Path Allocation Techniques with Conflicting Preferences Represented as Weighted Directed Acyclic Graphs

Application to Orbit Slot Ownership in Earth Observation Satellite Constellations

Sara Maqrot (sara.maqrot@onera.fr)

27 january 2021

Introduction

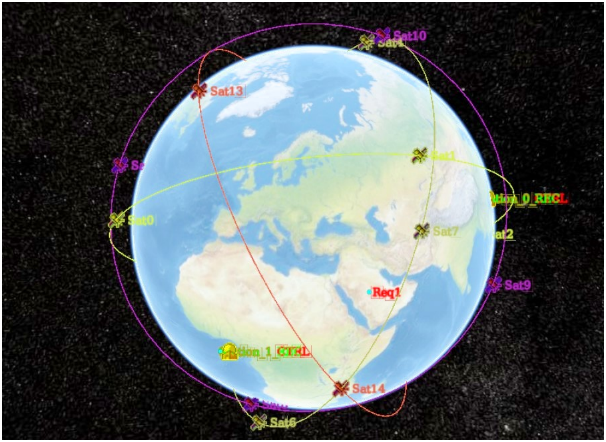


FIGURE – Earth Observation Satellite Constellation

- **Problem :** multi-user exploitation of Earth Observation Constellation' resources

Offline booking		Online booking
Systematic Exclusive orbit slot	—— exclusivity periods ——→	Image Acquisition

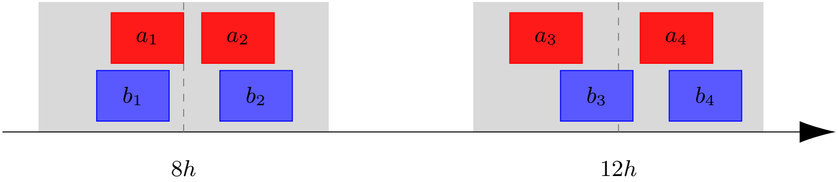
- **Usual concept to sell exclusivity over orbit slots :** first come, first served
- **Objective**

Users'preferences (POI, temporal request tolerance, ...) → **Candidate Orbit Slots** (objects to share) → **Optimal-fair and utilitarian multi-user resource sharing**

Orbit slot allocation problem

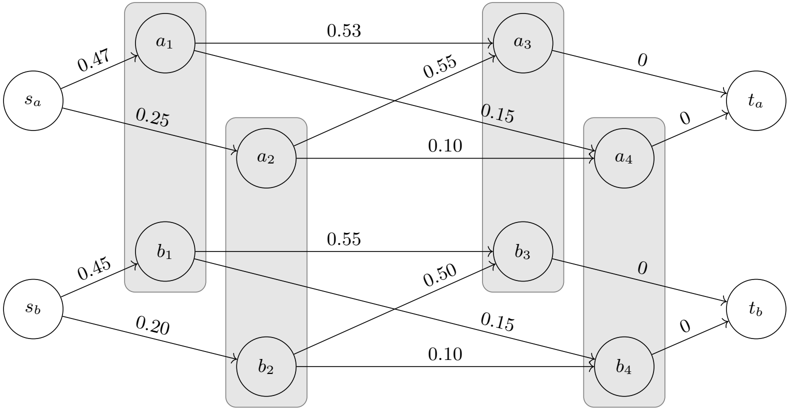
Example :

- 2 agents (a in red, b in blue) requesting :
  - Position : POIs belonging to the same area
  - Temporal : slots around 2 time plots ($8h$ and $12h$) every day, with tolerance windows around each plot (in gray)
- One satellite allowing 2 opportunities of candidate orbit slots for each plot ($a_1, \ldots, a_4, b_1, \ldots, b_4$)



Problem Model

Example :



Problem Model : Definitions and notations

A problem of PADAG

(*Path Allocation in multiple conflicting edge-weighted Directed Acyclic Graphs*) is a tuple $\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle$, where

- $\mathcal{A}$: a set of *agents*
- $\mathcal{G}$: a set of edge-weighted DAGs, each $g \in \mathcal{G}$ is a triple $\langle V_g, E_g, u_g \rangle$
- μ : maps each graph $g \in \mathcal{G}$ to its owner $a \in \mathcal{A}$
- $\mathcal{C}$: a set of conflicts between pairs of nodes from two distinct graphs from two distinct agents

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

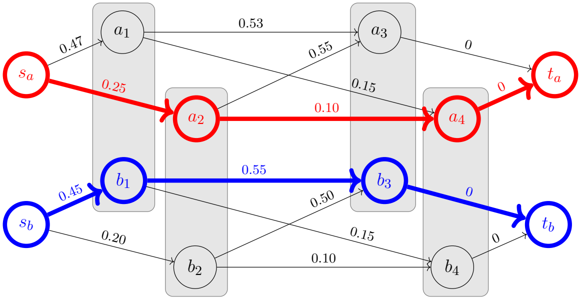
Path allocation schemes

1. **Greedy** (faster utilitarian)
2. **Classical utilitarian** (optimal utilitarian)
3. **Optimal leximin** (optimal fair utilitarian)
4. **Approximated leximin** (approx. fair utilitarian + faster than optimal leximin)

Algorithm 1 : Greedy algorithm

Data : A PADAG problem $\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle$
Result : A greedy path allocation π

```
while 𝒢 ≠ ∅ do
    get g* the graph with the max utility path p;
    π(g*) ← p;
    for g ∈ 𝒢 do
        V_g ← {v ∈ V_g | ∀w ∈ π(g*), {v, w} ∉ 𝒞};
        E_g ← {(v, w) ∈ E_g | v ∈ V_g, w ∈ V_g}
    𝒢 ← 𝒢 \ {g*};
return π
```



$$u(\pi_{\text{greedy}}) = u(a \mapsto \{s_a, a_3, a_4, t_a\}) + u(b \mapsto \{s_b, b_1, b_3, t_b\}) = 0.35 + 1.0 = 1.35$$

Path allocation schemes in MILP-based approaches

- **Binary variables :** for any DAG $g = \langle V_g, E_g, u_g \rangle$, we define :
  - x_e, stating whether edge $e \in E_g$ is selected in solution path $\pi(g)$
  - β_v, stating whether node $v \in V_g$ is selected in solution path $\pi(g)$

- **Constraints :**

to define all the possible paths

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

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

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

to account for item selection conflicts

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

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

to ensure that sources and sinks are selected

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

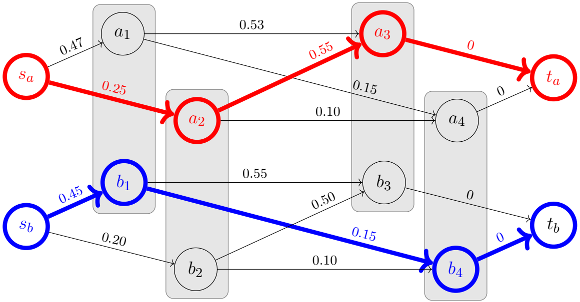
2. Utilitarian Allocation

$$P_{\text{util}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle) \quad \text{maximize} \quad \sum_{a \in \mathcal{A}} \sum_{g \in \mathcal{G}_a} \sum_{e \in E_g} u_g(e) \cdot x_e \tag{7}$$

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

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

Example :



$$u(\pi_{\text{util}}) = u(\{a \mapsto \{s_a, a_2, a_3, t_a\}) + u(b \mapsto \{s_b, b_1, b_4, t_b\}) = 0.80 + 0.60 = 1.40$$

3. Optimal Leximin Allocation

- **Step 1 :** maximize the worst utility

$$\Lambda_1 = 0.62$$

- **Step 2 :** maximize the second worst utility knowing that the worst utility is 0.62

$$\Lambda_2 = 0.70$$

- **Step 3 :** maximize the worst utility knowing that the worst utilities are 0.62 and 0.70

3. Optimal Leximin Allocation

Formally, let $\Lambda = [\Lambda_1, \dots, \Lambda_n]$ denote the vector of utilities sorted in non-descending order. The leximin mechanism returns the allocation that maximizes this vector in the lexicographic order.

Algorithm 2 : Leximin algorithm

Data : A PADAG problem $\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle$
Result : A leximin-optimal path allocation π

for $K = 1$ *to* $|\mathcal{A}|$ **do**
  $(\lambda^*, sol) \leftarrow$ solve $P_{\text{lexi}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle, K, [\Lambda_1, \dots, \Lambda_{K-1}])$;
  $\Lambda_K \leftarrow \lambda^*$
for $g \in \mathcal{G}$ **do**
  $\pi(g) \leftarrow \{v \in V_g \mid sol(\beta_v) = 1\}$;
return π

$P_{\text{lexi}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle, K, [\Lambda_1, \dots, \Lambda_{K-1}])$:

$$\text{maximize} \quad \lambda \tag{9}$$

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

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

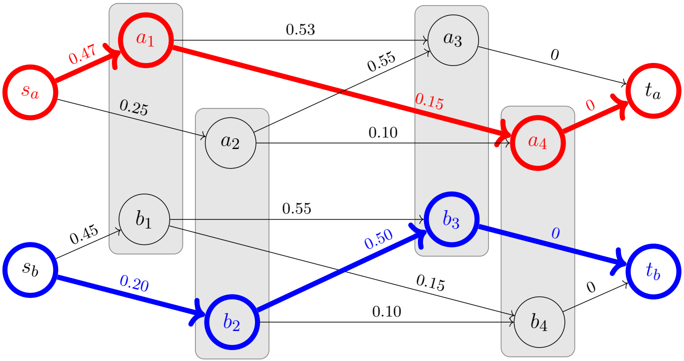
$$\sum_{a \in \mathcal{A}} y_{ak} = 1, \quad \forall k \in [1..K-1] \tag{11}$$

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

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

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

Example



$$u(\pi_{\text{lexi}}) = u(\{a \mapsto \{s_a, a_1, a_4, t_a\}) + u(b \mapsto \{s_b, b_2, b_3, t_b\}) = 0.62 + 0.70 = 1.32.$$

- **Step 1 :** maximize the worst utility and choose the associated agent

$$\sout{A_1 = 0.62} \qquad u_a = 0.62$$

- **Step 2 :** maximize the second worst utility knowing that agent 'a' has utility equal to 0.62

$$\sout{A_2 = 0.70} \qquad u_b = 0.70$$

- **Step 3 :** maximize the worst utility knowing that agent 'a' has a utility equal to 0.62 and agent 'b' has a utility equal to 0.70

4. Approximated Leximin Allocation

Algorithm 3 : Approximated leximin algorithm

Data : A PADAG problem $\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle$
Result : An iterated maximin-optimal allocation π

$\Delta \leftarrow [-1, \ldots, -1]$;
for $K = 1$ *to* $|\mathcal{A}|$ **do**
  $(\delta^*, sol) \leftarrow$ solve $P_{\text{approx}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle, \Delta)$;
  $S \leftarrow \underset{a \in \mathcal{A} \mid \Delta_a = -1}{\operatorname{argmin}} \sum_{g \in \mathcal{G}_a} \sum_{e \in E_g} u_g(e) sol(x_e)$;
  $\hat{a} \leftarrow$ choose an agent a in S ;
  $\Delta_{\hat{a}} \leftarrow \delta^*$;
for $g \in \mathcal{G}$ **do** $\pi(g) \leftarrow \{v \in V_g \mid sol(\beta_v) = 1\}$;
return π

$P_{\text{approx}}(\langle \mathcal{A}, \mathcal{G}, \mu, \mathcal{C} \rangle, \Delta)$

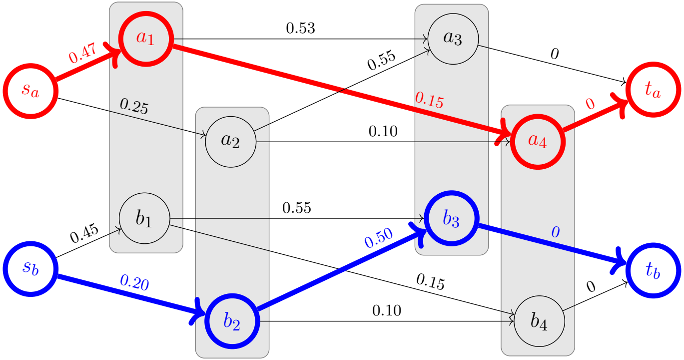
$$maximize \quad \delta \tag{15}$$

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

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

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

Example



$$u(\pi_{\text{approx}}) = u(a \mapsto \{s_a, a_1, a_4, t_a\}) + u(b \mapsto \{s_b, b_2, b_3\}) = 0.62 + 0.70 = 1.32.$$

Experimental Setup

Constellation

- Low-Earth Orbit constellation (500km altitude)
- 8 orbital planes (60-degree inclination)
- $n_s \in \{2, 4, 8, 16\}$ regularly-spaced satellites over each orbital plane.

Constellation'users

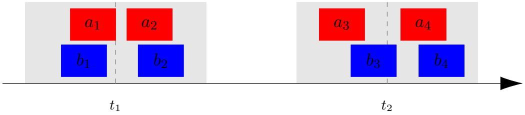
- 4 agents having the same request template to make the problems very conflicting :
  - position : POIs belonging to the same area (source : OpenStreetMap 2021),
  - repetitive ground acquisitions, every day at $8:00$, $12:00$, and $16:00$,
  - with a tolerance of 1 hour around each time plot.
- 2 requests per agents.
- an horizon of 7 days resulting in DAGs having 21 layers (21 time plots)

Software used

- Solvers are coded in Java 1.8
- Utilitarian, leximin and approx. leximin make use of the Java API of IBM CPLEX 20.1

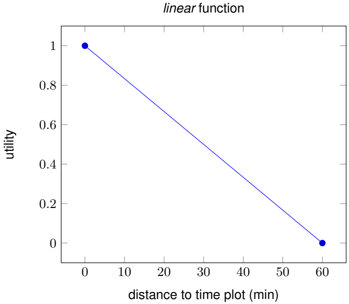
Utilities :

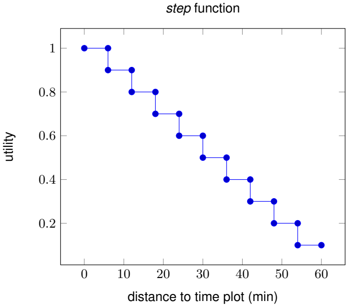
We consider utilities **attached to the slots** (and not to the transitions between slots).



We consider two functions (the same for all users) :

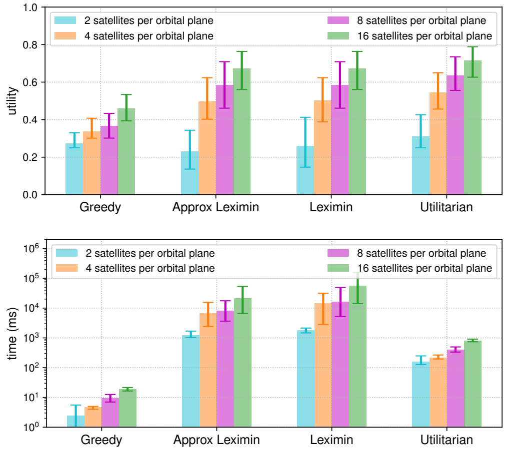
- *linear* on the distance between the middle of the slot and the requested time plot (utility 1 if exactly on the time plot, 0 when outside of the tolerance window),
- *step* function from 0.1 to 1.0. It degrades of 0.1 every 6 min, until a full hour is reached.



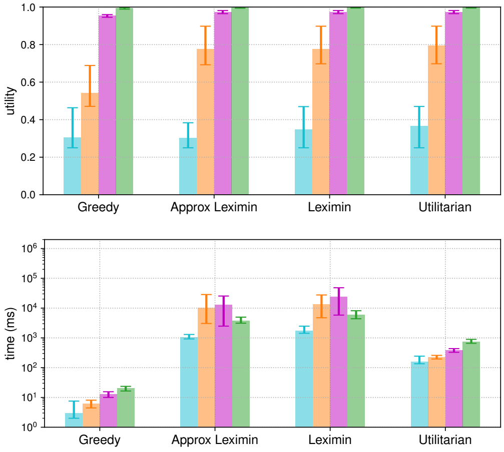


Results : Overall Utility / Computation Time

240 instances = 30 (POIs randomly generated) $\times$ 4 (config. of constellation) $\times$ 2 (utility functions)



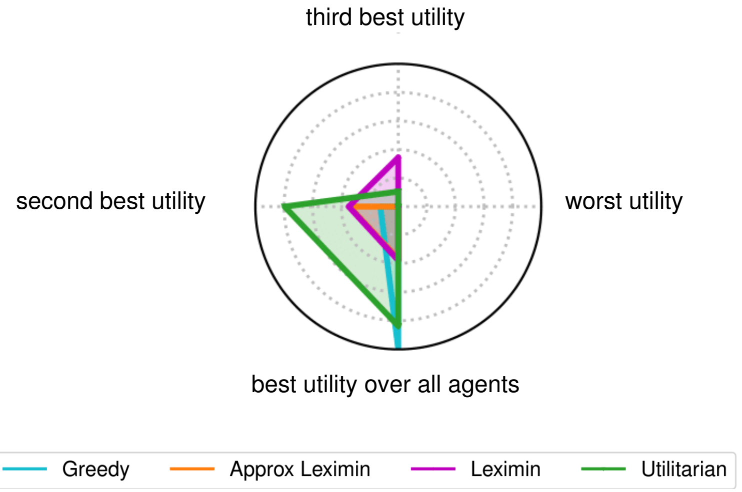
linear utility function



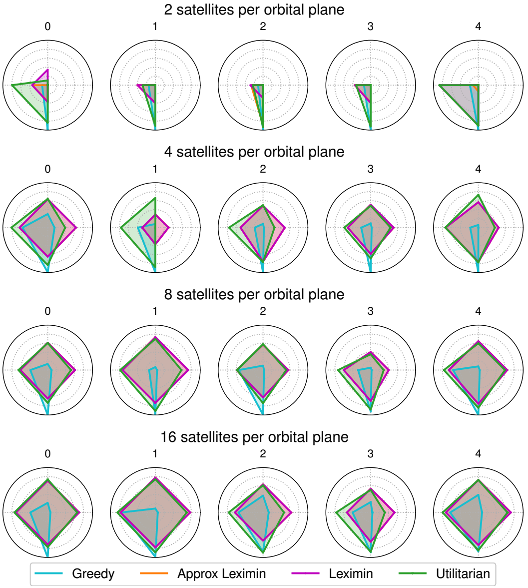
step utility function

Results : Fairness

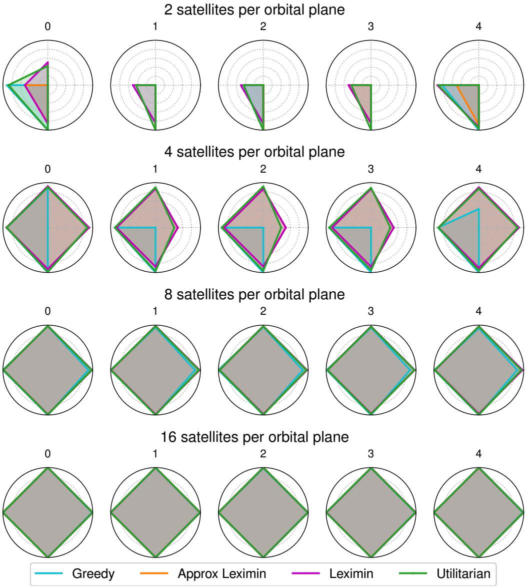
Utility profiles (in leximin order)



Utility profiles for the first 5 instances (over 30) for each constellation size and each algorithm :



Linear utility function



Step utility function

Conclusions

- We proposed a first approach of orbit slot allocation problem using Path Allocation in multiple conflicting edge-weighted Directed Acyclic Graphs, towards optimal-fair and utilitarian allocations.
- We presented several allocation strategies :
  - Greedy
  - Utilitarian
  - Optimal leximin
  - Approximated leximin
- Experiments show that the interesting strategies (trade-off between utilitarianism, fairness and computation time) for larger constellations :
  - Approximate leximin for *linear* utility function
  - Greedy algorithm for *step* utility function
- IJCAI-22 and ROADEF-22 submission

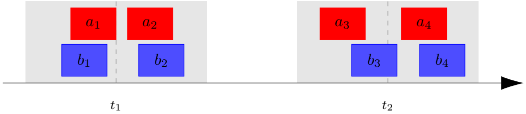
Perspectives

Work-in-progress

- solve large scale instances of orbit slot allocation problem using local search techniques, which have been successfully applied shortest paths, routing and flow problems, sharing some similarities with PADAG
- consider heterogeneous requests (systematic, periodic, punctual)

Future work : improve the formulation of the problem to be more realistic

- Consider dividing orbit slots when conflicts occur



- Consider larger areas (AOI instead of POI)
- In addition to user satisfaction, consider the long-term satellite operator satisfaction : ensure enough available orbit slots for the short-term planning

Long-term booking Systematic exclusive orbit slots —— exclusivity periods ——→ **Short-term booking** Image Acquisition

Thank you for your attention !