Bundle allocation with conflicting preferences represented as weighted directed acyclic graphs : application to orbit slot ownership

Sara Maqrot, Stéphanie Roussel, Gauthier Picard, Cédric Pralet

Bundle Allocation with Conflicting Preferences Represented as Weighted Directed Acyclic Graphs

Application to Orbit Slot Ownership

Sara Maqrot, Stéphanie Roussel, Gauthier Picard, and Cédric Pralet

ONERA/DTIS, Université de Toulouse
firstname.lastname@onera.fr

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.

1 Introduction

Imagine the following scenario. The operator of an Earth Observation Satellite Constellation X has to attribute ownership of some orbit portions to clients. Each client has some points of interest (POI) she wishes to acquire at some frequency, e.g. capture L'Aquila city every 2 hours for 6 months. Since several satellites may capture the very same point on Earth around the defined time plots, several bundles are specified by each client, which valuate differently depending on the quality of the sequence of orbit slot, e.g. proximity to the time plots or acquisition angles. Moreover, several clients may be interested in very close POIs, resulting in overlapping orbit slots that cannot be simultaneously allocated to the corresponding clients. This situation can be captured by the model we propose in this paper.

We consider a problem of allocation of bundles of indivisible items constrained by item chaining (to allocate to each agent a chain of successive items) and conflicting items. The first constraint is captured by an edge-weighted directed acyclic graph (DAG), with a source node and a sink node, representing all the valid bundles (i.e. paths) of items for an agent, where the quality of a bundle is represented by additive edge weights. The second constraint has to be handled so that each agent obtains one conflict-free path in her graph. Such a setting occurs in application domains such as network function virtualization (NFV) where users request to allocate directed graphs of services into a shared networked infrastructure [17], or in Earth observation using a constellation of satellites, where users demand the ownership of some repetitive orbit slots (without overlapping with other users' slots) to implement periodic observation requests [9, 15], as illustrated before. In such settings, beside the additive edge weights, other criteria can be considered to guide the allocation process, especially when constellation users are stakeholders expecting allocations to be fair or proportional to their investment.

Fig. 1: An orbit slot allocation problem involving 2 agents (a in red, b in blue) requesting slots around 2 time plots (t_1 and t_2), tolerance windows around each plot (in gray), and with 2 opportunities for each plot ($a_1, \ldots, a_4, b_1, \ldots, b_4$).

Related Work. Literature contains some work related to allocation of goods structured as graphs. In fair division of graph, the objective is to divide a graph of items between several agents, with additive utilities attached to nodes [1, 6]. These works provide interesting properties to find envy-free or Pareto-optimal allocations, in an efficient manner in some specific graph structures, e.g. paths, trees, stars. However, in our problem, (i) agents do not compete for the very same set of items, (ii) the graph is directed to compose paths from a start time to an end time, (iii) even by mapping our problem to a graph division one and by regrouping conflicting items into composite items, it is highly improbable that the resulting graph is acyclic. Here, graphs are used to express preferences, and not the goods to allocate. In short, our work does not fall into the existing graph fair division frameworks, or cannot benefit from theoretical results on path-shaped or star-shaped graphs.

Another related work is path auctions [7, 4, 18], where agents bid for paths in a graph where each edge is owned by an agent. The goal is to assign paths to agents by the means of auctions, and optionally to keep some privacy for the edge owners. In the case of a utilitarian objective function for the winner determination problem, without price privacy, this falls into the Vickrey-Clarke-Groves framework, and thus guarantees some efficient and *strategyproof* mechanisms. But, here again, agents bid on the very same set of nodes and edges.

In the transportation domain, investigations on very similar structures, that is flow networks, provide techniques for fair maximum flow in multi-source and multi-sink networks [10]. While the techniques used are very similar to ours (linear programming), the maximum flow objective is very different from path utility maximization. Besides, [5] worked on multiple shortest path problems based on deconflicting techniques. While the problem displays similar characteristics, once again the agents evolve on the very same graphs, and the objective is focused on minimizing path length and minimizing conflicting paths, without fairness desiderata.

In congestion games, agents are allocated paths so that delay incurred by crossing paths are minimized. The more agents are allocated the same nodes, the more delay is attached to their paths [12, 14]. In our work, we don't consider delay but incompatibilities. Even if they could be modeled as non linear $\{0,\infty\}$ functions, in our problem some path allocations are unfeasible, contrarily to congestion games. Besides, using congestion game solution methods as in [14] may result in unfair Nash equilibria, because of numerous unfeasible paths.

More generally, another classical approach to fair allocation of indivisible goods is *round-robin*, which is almost envy-free [2]. This is notably one favored technique to allocate virtual network functions in network function virtualization infrastructures [16], or to schedule tasks. We will use it as a competitor for our techniques.

Contributions. This paper introduces and expounds a model for such scenarios, under the prism of optimality and fairness, which captures any application where agents express

preferences as edge-weighted DAGs and where there exist conflicts between some nodes of these DAGs. We show that this allocation problem is NP-hard. We expound and assess several algorithms on data coming from simulated satellite constellations and requests, with respect to utilitarian optimality, computation time, and fairness.

2 Problem Model

We study allocation problems where agents' valuations of item bundles are represented as edge-weighted DAGs and some conflicts exists between nodes of these graphs. An allocation consists in choosing one full path in each graph, so that selected paths do not conflict with each other.

2.1 Definitions and Notations

Definition 1. *A problem of* path allocation in multiple conflicting edge-weighted directed acyclic graphs *(PADAG) is a tuple* $\langle \mathcal{A},\mathcal{G},\mu,\mathcal{C} \rangle$*, where*

- $\mathcal{A}=\{1,...,n\}$ *is a set of* agents*;*
- $\mathcal{G}=\{g_1,...,g_m\}$ *is a set of edge-weighted DAGs, where each* $g \in \mathcal{G}$ *is a triple* $\langle V_g,E_g,u_g \rangle$ *that represents some preferences over some items in* V_g*, with connections between items in* $E_g \subset V_g \times V_g$*, weighted using utility function* $u_g : V_g \times V_g \to \mathbb{R}$*; we also assume that* V_g *contains two specific nodes, the source* s_g *and the sink* t_g*, and that* E_g *contains an edge from* s_g *to* t_g *labeled by utility* 0 *(useful to deal with cases where no bundle of items can be selected in* g*);*
- $\mu : \mathcal{G} \to \mathcal{A}$ *maps each graph* g *in* $\mathcal{G}$ *to its owner* a *in* $\mathcal{A}$*;*
- $\mathcal{C} \subseteq \{(v,v') | (v,v') \in V_g \times V_{g'}, g,g' \in \mathcal{G}^2, \mu(g) \neq \mu(g')\}$ *is a set of conflicts between pairs of items from two distinct graphs in* $\mathcal{G}$ *from two distinct agents.*

For each graph g and each set of edges $X \subseteq E_g$, the utility of X for g is defined by $u_g(X) = \sum_{e \in X} u_g(e)$, which means that edge valuations are considered as additive in this paper. As a result, each path from s_g to t_g in a graph g is evaluated by summing the utilities of the traversed edges, and each DAG represents in a compact manner a set of valuations for bundles of items, as in combinatorial auctions. Also, we denote by $\mathcal{G}_a = \mu^{-1}(a)$ the set of graphs owned by agent a, and the utility of a set of edges X for agent a is defined by $u_a(X) = \sum_{g \in \mu^{-1}(a)} u_g(X \cap E_g)$.

Definition 2. *An* allocation *is a function* π *that associates, with each graph* $g \in \mathcal{G}$*, one path* $\pi(g)$ *from* s_g *to* t_g *in* g*. Formally,* $\pi(g)$ *can be represented as a set of nodes in* V_g*. Indeed, as DAGs are manipulated, it is easy to reconstruct the edges successively traversed by the path from this set. By extension, the allocation for agent* a *is given by* $\pi(a) = \cup_{g \in \mu^{-1}(a)} \pi(g)$*.*

By convention, we denote by $u(\pi(g)) = u_{\mu(g)}(\pi(g))$ (resp. $u(\pi(a)) = u_a(\pi(a))$), the utility of graph g (resp. agent a) for allocation π. Last, the global utility obtained with allocation π is given by $u(\pi) = \sum_{g \in \mathcal{G}} u(\pi(g))$ (or equivalently $u(\pi) = \sum_{a \in \mathcal{A}} u(\pi(a))$).

Definition 3. *An allocation* π *is* valid *if for each pair of distinct graphs* g *and* g' *there is no conflict between nodes in the resulting paths, i.e.* $(\pi(g) \times \pi(g')) \cap \mathcal{C} = \emptyset$*.*

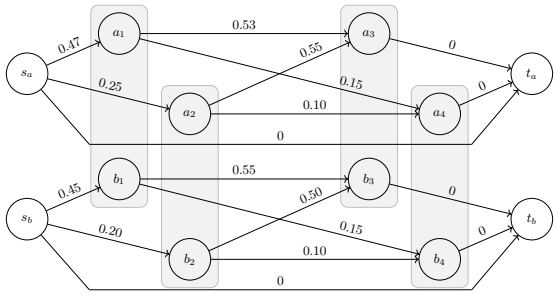

Fig. 2: Sample users' bundle valuations (or preferences) represented as DAGs. Conflicts are represented as gray hypernodes.

Example 1. Figure 2 illustrates a PADAG. While the best paths for agent a and b are $\{s_a, a_1, a_3, t_a\}$ and $\{s_b, b_1, b_3, t_b\}$ respectively, both valued at 1, they cannot both have these paths due to node conflicts, e.g. between a_1 and b_1. A valid allocation could be $\pi_{\mathsf{ex}} = \{a \mapsto \{s_a, a_2, a_4, t_a\}, b \mapsto \{s_b, b_1, b_3, t_b\}\}$ with global utility $u(\pi_{\mathsf{ex}}) = u(\pi_{\mathsf{ex}}(a)) + u(\pi_{\mathsf{ex}}(b)) = 0.35 + 1.0 = 1.35$.

The problems we consider in this paper are (i) *how to compute an optimal (utilitarian) allocation* π that maximizes $u(\pi)$, and (ii) *how to compute an optimal fair allocation* π, by the way of a leximin optimization, *i.e.* maximizing the ordered utility vector $(\Lambda_1,...,\Lambda_n)$ containing one component per agent. Formally, $\forall i \in \mathcal{A}$, there exists a unique j such that $u(\pi(i)) = \Lambda_j$ and $\forall i \leq j$, $\Lambda_i \leq \Lambda_j$.

2.2 Complexity Analysis

The two following propositions provide complexity results on the two problems studied, relatively to utilitarian optimization and leximin optimization respectively.

Proposition 1 *Determining whether there exists a valid allocation π such that utilitarian evaluation $u(\pi)$ is greater than or equal to a given value is NP-complete.*

Proof. First, the problem is NP since $u(\pi)$ is computable in polynomial time. Then, there exists a polynomial reduction of 3-SAT (which is NP-complete) to our problem. Basically, in a 3-SAT formula, each clause over propositional variables x,y,z can be represented as a weighted DAG g where (1) the set of nodes is $V_g = \{x,\neg x,y,\neg y,z,\neg z,s_g,t_g\}$, (2) the set of paths from s_g to t_g in g corresponds to the set of truth-values for x,y,z that satisfy the clause (decision diagram representation), (3) the weight of every edge is set to 0 except for edges $s_g \rightarrow n$ where $n \neq t_g$ that have weight $1/m$, with m the number of clauses in the 3-SAT formula. Last, for every propositional variable x, we can add one conflict (n,n') for each pair of nodes labeled by literals x and $\neg x$ in two distinct graphs. Then, as one path is selected in each graph and as there are m graphs, determining whether there exists a valid allocation π such that $u(\pi) \geq 1$ is equivalent to finding a solution that satisfies all the clauses, hence the NP-completeness result given that all operations used in the transformation are polynomial.

Proposition 2 *It is NP-complete to decide whether there exists a valid allocation whose leximin evaluation is greater than or equal to a given utility vector. The proposition holds even if there is a unique graph per agent.*

Proof. In the general case, it suffices to consider a problem involving a unique agent owning all the graphs, and to use the result of the previous proposition. If there is a unique graph per agent, it suffices to use the exact same 3-SAT encoding as before but to replace weights $1/m$ by weights 1. Then, it is possible to show that there exists a valid allocation whose leximin evaluation is greater than or equal to $(1,1,...,1)$ iff there exists a solution for the 3-SAT problem. Moreover, the leximin evaluation of an allocation π can be computed in polynomial time, hence the NP-completeness result.

3 Path Allocation Schemes

We propose here several allocation schemes for PADAGs. Some of them are based on integer linear programming (ILP) and mixed integer linear programming (MILP), so we first introduce decision variables and constraints for these models.

For any DAG $g=\langle V_g,E_g,u_g\rangle$, we define binary variables $x_e \in \{0,1\}$, for any $e \in E_g$, stating whether edge e is selected in the path defining the solution bundle. We also use auxiliary binary variables β_v stating whether node v is selected in solution path $\pi(g)$, i.e. $\beta_v = 1$ if $v \in \pi(g)$, 0 otherwise. For any node v in V_g, we denote by $\mathsf{In}(v)$ (resp. $\mathsf{Out}(v)$) its set of incoming (resp. outcoming) edges. In all ILP models introduced thereafter, we impose Constraints (1)–(3) to define all the possible paths, Constraints (4)–(5) to account for item selection conflicts, and Constraint (6) to ensure that sources and sinks are selected.

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

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

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

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

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

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

3.1 Utilitarian Allocation (util)

The classical approach to allocation is the utilitarian one. It consists in finding the allocation that maximizes the sum of utilities of all selected paths. This corresponds to solving the integer linear program $P_{\mathsf{util}}(\langle\mathcal{A},\mathcal{G},\mu,\mathcal{C}\rangle)$ given below:

$$\textbf{max} \quad \sum_{a\in\mathcal{A}}\sum_{g\in\mathcal{G}_a}\sum_{e\in E_g} u_g(e)\cdot x_e \tag{7}$$

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

$$x_e \in \{0,1\}, \quad \forall a \in \mathcal{A}, \forall g \in \mathcal{G}_a, \forall e \in E_g \tag{8}$$

The resulting allocation π is decoded from the β_v variables. Formally, for all $g \in \mathcal{G}$, $\pi(g) = \{v \in V_g \mid \beta_v = 1\}$.

Example 2. In Figure 2, the utilitarian allocation is $\pi_{\mathsf{util}} = \{a \mapsto \{s_a, a_2, a_3, t_a\}, b \mapsto \{s_b, b_1, b_4, t_b\}\}$, with global utility $u(\pi_{\mathsf{util}}) = u(\pi_{\mathsf{util}}(a)) + u(\pi_{\mathsf{util}}(b)) = 0.80 + 0.60 = 1.40$.

3.2 Leximin Allocation (lex)

Beyond utilitarianism, one way to implement fair allocation and Pareto-optimality is to consider the *leximin* rule that selects, among all possible allocations, an allocation leading to the best utility profiles wrt the leximin order [11]. More precisely, let $z = [z_1, ..., z_n]$ be the utility vector where each component $z_a \in [0, Z_a]$ represents the utility for agent $a \in \mathcal{A}$. Z_a denotes here the best utility value for user a considered alone, i.e. for the mono-agent problem where the best path can be chosen for each graph $g \in \mathcal{G}_a$. In leximin optimization, the objective is to lexicographically maximize vector $\Lambda = [\Lambda_1, ..., \Lambda_n]$ obtained after ordering $[z_1, ..., z_n]$ following an increasing order.

Such a leximin rule can be implemented through a sequence of ILP [8]. We adapt here such a procedure to the specific case of PADAGs. Suppose we have already optimized over the first $K-1$ components $[\Lambda_1, ..., \Lambda_{K-1}]$ of Λ, for $K \in [1..n]$. Then, one can use the MILP presented thereafter to optimize the K^{th} component Λ_K of the leximin profile of z. In this model, λ represents the utility obtained at level K in Λ, with $\lambda \in [\Lambda_{K-1}, \max_{a \in \mathcal{A}} Z_a]$ and by convention $\Lambda_0 = 0$. y_{ak} is a binary variable equal to 1 if agent $a \in \mathcal{A}$ plays the role of the agent associated with level $k \in [1..K-1]$ in $[\Lambda_1, ..., \Lambda_{K-1}]$, 0 otherwise. The optimization of Λ_K can be performed using program $P_{\mathsf{lex}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle, K, [\Lambda_1, ..., \Lambda_{K-1}])$ given below:

$$\textbf{max} \quad \lambda \tag{9}$$

s.t. (1),(2),(3),(4),(5),(6)

$$z_a = \sum_{g \in \mathcal{G}_a} \sum_{e \in E_g} u_g(e) \cdot x_e, \quad \forall a \in \mathcal{A} \tag{10}$$

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

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

$$\lambda \leq z_a + M \sum_{k \in [1..K-1]} y_{ak}, \quad \forall a \in \mathcal{A} \tag{13}$$

$$z_a \geq \sum_{k \in [1..K-1]} \Lambda_k \cdot y_{ak}, \quad \forall a \in \mathcal{A} \tag{14}$$

In Constraint 13, $M = \max_{a \in \mathcal{A}} Z_a$ is used to ignore agents associated with levels strictly lower than K when optimizing λ (big-M formulation). Constraint 14 ensures that the utility obtained for the agent associated with level $k \in [1..K-1]$ must not be less than Λ_k. To implement the leximin rule, it then suffices to solve a sequence of P_{lex} problems for $K \in \mathcal{A}$ to optimize the value of each component of the utility profile.

Example 3. For the example in Figure 2, the leximin-optimal allocation is $\pi_{\mathsf{lex}} = \{a \mapsto \{s_a, a_1, a_4, t_a\}, b \mapsto \{s_b, b_2, b_3, t_b\}\}$, with utility vector $(u(\pi_{\mathsf{lex}}(a)), u(\pi_{\mathsf{lex}}(b))) = (0.62, 0.70)$ and global utility $u(\pi_{\mathsf{lex}}) = 0.62 + 0.70 = 1.32$.

Algorithm 1: Leximin algorithm

Data: A PADAG problem $\langle \mathcal{A},\mathcal{G},\mu,\mathcal{C}\rangle$
Result: A leximin-optimal path allocation π
1 **for** $K=1$ *to* $|\mathcal{A}|$ **do**
2 $\quad(\lambda^*,sol)\leftarrow$ solve $P_{\mathsf{lex}}(\langle \mathcal{A},\mathcal{G},\mu,\mathcal{C}\rangle,K,[\Lambda_1,...,\Lambda_{K-1}])$
3 $\quad\Lambda_K\leftarrow\lambda^*$
4 **for** $g\in\mathcal{G}$ **do** $\pi(g)\leftarrow\{v\in V_g \mid sol(\beta_v)=1\}$
5 **return** π

Algorithm 2: Approximated leximin algorithm

Data: A PADAG problem $\langle \mathcal{A},\mathcal{G},\mu,\mathcal{C}\rangle$
Result: An iterated maximin-optimal allocation π
1 $\Delta\leftarrow[-1,...,-1]$
2 **for** $K=1$ *to* $|\mathcal{A}|$ **do**
3 $\quad(\delta^*,sol)\leftarrow$ solve $P_{\mathsf{a\text{-}lex}}(\langle \mathcal{A},\mathcal{G},\mu,\mathcal{C}\rangle,\Delta)$
4 $\quad S\leftarrow \underset{a\in\mathcal{A} \mid \Delta_a=-1}{\mathbf{argmin}} \sum_{g\in\mathcal{G}_a}\sum_{e\in E_g} u_g(e)sol(x_e)$
5 $\quad\hat{a}\leftarrow$ choose an agent a in S
6 $\quad\Delta_{\hat{a}}\leftarrow\delta^*$
7 **for** $g\in\mathcal{G}$ **do** $\pi(g)\leftarrow\{v\in V_g \mid sol(\beta_v)=1\}$
8 **return** π

3.3 Approximated Leximin Allocation (a-lex)

This previous model implements an exact leximin rule, and thus enforces fairness in the resulting allocation, but may not scale well when increasing the number of agents and edges. This is why we provide an approximate version of the computation of the leximin, based on an iterated maximin scheme. Basically, this approach considers at each step a minimum utility $\Delta_a \geq 0$ for some agents and maximizes the worst utility among the remaining agents, for which we arbitrarily assume $\Delta_a = -1$. The problem to solve, referred to as $P_{\mathsf{a\text{-}lex}}(\langle \mathcal{A},\mathcal{G},\mu,\mathcal{C}\rangle,\Delta)$, is the following one:

$$\textbf{max} \quad \delta \tag{15}$$

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

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

$$\sum_{g\in\mathcal{G}_a}\sum_{e\in E_g} u_g(e)x_e \geq \Delta_a, \quad \forall a\in\mathcal{A} \mid \Delta_a\neq-1 \tag{17}$$

The solution method then consists in optimizing in an iterated manner, as for leximin. As sketched in Algorithm 2, at each iteration (one per agent), $P_{\mathsf{a\text{-}lex}}$ is solved, one worst agent $\hat{a}$ is determined, and its minimum utility $\Delta_{\hat{a}}$ is fixed.

The main difference with P_{lex} is that at each iteration, in $P_{\mathsf{a\text{-}lex}}$, the position of an agent in the order is implicitly determined once for all, while in P_{lex} the order can be revised at each iteration. Moreover, if any equality occurs at line 5 to determine the worst agent (case

$|S|>1$), one may rely on some heuristic or arbitrary choice. Thus, $P_{\text{a-lex}}$ is an approximation of P_{lex} that contains fewer variables and constraints.

Example 4. The approximated leximin allocation for the example in Figure 2 is $\pi_{\text{a-lex}} = \{a \mapsto \{s_a, a_1, a_4, t_a\}, b \mapsto \{s_b, b_2, b_3, t_b\}\}$, with utility vector $(u(\pi_{\text{a-lex}}(a)), u(\pi_{\text{a-lex}}(b))) = (0.62, 0.70)$ and global utility $u(\pi_{\text{a-lex}}) = 0.62 + 0.70 = 1.32$. This is the same as π_{lex} since there are only two agents and no equality between the worst utilities.

3.4 Greedy Allocation (greedy)

For very fast decisions, iterated maximin might still be too slow. In such cases, a greedy approach can provide valid allocations quickly. The main idea of greedy path allocation is to iterate over the set of graphs. At each step, one graph g^* that has the best utility path is selected, this path is chosen as $\pi(g^*)$, and all nodes in the other graphs that are in conflict with nodes in $\pi(g^*)$ are deactivated. Graph g^* is then removed, and the process continues until there is no more graph to consider. This process ensures that constraints (1), (2), (3), (4), (5), (6) are met.

Determining the best path in a DAG g is linear time $\mathcal{O}(|E_g| + |V_g|)$ [3]. Obviously, greedy is equivalent to utilitarian when there is no conflict between graphs. Indeed, greedy will return the best path for each graph, which is the best utilitarian solution in such settings. Moreover, this greedy approach results in a Nash equilibrium where no agent might be able to improve its utility without a negative impact on other agents. This is equivalent to the *Nashify* procedure from [14] in the context of congestion games, with only one turn. We will see in the experiments that this equilibrium is far from being fair.

Example 5. The greedy allocation for the example in Figure 2 is $\pi_{\text{greedy}} = \{a \mapsto \{s_a, a_1, a_3, t_a\}, b \mapsto \{s_b, b_2, b_4, t_b\}\}$, with global utility $u(\pi_{\text{greedy}}) = u(\pi_{\text{greedy}}(a)) + u(\pi_{\text{greedy}}(b)) = 1.0 + 0.3 = 1.3$ and utility vector $(1.0, 0.3)$.

3.5 Round-Robin Allocations (p-rr and n-rr)

One fast approach to fair allocation of indivisible goods is *round-robin*. It consists in making each agent choose in turn (in a predefined fixed order) one item (depending on her preferences) until there is no more item to allocate. As greedy, it is polynomial in the number of agents and items. In our case, one may consider two kinds of items to allocate: paths (noted p-rr) or nodes (noted n-rr). In the case of paths, at her turn, each agent selects her best feasible path, given the already allocated nodes (to prevent conflicts). This process operates similarly to greedy, but alternates between users to balance the utilities. In the case of nodes, each agent incrementally builds the path associated with each of her graphs, by choosing in turn her next best feasible node until either it reaches the sink or there is no more feasible node to choose (a dead-end path). In the latter case, the agent is allocated the 0-utility source-to-sink path and looses the previously chosen nodes. In both approaches, constraints (1), (2), (3), (4), (5), (6) are met since considered paths are all feasible. Let's note that p-rr results in a Nash equilibrium where each agent has been allocated the best path given the other allocations. This is not the case for n-rr, since some nodes left by some agent falling in a dead-end may have prevented some other agents to find a better solution.

Example 6. The path-round-robin allocation $\pi_{\text{p-rr}}$ for the example in Figure 2 is equivalent to π_{greedy}, since a chooses $\{s_a,a_1,a_3,t_a\}$ and then b chooses $\{s_b,b_2,b_4,t_b\}$. The node-round-robin allocation $\pi_{\text{n-rr}}$ is also equivalent to π_{greedy} because a first chooses a_1, then b chooses b_2 (only feasible option), then a chooses a_3 (best option), and finally b chooses b_4 (only feasible option).

4 Experimental Evaluation

In this section, we evaluate the different allocation methods proposed when applied to an orbit slot allocation problem[1] encoded as PADAGs. We expound the experimental setup and analyze some results obtained on synthetic instances.

Experimental Setup. We consider a Low-Earth Orbit constellation (500km altitude) composed of n_p regularly-spaced orbital planes having a 60-degree inclination, with $n_p \in \{2,4,8,16\}$ and 2 regularly spaced satellites over each orbital plane. To generate PADAG instances, we randomly generate requests for 4 agents wishing to obtain orbit slot ownerships to implement some repetitive ground acquisitions of Points Of Interest (POIs) belonging to the same area. POIs are randomly selected within a extracted subset from [13]. All the agents have the same template for a request r: getting an observation every day at 8:00 + δ_r, 12:00 + δ_r, and 16:00 + δ_r, with a tolerance of 1 hour around each time plot, and uniform random time shift $\delta_r \in [-2,2]$ for all the time plots of the same request. For each POI and each time plot, the orbit slots over which orbit ownership can be claimed for performing observations are determined thanks to a space mechanics toolbox, based on the assumption that a satellite is relevant for a POI as soon as its elevation above the horizon is greater than 15 degrees. Incompatible time slots are those which overlaps while belonging to the same satellite.

We then encode these requests and orbit slots into PADAGs. Each request is mapped to a graph, in which nodes (except source and sink) are orbit slots for capturing a POI at some time plot, and edges links two such consecutive orbit slots to answer the request. For instance, Figure 2 represents two requests from two users (a and b), with two time plots, each having two possible orbit slots per time plot per user (e.g. for user a, a_1 and a_2 for the first time plot, a_3 and a_4 for the second one). For simplicity, we only consider utilities attached to the slots, and not to the transitions between slots. We study a *linear* utility function, which is linear in the distance between the middle τ of the allocated slot and the requested time plot (utility linearly decreasing from 1 when τ is exactly on the time plot to 0 when τ reaches the bounds of the tolerance window). We normalize each utility wrt the maximum utility that can be achieved for each graph individually. We consider 2 requests per agent, and a horizon of 365 days resulting in DAGs having 1095 layers (1095 time plots). This setting results in DAGs having the following properties on average:

n_p	2	4	8	16
DAG width	3.08	5.41	10.05	19.38
conflicts	26798.80	45636.06	82971.20	158180.20
slot duration (s)	603.28	600.10	599.87	598.75

Solvers are coded in Java 1.8 and executed on 20-core Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz, 62GB RAM, Ubuntu 18.04.5 LTS. Utilitarian, leximin and approximated

[1] which is a novel problem, not to be confused with orbit scheduling problems [9].

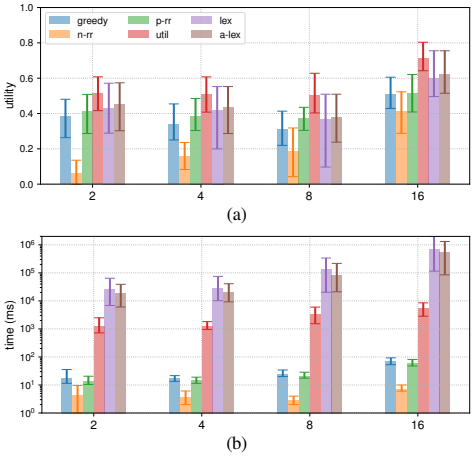

Fig. 3: Average overall utility (left) and computation time (right) obtained by each algorithm for each constellation size.

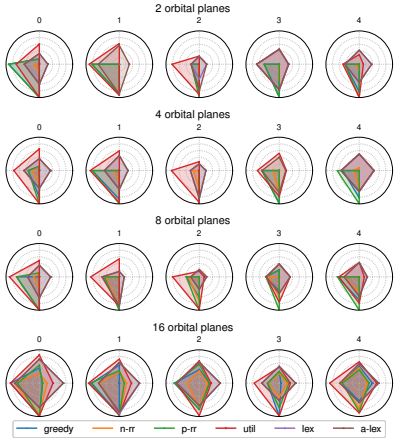

Fig. 4: Utility profiles (in leximin order) for the first 5 instances (over 30) for each constellation size and each algorithm (south: best utility over all agents; west: second best utility; north: third best utility; east: worst utility).

leximin make use of the Java API of IBM CPLEX 20.1 (with 10min timeout). We ran 30 instances of randomly generated PADAGs and plot the average with $[0.05,0.95]$ confidence.

Utility. Figure 3a shows the average normalized global utility for each algorithm and constellation size (expressed in number of orbital planes). Normalized global utility is the mean graph utility, thus between 0 and 1. Obviously util provides the optimal utilitarian allocation. In second position, a-lex provides good allocations at almost 85% of the optimal value on average. Interestingly, lex performs equivalently to a-lex (gap less than 5% on average). Indeed, a-lex provides slightly better allocations from the utilitarian point of view, at the expense of fairness approximation. Round-robin approaches really differ in utility. While p-rr provides allocations at almost 71% of the optimal, n-rr results in low-utility allocations, at almost 10% of the optimal on smaller constellations. In fact, in such settings, there are few feasible paths for each request. Thus, for most of the requests, the myopic incremental building of paths results in dead-ends, and thus on 0-utility allocations; while, by considering another agent order, the allocations could have been better. Finally, greedy behaves slightly worse than p-rr, at almost 68% of the optimal value on average. On larger instances, where many paths exist to answer each request, greedy performs even better. More generally, larger constellation problems are easier to solve from the utilitarian point-of-view for non-optimal algorithms, since there are more options to avoid conflicts, even if the number of conflicts is higher.

Fairness. To analyze the fairness of the resulting allocations, Figure 4 provides the utility profiles obtained for the first 5 instances for each algorithm. greedy, by principle, seeks to first allocate the highest utility paths, resulting in unfair allocations where only the first 2 or 3 users are served, while most of the time the fourth user has no request fulfilled. Round-robin approaches are fairer than greedy and often fulfill requests for more users. util results in

profiles with the larger surface, but most of the time the fourth user is neglected. Finally, lex and a-lex behave almost identically (their profiles are superimposed), showing that the a-lex approximation is sufficient to output fair allocations. Let's note that with growing constellations, since there is more and more options to serve users, all the algorithms tend to result in fairer allocations. Still, lex and a-lex are the best choices here, and the round-robin competitors output unsatisfactory allocations.

Computation Time. Figure 3b shows the average computation time in milliseconds for each algorithm and each constellation size. As expected (by design), greedy, p-rr and n-rr are the fastest. n-rr which does not even perform shortest path operations is the fastest by far, but results in very bad and not-so-fair allocations. greedy and p-rr are very fast, but are based on multiple max path search in DAGs. p-rr still quickly provides quite good utilitarian and fair allocations. util, based on a single ILP solving, is 100 times slower than the fastest algorithms. Then, lex and a-lex are up to two orders of magnitude slower than util on the largest constellations. This is due to the multiple calls ($|\mathcal{A}|$) to the MILP solver on large problems. a-lex is 2 to 3 times faster than lex since it solves smaller MILPs, while resulting in allocations that are as fair as lex's ones.

5 Conclusion

In this paper, we proposed the PADAG model, a novel allocation problem where agents express their preferences over bundles of items as edge-weighted DAGs. We introduced and analyzed several solution methods (utilitarian, leximin, approximate leximin, greedy) against the round-robin allocations, from the utilitarianism and fairness perspectives. We evaluated these methods on large randomly generated instances of orbit slot allocation problems, with more than 1000 layers. On larger constellations, we observe that approximate leximin constitutes a good trade-off between utilitarian optimality, leximin optimality and computation time, compared to all other solution methods. It is even equivalent to exact leximin on most of the instances, while dividing the computation times by a factor 2 to 3.

We identify several tracks for future investigations. First, as approximate leximin is a trade-off between fairness and computation times, we would like to investigate other solution methods, even more reactive, as to solve larger PADAGs and specific topologies. Indeed, as for path division and congestion games, dedicated techniques could be devised for given topologies (stars, chains, etc.). Second, since PADAGs are strongly constrained by conflicts, we aim to explore min-conflict heuristics to improve our algorithms. Finally, we believe PADAGs have great potential to be used in a variety of domains, and we thus aim to evaluate the proposed techniques on problems coming from other application fields, like the NFV domain where function chains are modeled as graphs, and incompatibilities control the access to nodes or multi-agent path finding domain (path preferences are modeled as graphs, and incompatibilities model the constraints to avoid two agents to occupy the same position at the same time).

References

1. Bouveret, S., Cechlárová, K., Elkind, E., Igarashi, A., Peters, D.: Fair division of a graph. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 135–141. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/20

2. Caragiannis, I., Kurokawa, D., Moulin, H., Procaccia, A.D., Shah, N., Wang, J.: The unreasonable fairness of maximum nash welfare. In: Proceedings of the 2016 ACM Conference on Economics and Computation. p. 305–322. EC '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2940716.2940726, https://doi.org/10.1145/2940716.2940726
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, 2nd edn. (2001)
4. Du, Y., Sami, R., Shi, Y.: Path auctions with multiple edge ownership. Theor. Comput. Sci. **411**(1), 293–300 (2010). https://doi.org/10.1016/j.tcs.2009.09.032
5. Hughes, M.S., Lunday, B.J., Weir, J.D., Hopkinson, K.M.: The multiple shortest path problem with path deconfliction. Eur. J. Oper. Res. **292**(3), 818–829 (2021). https://doi.org/10.1016/j.ejor.2020.11.033
6. Igarashi, A., Peters, D.: Pareto-optimal allocation of indivisible goods with connectivity constraints. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019. pp. 2045–2052. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33012045
7. Immorlica, N., Karger, D.R., Nikolova, E., Sami, R.: First-price path auctions. In: Riedl, J., Kearns, M.J., Reiter, M.K. (eds.) Proceedings 6th ACM Conference on Electronic Commerce (EC-2005), Vancouver, BC, Canada, June 5-8, 2005. pp. 203–212. ACM (2005). https://doi.org/10.1145/1064009.1064031
8. Kurokawa, D., Procaccia, A.D., Shah, N.: Leximin allocations in the real world. ACM Transactions on Economics and Computation **6**(3–4) (2018). https://doi.org/10.1145/3274641, https://doi.org/10.1145/3274641
9. Lemaître, M., Verfaillie, G., Fargier, H., Lang, J., Bataille, N., Lachiver, J.M.: Equitable allocation of earth observing satellites resources. In: 5th ONERA-DLR Aerospace Symposium (ODAS'03) (2003)
10. Megiddo, N.: Optimal flows in networks with multiple sources and sinks. Math. Program. **7**(1), 97–107 (1974). https://doi.org/10.1007/BF01585506
11. Moulin, H.: Fair division and collective welfare. MIT Press (2003)
12. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, USA (2007)
13. OpenStreetMap: Openstreetmap points of interest (on french territory). https://www.data.gouv.fr/fr/datasets/points-dinterets-openstreetmap/ (2021), accessed: 30-08-2021
14. Panagopoulou, P.N., Spirakis, P.G.: Algorithms for pure nash equilibria in weighted congestion games. ACM J. Exp. Algorithmics **11**, 2.7–es (feb 2007). https://doi.org/10.1145/1187436.1216584, https://doi.org/10.1145/1187436.1216584
15. Picard, G.: Auction-based and distributed optimization approaches for scheduling observations in satellite constellations with exclusive orbit portions. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS-22). IFAAMAS (2022)
16. Riera, J.F., Escalona, E., Batallé, J., Grasa, E., García-Espín, J.A.: Virtual network function scheduling: Concept and challenges. In: 2014 International Conference on Smart Communications in Network Technologies (SaCoNeT). pp. 1–5 (2014). https://doi.org/10.1109/SaCoNeT.2014.6867768
17. Yang, S., Li, F., Trajanovski, S., Yahyapour, R., Fu, X.: Recent advances of resource allocation in network function virtualization. IEEE Transactions on Parallel and Distributed Systems **32**(2), 295–314 (2021). https://doi.org/10.1109/TPDS.2020.3017001
18. Zhang, L., Chen, H., Wu, J., Wang, C., Xie, J.: False-name-proof mechanisms for path auctions in social networks. In: ECAI 2016. Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1485–1492. IOS Press (2016). https://doi.org/10.3233/978-1-61499-672-9-1485