Allocation de chemins avec des préférences conflictuelles sous forme de graphes pour le partage d'orbites

Sara Maqrot, Gauthier Picard, Cédric Pralet, Stéphanie Roussel

ONERA/DTIS, Université de Toulouse, France

25/02/2022

Introduction

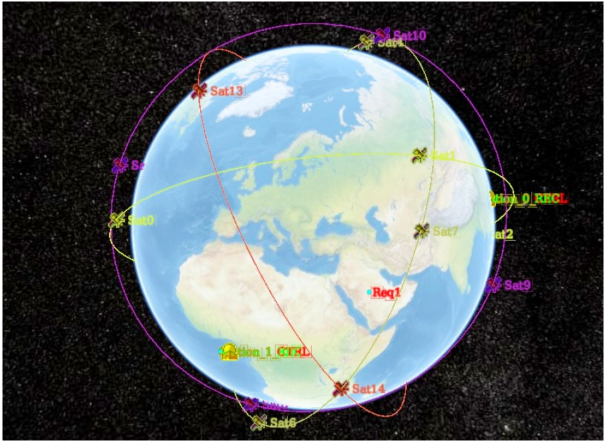


FIGURE – Earth Observation Satellite Constellation

Introduction

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

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

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

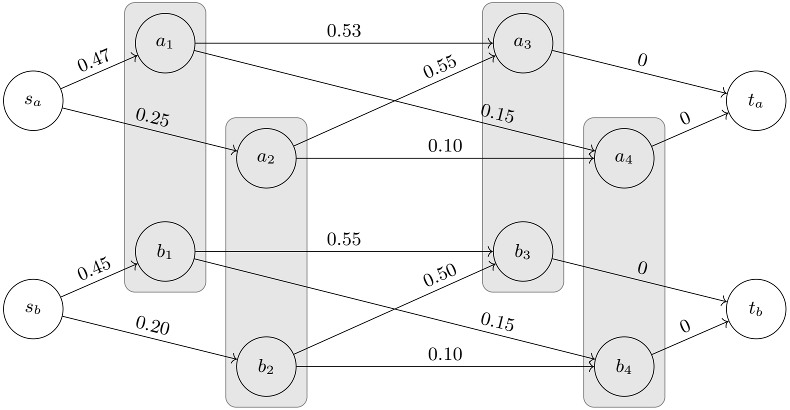
Users'preferences (POI, temporal request tolerance, ...) → **Candidate Orbit Slots** (objects to share) → **Optimal-fair and utilitarian multi-user resource sharing**

Example :

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

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

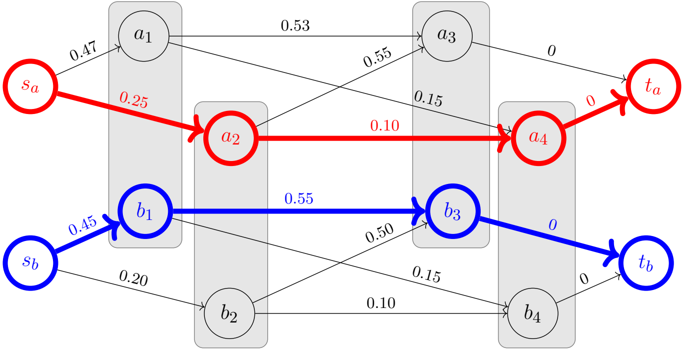
An allocation

- 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

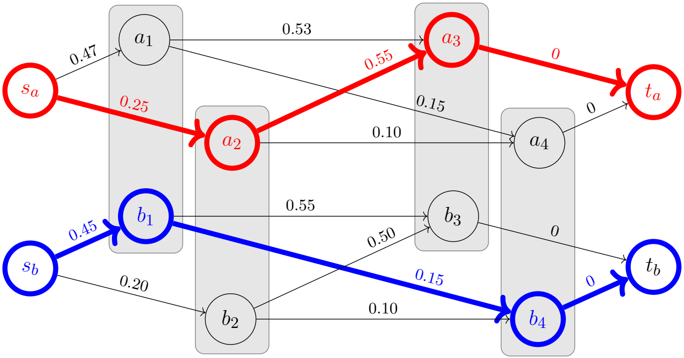
1. **Greedy** (faster utilitarian)
2. **Classical utilitarian** (optimal utilitarian)
3. **Optimal leximin** (optimal fair utilitarian)
4. **Approximated leximin** (approx. fair utilitarian + faster than optimal leximin)

1. Greedy Allocation (faster utilitarian)



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

2. Utilitarian Allocation (optimal utilitarian)



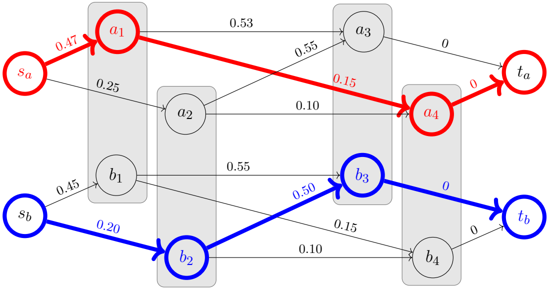
$$u(\pi_{\text{util}}) = u(\{a \mapsto \{s_a, a_2, a_3, t_a\}) + u(b \mapsto \{s_b, b_1, b_4, t_b\}) = 0.80 + 0.60 = 1.40$$

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 1 : 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

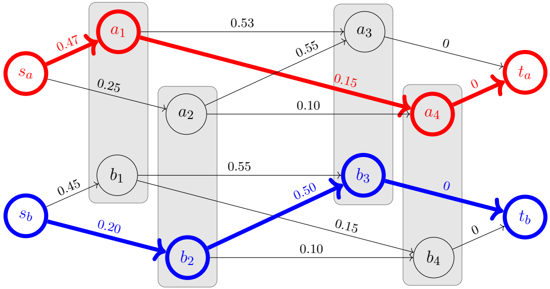


$u(\pi_{\text{lexi}}) = u(\{a \mapsto \{s_a, a_1, a_4, t_a\}) + u(b \mapsto \{s_b, b_2, b_3, t_b\}) = 0.62 + 0.70 = 1.32.$

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

$\Delta \leftarrow [-1, \ldots, -1]$;
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



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

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.

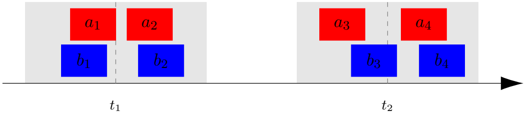
Constellation'users

- 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

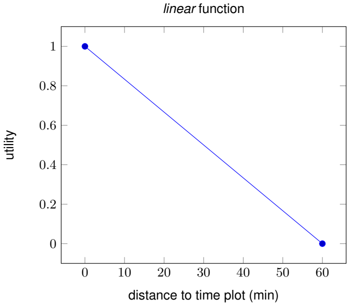
- Solvers are coded in Java 1.8
- Utilitarian, leximin and approx. leximin make use of the Java API of IBM CPLEX 20.1

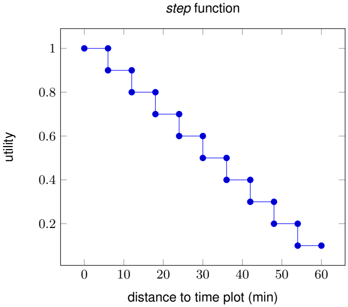
Utilities attached to the slots (and not to the transitions between slots)



Two functions (the same for all users) :

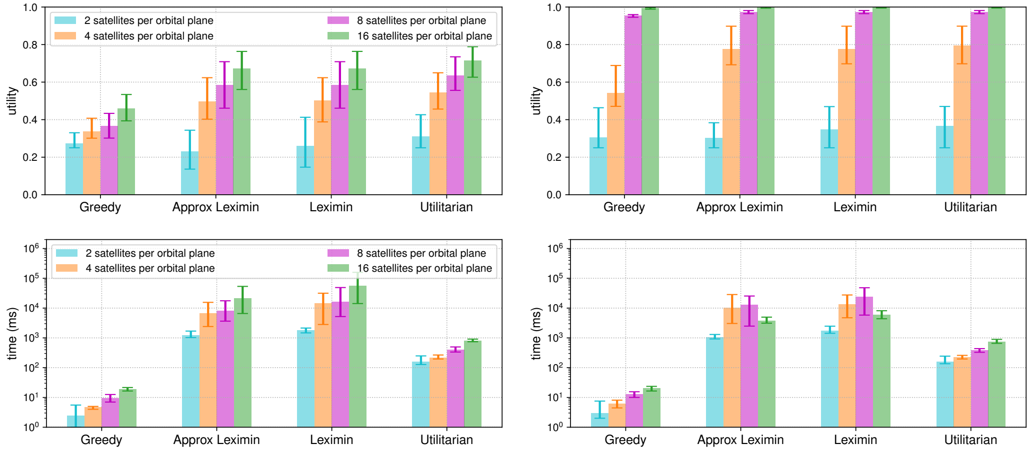
- *linear* on the distance between the middle of the slot and the requested time plot (utility 1 if exactly on the time plot, 0 when outside of the tolerance window),
- *step* function from 0.1 to 1.0. It degrades of 0.1 every 6 min, until a full hour is reached.





Results : Overall Utility / Computation Time

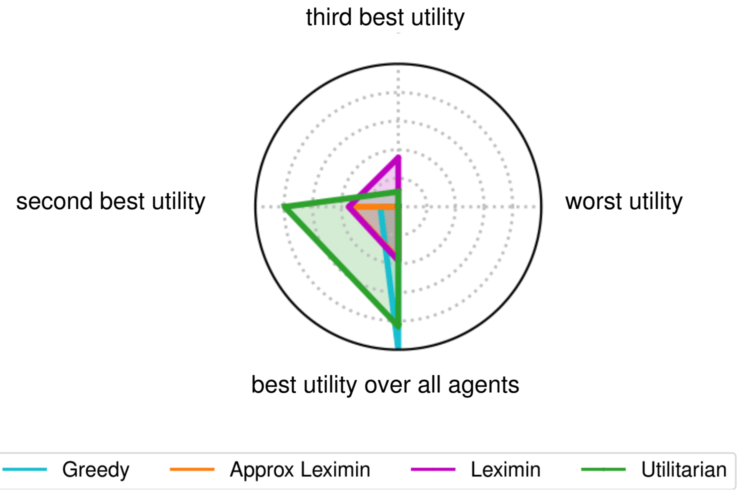
240 instances = 30 (POIs randomly generated) × 4 (config. of constellation) × 2 (utility functions)



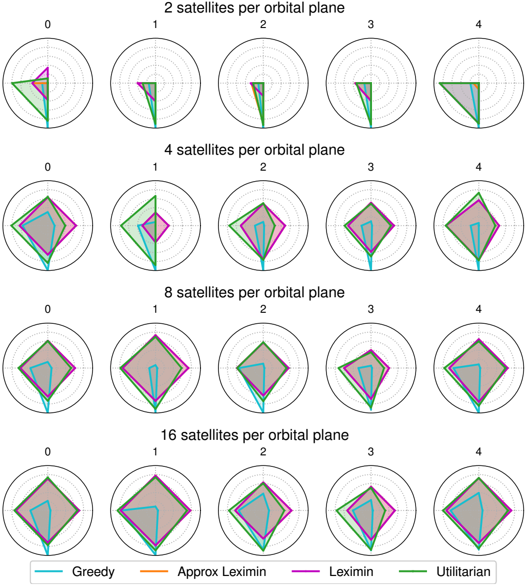
linear utility function

step utility function

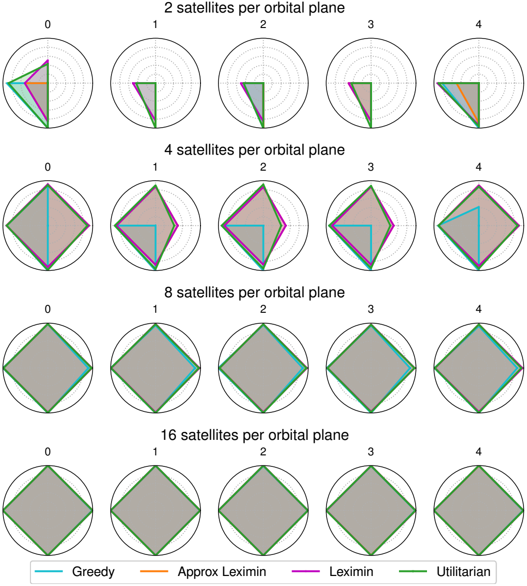
Utility profiles (in leximin order)



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



Linear utility function



Step utility function

Conclusions

- First approach of orbit slot allocation problem using *Path Allocation in multiple conflicting edge-weighted Directed Acyclic Graphs*
- Several allocation strategies :
  - Greedy
  - Utilitarian
  - Optimal leximin
  - Approximated leximin
- Best trade-off between utilitarianism, fairness and computation time :
  - Approximate leximin for *linear* utility function
  - Greedy algorithm for *step* utility function

Perspectives

Work-in-progress

- Solve large scale instances of orbit slot allocation problem using iterative conflict-repair method
- Consider heterogeneous requests (systematic, periodic, punctual)
- Consider dividing orbit slots when conflicts occur

Future work :

- 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