous formulons ensuite ces requêtes et les portions d'orbite candidates sous forme de PADAGs. Chaque requête est représentée par un graphe, dans lequel les nœuds (à l'exception de la source et du puits) sont des portions d'orbite pour capturer un POI à un moment donné, et les arêtes relient deux portions d'orbite consécutives pour répondre à la requête. Par exemple, la figure 1 représente un PADAG pour deux requêtes de deux utilisateurs (a et b), avec deux plots temporels. Chaque utilisateur a deux portions d'orbite candidates par plot temporel, illustré en figure 2. Pour simplifier, nous ne considérons que les utilités attachées aux portions d'orbite, sans prendre en considération les transitions entre elles. Nous étudions une fonction d'utilité *linéaire*, qui est linéaire en fonction de la distance entre le milieu τ de la portion allouée et le plot temporel demandé (utilité linéairement décroissante de 1 lorsque τ est exactement sur le plot à 0 lorsque τ atteint les limites de la fenêtre de tolérance). Nous normalisons chaque utilité par rapport à l'utilité maximale qui peut être obtenue pour chaque graphe individuellement. Nous considérons 2 requêtes par agent, et un horizon de 365 jours, ce qui donne des DAGs ayant 1095 couches. Ce paramètre donne des DAGs avec les propriétés suivantes en moyenne :

n_p	2	4	8	16
largeur	3.08	5.41	10.05	19.38
conflits	26798.80	45636.06	82971.20	158180.20
durée (s)	603.28	600.10	599.87	598.75

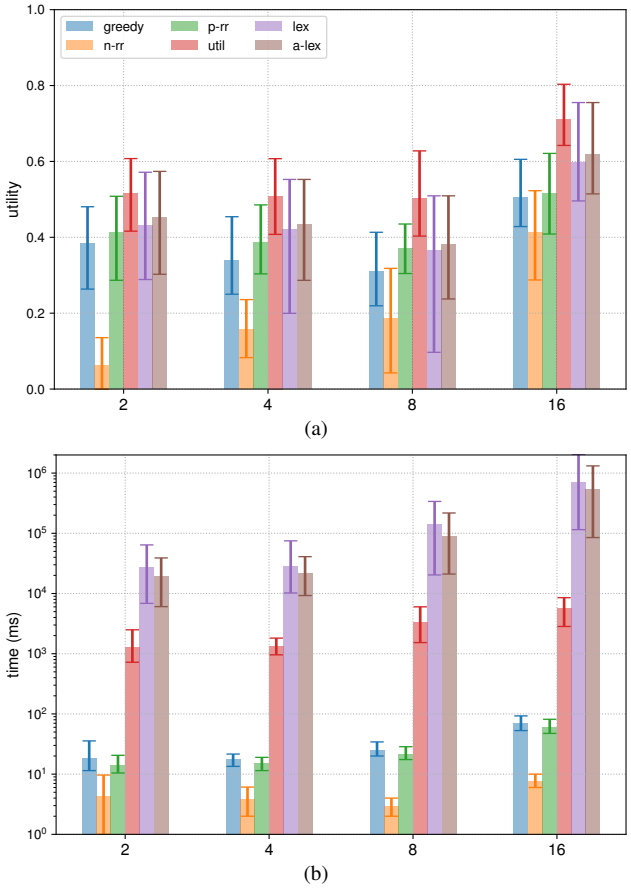

FIGURE 3 – Utilité globale moyenne (en haut) et temps de calcul (en bas) obtenus pour chaque algorithme et chaque taille de constellation.

En bref, la largeur du DAG (nombre de nœuds par couche) et le nombre de conflits augmentent proportionnellement avec le nombre de satellites. La durée des portions d'orbite est d'environ 10 minutes.

Les solveurs sont codés en Java 1.8 et exécutés sur un CPU Intel(R) Xeon(R) E5-2660 v3 @ 2.60GHz à 20 cœurs, 62GB RAM, Ubuntu 18.04.5 LTS. util, lex et a-lex utilisent l'API Java de IBM CPLEX 20.1 (avec un temps limite de 10 minutes). Nous avons exécuté sur 30 instances de PADAGs générés aléatoirement et tracé la moyenne avec une confiance de $[0.05, 0.95]$.

4.3 Utilité

La figure 3a montre l'utilité globale normalisée moyenne pour chaque algorithme et chaque taille de constellation (exprimée en nombre de plans orbitaux). L'utilité globale normalisée est l'utilité moyenne du graphe, donc entre 0 et 1. De toute évidence, util fournit l'allocation utilitaire optimale. En deuxième position, a-lex fournit de bonnes allo-

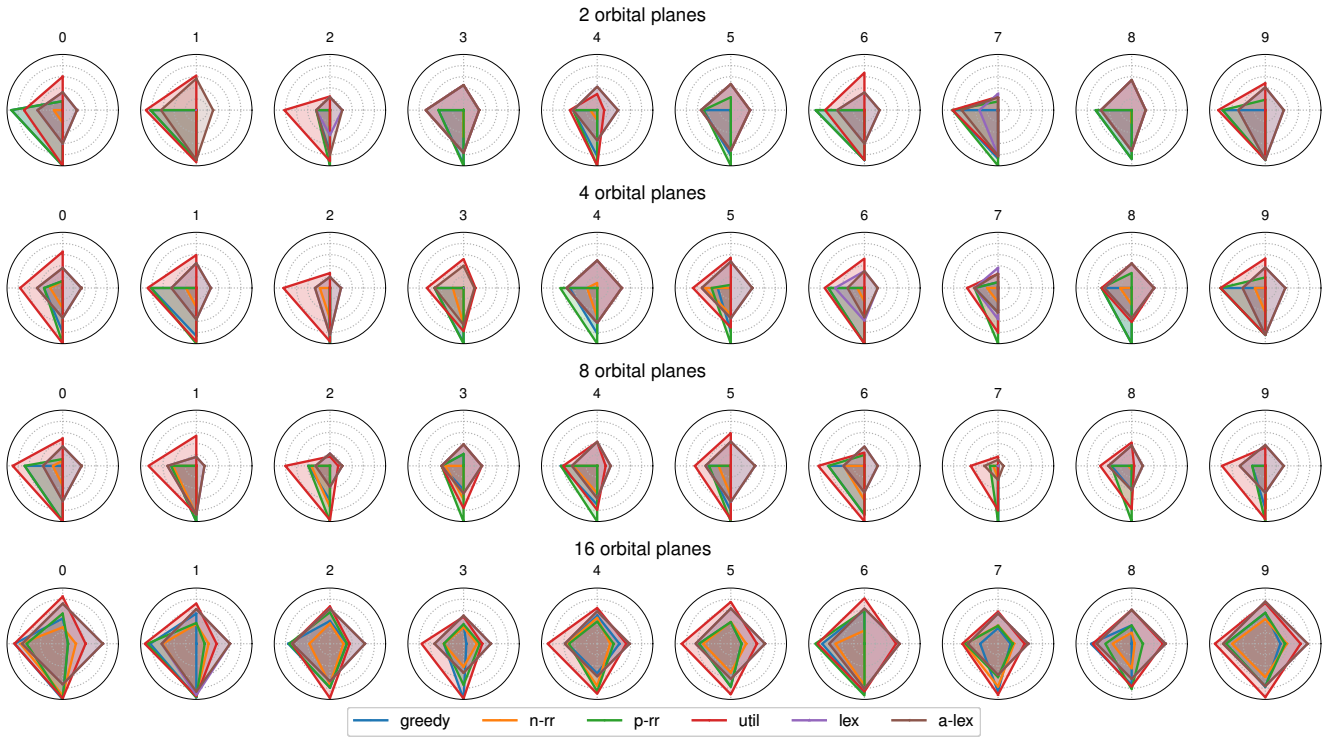

FIGURE 4 – Profils d'utilité (par ordre leximin) pour les 10 premières instances (sur 30) pour chaque taille de constellation et chaque algorithme (sud : meilleure utilité sur tous les agents ; ouest : deuxième meilleure utilité ; nord : troisième meilleure utilité ; est : pire utilité).

cations à presque 85% de la valeur optimale en moyenne. Il est intéressant de noter que lex a des performances équivalentes à a-lex (écart inférieur à 5% en moyenne). En effet, a-lex fournit des allocations légèrement meilleures du point de vue utilitaire, au détriment de l'approximation de l'équité. Les approches *round-robin* diffèrent vraiment en termes d'utilité. Bien que p-rr fournit des allocations à presque 71% de l'optimal, n-rr donne des allocations de faible utilité, à presque 10% de l'optimal sur des constellations plus petites. D'ailleurs, dans de telles configurations, il y a peu de chemins réalisables pour chaque demande. Ainsi, pour la plupart des demandes, la construction incrémentale myope des chemins aboutit à des impasses, et donc à des allocations d'une utilité de 0 ; alors qu'en considérant un autre ordre d'agents, les allocations auraient pu être meilleures. Enfin, greedy se comporte légèrement moins bien que p-rr, à près de 68% de la valeur optimale en moyenne. Sur des instances plus grandes, où de nombreux chemins existent pour répondre à chaque requête, greedy se comporte encore mieux. Plus généralement, les problèmes de constellation de grande taille sont plus faciles à résoudre du point de vue utilitaire par les algorithmes non optimaux, puisqu'il existe plus d'options pour éviter les conflits malgré leur nombre élevé.

4.4 Équité

Pour analyser l'équité des allocations résultantes, la Figure 4 fournit les profils d'utilité obtenus pour les 10 premières instances pour chaque algorithme. greedy, par principe, cherche à allouer d'abord les chemins à plus forte utilité, ce qui entraîne des allocations injustes où seuls les 2 ou 3 premiers utilisateurs sont servis, alors que plus souvent le quatrième utilisateur n'a aucune requête satisfaite. Les approches *round-robin* sont plus équitables que l'approche greedy et satisfont souvent les requêtes d'un plus grand nombre d'utilisateurs. util donne des profils avec la plus grande surface, mais la plupart du temps le quatrième utilisateur est négligé. Enfin, lex et a-lex se comportent presque identiquement (leurs profils sont superposés), montrant que l'approximation a-lex est suffisante pour produire des allocations équitables. Notons qu'avec des constellations plus grandes, puisqu'il y a de plus en plus d'options pour servir les utilisateurs, tous les algorithmes ont tendance à donner des allocations plus équitables. Néanmoins, lex et a-lex sont les meilleurs choix ici, et les concurrents *round-robin* produisent des allocations insatisfaisantes.

4.5 Temps de calcul

La figure 3b montre le temps de calcul moyen en millisecondes pour chaque algorithme et chaque taille de constellation. Comme prévu (par conception), greedy, p-rr et n-rr sont les plus rapides. n-rr qui n'effectue même pas les opérations de chemin le plus court, est de loin le plus rapide, mais donne lieu à des allocations très mauvaises et peu équitables. greedy et p-rr sont très rapides, mais sont basés sur de multiples recherches du chemin maximum dans les DAGs. p-rr fournit encore rapidement d'assez bonnes allocations utilitaires et équitables. util, basé sur la résolution d'un seul programme linéaire en nombres entiers, est 100 fois plus lent que les algorithmes les plus rapides. Ensuite,

lex et a-lex sont jusqu'à deux ordres de grandeur plus lents que util sur les plus grandes constellations. Ceci est dû aux multiples appels ($|\mathcal{A}|$) au solveur MILP sur les grands problèmes. a-lex est 2 à 3 fois plus rapide que lex puisqu'il résout des MILP plus petits, tout en donnant des allocations aussi justes que celles de lex.

5 Conclusion

Dans cet article, nous avons proposé le modèle PADAG, un nouveau problème d'allocation où les agents expriment leurs préférences sur des lots d'articles sous forme de DAGs pondérés par les arêtes. Nous avons introduit et analysé plusieurs méthodes de résolution (utilitaire, leximin, leximin approché, glouton) contre les allocations *round-robin*, du point de vue de l'utilitarisme et de l'équité. Nous avons évalué ces méthodes sur de grandes instances de problèmes d'allocation de créneaux orbitaux, générées aléatoirement, avec plus de 1000 couches. Sur de grandes constellations, nous observons que la méthode leximin approché constitue un bon compromis entre l'optimalité utilitaire, l'optimalité leximin et le temps de calcul, par rapport à toutes les autres méthodes de solution. Elle est même équivalente au leximin exact sur la plupart des instances, tout en divisant les temps de calcul par un facteur de 2 à 3.

Nous identifions plusieurs pistes pour de futures investigations. Tout d'abord, comme le leximin approximatif est un compromis entre l'équité et les temps de calcul, nous aimerions étudier d'autres méthodes de résolution, encore plus réactives, notamment pour résoudre des PADAGs plus grands et des topologies spécifiques. En effet, comme pour la division des chemins et les jeux de congestion, des techniques dédiées pourraient être conçues pour des topologies données (étoiles, chaînes, etc.). Deuxièmement, puisque les PADAGs sont fortement contraints par les conflits, nous cherchons à explorer des heuristiques *min-conflict* pour améliorer nos algorithmes. Enfin, nous pensons que les PADAGs ont un grand potentiel pour être utilisés dans une variété de domaines, et nous souhaitons donc évaluer les techniques proposées sur des problèmes provenant d'autres applications, comme le domaine NFV où les chaînes de fonctions sont modélisées comme des graphes, et les incompatibilités contrôlent l'accès aux nœuds ou le domaine de la recherche de chemins dans un cadre multi-agent (les préférences de chemin sont modélisées comme des graphes, et les incompatibilités modélisent les contraintes pour éviter que deux agents occupent la même position en même temps).

Références

[1] Sylvain Bouveret, Katarína Cechlárová, Edith Elkind, Ayumi Igarashi, and Dominik Peters. Fair division of a graph. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 135–141. ijcai.org, 2017.

[2] Coen Bron and Joep Kerbosch. Algorithm 457 : Finding all cliques of an undirected graph. *Commun. ACM*, 16(9) :575–577, September 1973.

[3] Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum nash welfare. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, page 305–322, New York, NY, USA, 2016. Association for Computing Machinery.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[5] Peter Cramton, Yoav Shoham, and Richard Steinberg. *Combinatorial Auctions*. MIT Press, 2005.

[6] Ye Du, Rahul Sami, and Yaoyun Shi. Path auctions with multiple edge ownership. *Theor. Comput. Sci.*, 411(1) :293–300, 2010.

[7] Michael S. Hughes, Brian J. Lunday, Jeffrey D. Weir, and Kenneth M. Hopkinson. The multiple shortest path problem with path deconfliction. *Eur. J. Oper. Res.*, 292(3) :818–829, 2021.

[8] Ayumi Igarashi and Dominik Peters. Pareto-optimal allocation of indivisible goods with connectivity constraints. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 2045–2052. AAAI Press, 2019.

[9] Nicole Immorlica, David R. Karger, Evdokia Nikolova, and Rahul Sami. First-price path auctions. In John Riedl, Michael J. Kearns, and Michael K. Reiter, editors, *Proceedings 6th ACM Conference on Electronic Commerce (EC-2005), Vancouver, BC, Canada, June 5-8, 2005*, pages 203–212. ACM, 2005.

[10] David Kurokawa, Ariel D. Procaccia, and Nisarg Shah. Leximin allocations in the real world. *ACM Transactions on Economics and Computation*, 6(3–4), 2018.

[11] M. Lemaître, G. Verfaillie, H. Fargier, J. Lang, N. Bataille, and J.-M. Lachiver. Equitable allocation of earth observing satellites resources. In *5th ONERA-DLR Aerospace Symposium (ODAS'03)*, 2003.

[12] Nimrod Megiddo. Optimal flows in networks with multiple sources and sinks. *Math. Program.*, 7(1) :97–107, 1974.

[13] Hervé Moulin. *Fair division and collective welfare*. MIT Press, 2003.

[14] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, USA, 2007.

[15] OpenStreetMap. Openstreetmap points of interest (on french territory). https://www.data.gouv.fr/fr/datasets/points-dinterets-openstreetmap/, 2021. Accessed : 30-08-2021.

[16] Panagiota N. Panagopoulou and Paul G. Spirakis. Algorithms for pure nash equilibria in weighted congestion games. *ACM J. Exp. Algorithmics*, 11 :2.7–es, feb 2007.

[17] Gauthier Picard. Auction-based and Distributed Optimization Approaches for Scheduling Observations in Satellite Constellations with Exclusive Orbit Portions. In *International Workshop on Planning and Scheduling for Space (IWPSS'21)*, 2021.

[18] Gauthier Picard, Clément Caron, Jean-Loup Farges, Jonathan Guerra, Cédric Pralet, and Stéphanie Roussel. Autonomous Agents and Multiagent Systems Challenges in Earth Observation Satellite Constellations. In U. Endriss, A. Nowé, F. Dignum, and A. Lomuscio, editors, *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 39–44, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems.

[19] Jordi Ferrer Riera, Eduard Escalona, Josep Batallé, Eduard Grasa, and Joan A. García-Espín. Virtual network function scheduling : Concept and challenges. In *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, pages 1–5, 2014.

[20] Song Yang, Fan Li, Stojan Trajanovski, Ramin Yahyapour, and Xiaoming Fu. Recent advances of resource allocation in network function virtualization. *IEEE Transactions on Parallel and Distributed Systems*, 32(2) :295–314, 2021.

[21] Lei Zhang, Haibin Chen, Jun Wu, Chong-Jun Wang, and Junyuan Xie. False-name-proof mechanisms for path auctions in social networks. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1485–1492. IOS Press, 2016.