# Orbit slot allocation in earth observation constellations

Sara Maqrot, Stéphanie Roussel, Gauthier Picard, Cédric Pralet

*PAIS 2022*
*A. Passerini and T. Schiex (Eds.)*


*doi:10.3233/FAIA220061*

# Orbit Slot Allocation in Earth Observation Constellations

Sara MAQROT, Stéphanie ROUSSEL, Gauthier PICARD and Cédric PRALET
*ONERA/DTIS, Université de Toulouse*

**Abstract.** In the context of Earth observation constellations, we consider the problem of allocating orbit slots to clients requesting some ownership of orbit portions overflying desired regions on Earth. This problem arises prior to operational scheduling of observation tasks, in constellations where users can directly communicate with the satellites using their own ground stations. Observation scheduling in the exclusive slots is then delegated to the clients themselves. To perform the allocation of exclusive slots, we propose a two-level optimization approach, where the optimization process (led by either utilitarian or fair criterion) explores the solution space using a feasibility checker based on a constraint solver. We experimentally evaluate and analyze their performance on randomly generated order books and real constellation configurations.

**Keywords.** Constraint programming, Earth observation, orbit slot allocation, utilitarianism, leximin fairness

## 1. Introduction

Every day, Earth observation satellites perform a huge number of images of the Earth surface and deliver the associated image products to the end-users that posted observation requests for various purposes (observation of critical areas due to natural disasters or crisis situations, infrastructure observation, environment monitoring, etc.). In such systems, the users post requests to the main mission center, this mission center computes observation plans, the latter are sent to the satellite when it overflies a ground control stations, and then the satellite performs its images and communicates the collected data when it overflies a ground reception station. In this context, constellations of Earth observation satellites have also been developed, one ambition being to deliver images as soon as possible after imaging requests are formulated. Based on these new observation capabilities, another ambition is to allow the end-users to express more complex requests, such as *periodic requests* that consist in observing an area of interest every $H$ hours. However, despite the increase in the number of satellites, the system can still be over-constrained, meaning that it is not possible to complete all user requests over a short time period. As a result, the main mission center needs to select the subset of requests that will be performed for the next period, for instance the next day, based on some request prioritization depending on various features (nature of the observation goal, long-term relationship with some users, financial reward for the request, etc.).

With such a request selection process, users are never guaranteed to have their requests quickly fulfilled. Therefore, another paradigm is currently being developed in the

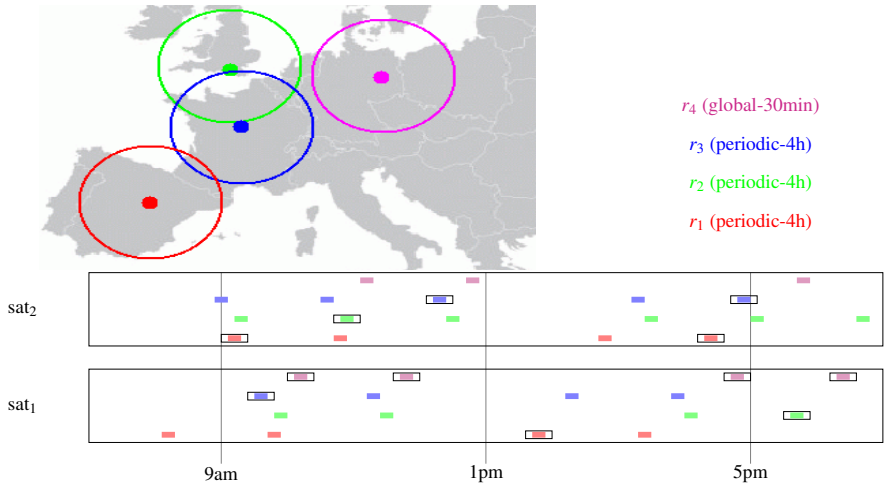



**Figure 1.** An orbit slot allocation example involving two satellites and four requests, with all slot opportunities for each request over each satellite, together with the slots that could be selected by the system.

space domain. In this new paradigm, the users can buy in advance so-called *exclusive time windows* over some satellites, and then exploit these reserved windows to communicate with the satellites, using their own ground stations, or to perform observations. For instance, a user having a ground station $X$ might be interested in first booking time window [10:32:05-10:41:23] over satellite $s_8$, when this satellite overflies station $X$ with a sufficient elevation angle, and then exploit this window at execution time to (1) send an activity plan to $s_8$ over window [10:32:05-10:32:40], (2) perform two observations of areas located around $X$ over time windows $w_1$=[10:35:09-10:35:52] and $w_2$=[10:37:21-10:37:40], (3) get the data back at station $X$ over time window [10:39:00-10:41:23]. In addition to strong guarantees over the access to satellites of the constellation, the exclusivity booking paradigm also allows the users to keep some privacy over the observations performed, since for instance in the previous example no third-party can inspect the precise images collected over time windows $w_1$ and $w_2$. Moreover, several end-users may formulate exclusivity requests and there is a need to allocate exclusivity windows, given that the set of exclusivity windows associated with a single satellite must be disjoint. Figure 1 provides an example involving four exclusivity requests and two satellites. It gives, for each satellite, the candidate orbit slots associated with each of the four exclusivity requests. Requests $r_1$, $r_2$, $r_3$ are periodic requests of period $4h$, where the goal for each request is to get one orbit slot around $9am$, one orbit slot around $1pm$, and one orbit slot around $5pm$. Request $r_4$ is a global reservation request where the goal for the end-user is to have 30 minutes of satellite useful time over the whole day. A possible allocation is also given in the figure. In this allocation, we can see that periodic requests $r_1$ and $r_3$ get three satellite slots over the day, periodic request $r_2$ is not fully satisfied and only gets 2 satellite slots over the day, while request $r_4$ gets all its slots over satellite number 1.

For the exclusive window allocation process, a first baseline approach consists in never simultaneously considering users whose window requirements overlap, such as users having ground stations that are too near to each other. This is however not fully satisfactory since it prevents the satellite constellation manager from dealing with scenarios such as the one given in Figure 1 where ground station visibility cones associated

with different requests overlap. Another option is to use a FIFO approach where windows are allocated to users in the same order as exclusivity requirements are formulated. However, some requests can be simultaneously available since the main mission center might decide to allocate exclusive orbit slots not continuously but on an everyday basis. It can also be relevant to partially satisfy some exclusivity requirements and to share the satellites of the constellations in a fair way. In case of an over constrained problem, this also ensures that end-users all have access to satellites. Additionally, there is a need for the main mission center to keep satellite time windows for itself, to still be able to satisfy the requests formulated by all the other end-users.

All in all, the system targeted involves several constraints and many objective functions, and using a manual orbit slot allocation approach is not so easy in this case, which is why we develop an automatic orbit slot allocation process. Given the nature of the problem, we exploit a Constraint Programming (CP) engine, namely CpOptimizer, to explore various slot allocation alternatives. More specifically, we explore two options, including (1) a full CP approach where a global allocation model is defined to get a fair sharing of the satellites of the constellation among the users, and (2) a decomposition approach where the fair sharing is handled outside CpOptimizer to try and get good solutions more quickly.

With regards to existing works, the orbit slot allocation problem can be related to the shared ground station allocation problem. Basically, in this other problem, we consider a set of ground stations and a set of satellites for which communication windows must be reserved over the ground stations, and the goal is to allocate communication windows to the different satellites while satisfying numerous constraints (minimum communication duration available over a given period, minimum and maximum time between two successive communication windows, minimum duration for each communication window allocated, etc.). This problem was considered for several systems, including: (i) the *Air Force Satellite Control Network* (AFSCN), which is composed of 16 ground station antennas that receive around 500 requests for communication windows [1], and where the goal is to maximize the number of communication requests that are satisfied; the associated allocation engine is based on MIP, heuristic search, or genetic algorithms [2,3,4,1]; (ii) the *Deep Space Network* (DSN), which is used for deep space missions and missions beyond geostationary orbits, that is composed of 13 ground station antennas and used by 35 users [5]; the resolution approaches are based on MIP [6], local search [7,8], genetic algorithms [9], or interactive resolution [10,5]; (iii) the ESA Tracking network (ESTRACK) which is composed of more than 10 ground stations used by 10 missions of the European Space Agency in the version described in [11,12]; the first resolution schemes included a constructive approach that tries to satisfy one more communication request at each step and that uses temporal constraint reasoning techniques to check some basic constraints [11]; optimization was also considered based on dynamic programming and local search [12]; (iv) the academic ground stations network [13] for which communication windows must be allocated to a set of small satellites (CubeSats), and for which the fair sharing of communications among the different users matters a lot [14].

In these existing works, the goal is to share ground station time instead of sharing the satellite time, and the partial satisfaction of a single communication request is often not considered. The rest of the paper is organized as follows. Section 2 formally introduces the problem tackled. Section 3 provides a CP model of this problem. Section 4 details the decomposition approach where a CP engine is called by a global orbit slot allocation

process that takes care of all fair sharing issues. Section 5 presents experimental results for orbit slot allocation problems given a constellation containing 8 satellites. Section 6 provides perspectives for this work.

## 2. Core Concepts and Problem Definition

This section formally introduces the Orbit Slot Allocation Problem (OSAP) for a set of satellites referred to as $\mathscr{S}$. As shown below, we consider two kinds of allocation requests, namely *global allocation requests* and *time-tagged allocation requests*. A global allocation request corresponds to a demand for a minimum total duration of orbit slots over a given period, while a time-tagged allocation request expresses demands for orbit slots around a few number of fixed time references over the day. Also, given the number of requests in real world instances, a feasible solution that fulfills all requests generally does not exist. Moreover, the reward associated with a request that is not completely fulfilled is generally not null. Therefore, we consider partial fulfillment of requests that we call *mode*. Intuitively, a mode consists in considering a weaker observation pattern.

**Definition 1** (Satellite reservation window)**.** *A satellite reservation window $w$ is defined by a satellite* $\text{sat}_w$ *and a time window* $[\text{start}_w, \text{end}_w]$ *during which a reservation can be made on satellite* $\text{sat}_w$*. A reservation window $w$ also has an individual score $\omega_w$ that can depend on various features such that the angular position of the satellite with regards to points of interest (which has an impact on image quality) or cloud cover forecast.*

**Definition 2** (Allocated orbit slot)**.** *An allocated orbit slot $o$ within satellite reservation window $w$ correspond to a time window* $[\text{start}_o, \text{end}_o]$ *included in* $[\text{start}_w, \text{end}_w]$.

**Definition 3** (Global allocation request)**.** *A (multi-mode) global allocation request $r$ is defined by:*

- *a set of satellite reservation windows $\mathscr{V}_r$ during which satellites might be booked;*
- *a minimum duration* $\text{minSlotDur}_r$ *required for each reserved orbit slot;*
- *a list of allocation modes $\mathscr{M}_r = [\mathscr{M}_{r,1}, \ldots, \mathscr{M}_{r,K}]$, where each mode defines an alternative to fulfill the request and is defined by a quantity* $\text{globalDur}_m$*, for each $m \in \mathscr{M}_r$, specifying the global duration required for the orbit slots reserved over windows belonging to $\mathscr{V}_r$.*

**Definition 4** (Allocation for a global request)**.** *An allocation $\mathscr{A}_r$ for a global request $r$ is defined by a mode $m(\mathscr{A}_r) \in \mathscr{M}_r$ chosen for $r$ and by one orbit slot $\mathscr{A}_{r,w}$ booked within each reservation window $w \in \mathscr{V}_r$ (with $\mathscr{A}_{r,w} = \emptyset$ if no orbit slot is reserved within $w$).*

*An allocation $\mathscr{A}_r$ is valid if and only if (1) the sum of the duration of the orbit slots defined by $\mathscr{A}_r$ is greater than or equal to* $\text{globalDur}_{m(\mathscr{A}_r)}$*, and (2) the duration of each non-empty orbit slot $\mathscr{A}_{r,w}$ is not less than* $\text{minSlotDur}_r$.

**Assumption 1.** *For a global allocation request $r$, we assume that $\mathscr{M}_{r,i+1}$ is always preferred to mode $\mathscr{M}_{r,i}$ from the point of view of the user, or in other words that* $\text{globalDur}_{\mathscr{M}_{r,i}} < \text{globalDur}_{\mathscr{M}_{r,i+1}}$.

**Definition 5** (Time-tagged allocation request)**.** *A time-tagged allocation request $r$ is defined by:*

- *a set of satellite reservation windows $\mathscr{V}_r$ during which satellites might be booked;*
- *a minimum duration* $\text{minSlotDur}_r$ *required for each reserved orbit slot;*
- *a set of time references $\mathscr{T}_r$, with for each time reference $t \in \mathscr{T}_r$ a subset $\mathscr{V}_t^r \subseteq \mathscr{V}_r$ that defines the reservation windows associated with t, with the assumption that no reservation window is shared by two distinct time references;*
- *a list of allocation modes $\mathscr{M}_r = [\mathscr{M}_{r,1}, \ldots, \mathscr{M}_{r,K}]$, where each allocation mode $m \in \mathscr{M}_r$ is defined by a subset $\mathscr{T}_m \subseteq \mathscr{T}_r$ of time references around which an orbit slot must actually be reserved.*

**Definition 6** (Allocation for a time-tagged request)**.** *An allocation $\mathscr{A}_r$ for a time-tagged allocation request r is defined by a mode $m(\mathscr{A}_r) \in \mathscr{M}_r$ chosen for r and by one orbit slot $\mathscr{A}_{r,t}$ associated which each time reference $t \in \mathscr{T}_{m(\mathscr{A}_r)}$.*

*An allocation $\mathscr{A}_r$ satisfies request r if and only if, for each time reference $t \in \mathscr{T}_{m(\mathscr{A}_r)}$, (1) orbit slot $\mathscr{A}_{r,t}$ is contained within one of the candidate reservation windows in $\mathscr{V}_r$, and (2) the duration of slot $\mathscr{A}_{r,t}$ is not less than* $\text{minSlotDur}_r$.

**Assumption 2.** *For a time-tagged allocation request r, we assume that $\mathscr{M}_{r,i+1}$ is always preferred to mode $\mathscr{M}_{r,i}$ from the point of view of the user. More precisely, we assume that $\mathscr{T}_{\mathscr{M}_{r,i}} \subset \mathscr{T}_{\mathscr{M}_{r,i+1}}$ always holds.*

**Definition 7** (Orbit Slot Allocation Problem)**.** *An Orbit Slot Allocation Problem (OSAP) is defined by: a set of satellites $\mathscr{S}$, and a set of requests $\mathscr{R} = \mathscr{R}_G \cup \mathscr{R}_T$, with $\mathscr{R}_G$ a set of global allocation requests, and $\mathscr{R}_T$ a set of tagged-time allocation requests.*

**Definition 8** (Solution for an OSAP)**.** *A solution $\mathscr{A}$ for an OSAP is defined by one allocation $\mathscr{A}_r$ for each $r \in \mathscr{R}$. A solution is said to be feasible if and only if:*

- *for every request r, allocation $\mathscr{A}_r$ satisfies request r;*
- *for each satellite $s \in \mathscr{S}$, there is no overlapping between the orbit slots booked over s for all allocations in $\{\mathscr{A}_r \,|\, r \in \mathscr{R}\}$.*

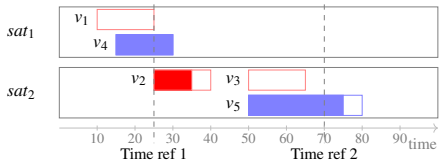
To be able to get a fair sharing of the satellites among the requests, we introduce a unified concept of reward associated with each mode *m* that represents the total satellite duration booked for *m*.

**Definition 9** (Mode reward)**.** *For a global allocation request r and a possible mode $m \in \mathscr{M}_r$, the reward $\Omega_m$ associated with m corresponds to quantity* $\text{globalDur}_m$.

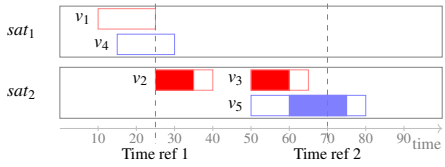
*For a time-tagged allocation request r and a possible mode $m \in \mathscr{M}_r$, the reward $\Omega_m$ associated with mode m corresponds to quantity $|\mathscr{T}_m| \cdot \text{minSlotDur}_r$, that is to the total satellite time required by m over all its relevant time references.*

This concept of reward based on a common time scale makes it easier to compare the different modes of the different requests and get a fair sharing in situations where some requests have numerous modes while others only have a few. Additionally, mode rewards can be refined in a second step based on the scores $\omega_w$ associated with the possible reservation windows, that do not need to have a common scale for all the users.

**Definition 10** (Optimal solution for an OSAP: mode utility)**.** *Let us consider an OSAP containing a list of requests $[r_1, \ldots, r_n]$. A valid solution $\mathscr{A}$ is said to be* utilitarian-




(a) Utilitarian-optimal allocation




(b) Fair-optimal allocation

**Figure 2.** Solutions associated with example 1

optimal *if it maximizes the global utility* $u(\mathscr{A}) = \sum_{r\in\mathscr{R}} \Omega_{m(\mathscr{A}_r)}$ *among all valid solutions. A valid solution* $\mathscr{A}$ *is said to be* fairness-optimal *if utility vector*

$$\vec{u}(\mathscr{A}) = [\Omega_{m(\mathscr{A}_{r_1})}, \ldots, \Omega_{m(\mathscr{A}_{r_n})}] \tag{1}$$

*is leximin-optimal, meaning that the priority is always given to the requests whose satisfaction level is the lowest (more formally, given two vectors* $a = [a_1, \ldots, a_n]$ *and* $b = [b_1, \ldots, b_n]$*, a is leximin-dominated by b if and only if when sorting a and b by increasing values, the sorted versions of a is lexicographically smaller than the sorted version of b).*

**Definition 11** (Optimal solution for an OSAP: window utility)**.** *Let* $\mathscr{A}$ *be a valid allocation for an OSAP. The window utility of* $\mathscr{A}$*, denoted* $u^{\text{slot}}(\mathscr{A})$*, is equal to the sum of rewards of visibility windows v selected in* $\mathscr{A}$*. Formally,* $u^{\text{slot}}(\mathscr{A}) = \sum_{r\in\mathscr{R}} \sum_{v\in\mathscr{A}_r} \omega_v$.

**Example 1.** *Let us consider two satellites* $\text{sat}_1$ *and* $\text{sat}_2$*, a time-tagged allocation request A and a global allocation request B. Minimum slot duration for A and B are respectively equal to* 10 *and* 15 *time units. The set of satellite reservation windows for A is* $\mathscr{V}_A = \{v_1, v_2, v_3\}$ *with* $v_1 = [10, 25]$, $v_2 = [25, 40]$, $v_3 = [50, 65]$, $\text{sat}_{v_1} = \text{sat}_1$ *and* $\text{sat}_{v_2} = \text{sat}_{v_3} = \text{sat}_2$*. Two time references* $t_1$ *and* $t_2$ *are associated with A, with* $\mathscr{V}_{r_1}^{t_1} = \{v_1, v_2\}$ *and* $\mathscr{V}_{r_1}^{t_2} = \{v_3\}$*. We consider modes* $a_1$*,* $a_2$ *and* $a_3$ *for A. Their associated time references are:* $\mathscr{T}_{a_1} = \emptyset$*,* $\mathscr{T}_{a_2} = \{t_1\}$ *and* $\mathscr{T}_{a_3} = \{t_1, t_2\}$*. Their respective rewards are* 0*,* 10 *and* 20.[1]

*The set of satellite reservation windows for B is* $\mathscr{V}_B = \{v_4, v_5\}$ *with* $v_4 = [15, 30]$, $v_5 = [50, 80]$, $\text{sat}_{v_4} = \text{sat}_1$ *and* $\text{sat}_{v_5} = \text{sat}_2$*. We consider modes* $b_1$*,* $b_2$*,* $b_3$ *for B. The global duration associated with* $b_1$*,* $b_2$ *and* $b_3$ *are respectively equal to* 0*,* 15 *and* 40 *time units.*

*An utilitarian-optimal allocation* $\mathscr{A}_{\text{util}}$ *is illustrated on Figure 2a. It selects modes* $a_2$ *for A and* $b_3$ *for B, which gives a total mode utility equal to* 50*. The utility vector associated with this allocation is* $\vec{u}(\mathscr{A}_{\text{util}}) = [10, 40]$*. Such a vector is not leximin-optimal. In fact, a fairness-optimal allocation* $\mathscr{A}_{\text{fair}}$*, illustrated on Figure 2b, consists in selecting modes* $a_3$ *for A and* $b_2$ *for B (it is not possible to reach duration of* 40 *for request B with* $v_5$ *and* $v_4$*). The associated utility vector is* $\vec{u}(\mathscr{A}_{\text{fair}}) = [20, 15]$*. This vector leximin-dominates* $\vec{u}(\mathscr{A}_{\text{util}})$*. The global utility of this allocation is* $u(\mathscr{A}_{\text{fair}}) = 35$*, which is lower than* $u(\mathscr{A}_{\text{util}})$.

**Theorem 1** (NP-hardness)**.** *OSAP is NP-hard.*

[1] Note that, depending on the satellites orbits, time references of time-tagged requests might not be included in any reservation window.

*Proof sketch.* The decision problem associated to OSAP, *i.e.* deciding whether there exists a solution with a mode utility greater than $U$ is NP-complete. In fact, checking the validity of a solution is polynomial. For NP-hardness, the NP-hard Resource-Constrained Project Scheduling Problem (RCPSP) with disjunctive resources, activities that consume at most one resource and no precedence ([15]) can be reduced to an OSAP. An RCPSP instance is defined by a set of activities $\mathscr{A}$, a set of resources $\mathscr{R}$ and a makespan $C_{max}$ to reach. Each resource $r$ in $\mathscr{R}$ is modelled by a satellite $sat_r$, each activity $a$ in $\mathscr{A}$ that consumes $r$ is modelled by a global request $g_a$ with a $minSlotDur_{g_a}$ equal to $a$'s duration. $g_a$ has a unique mode with a reward equal to 1 and with a unique reservation window $v$ on $sat_r$ such that $v = [0, C_{max}]$. It is possible to schedule all activities within $C_{max}$ if and only if the OSAP has a mode utility greater than or equal to the number of activities. □

## 3. Constraint Model for Allocation Optimization

In this section, we present CP encodings for computing utilitarian-optimal and fair-optimal allocations.

*Utilitarian CP Encoding.* We first describe the decision variables used for the utilitarian CP encoding. (i) For each request $r$ in $\mathscr{R}$ and for each reservation window $v \in \mathscr{V}_r$, $\mathsf{itv}_v$ is an interval variable in $[start_v, end_v]$. We recall that an interval variable is a CP Optimizer structure that encompasses several features: a start date, an end date and a boolean variable that represents the presence of the interval in the generated allocation. In our case, $\mathsf{itv}_v$ is optional and if present, it indicates an orbit slot is booked in $v$, along with its start and end dates. We also assign $minSlotDur_r$ as a minimum size for $\mathsf{itv}_v$. (ii) For each request $r$ in $\mathscr{R}$ and for each mode $m$ in $\mathscr{M}_r$, $\mathsf{x}_m$ is a boolean variable that is true if mode $m$ is chosen for request $r$.

We now formally define the constraints associated with an OSAP.

$$\forall r \in \mathscr{R}, \quad \sum_{m \in \mathscr{M}_r} \mathsf{x}_m = 1 \tag{2}$$

$$\forall s \in S, \quad noOverlap(\{\mathsf{itv}_v | v \in \bigcup_{r \in \mathscr{R}} \mathscr{V}_r \wedge sat_v = s\}) \tag{3}$$

$$\forall r \in \mathscr{R}_T, \forall m \in \mathscr{M}_r, \forall t \in \mathscr{T}_m, \quad \sum_{v \in \mathscr{V}_r^t} presenceOf\,(\mathsf{itv}_v) \geq \mathsf{x}_m, \tag{4}$$

$$\forall r \in \mathscr{R}_T, \forall t \in \mathscr{T}_r, \quad \sum_{v \in \mathscr{V}_t} presenceOf\,(\mathsf{itv}_v) \leq 1 \tag{5}$$

$$\forall r \in \mathscr{R}_G, \forall m \in \mathscr{M}_r, \quad \sum_{v \in \mathscr{V}_r} lengthOf\,(\mathsf{itv}_v) \geq \mathsf{x}_m \cdot globalDur_r \tag{6}$$

Through Constraint (2), exactly one mode is selected for each request. Constraint (3) expresses that orbit slots of a given satellite cannot overlap. Constraints (4) and (5) address time-tagged requests. The first ensures that at least one orbit slot is booked for each time reference of each mode that is selected. The second guarantees that at most one orbit slot is booked for each time reference. Constraint (6) addresses global requests. It expresses that the global duration of booked orbit slots reaches the global duration associated with the selected mode.

The utilitarian-optimality of an OSAP is expressed through Eq. (7) and consists in maximizing the sum of selected mode rewards.

$$\textbf{max} \quad \sum_{r\in\mathscr{R}}\sum_{m\in\mathscr{M}_r} \Omega_m \cdot \mathsf{x}_m \tag{7}$$
$$\text{s.t.} \quad (2),(3),(4),(5),(6)$$

*Leximin CP Encoding.* In order to handle leximin optimality, we follow [16] and solve several CP optimization problems. Intuitively, the $k$-th CP problem allows to compute the $k$-th component of the sorted leximin vector $\vec{u} = [u_1,\ldots,u_n]$. Each component $u_r \in [0, \mathbf{u}_r^{max}]$ represents the utility for request $r \in \mathscr{R}$. $\mathbf{u}_r^{max}$ denotes here the best utility value for request $r$ considered alone, i.e. the best mode that can be chosen. In leximin optimization, the objective is to lexicographically maximize vector $\Lambda = [\Lambda_1,\ldots,\Lambda_n]$ obtained after ordering $[u_1,\ldots,u_n]$ following an increasing order.

Suppose we have already optimized over the first $K-1$ components $[\Lambda_1,\ldots,\Lambda_{K-1}]$ of $\Lambda$, for $K \in [1..n]$. Then, one can use the program presented thereafter to optimize the $K^{\text{th}}$ component $\Lambda_K$ of the leximin profile of $\vec{u}$. In this model, $\lambda$ is a real variable representing the utility obtained at level $K$ in $\Lambda$, with $\lambda \in [\Lambda_{K-1}, \max_{r\in\mathscr{R}} \mathbf{u}_r^{max}]$ (by convention $\Lambda_0 = 0$). $\mathsf{y}_{rk}$ is a binary variable equal to 1 if request $r \in \mathscr{R}$ plays the role of the request associated with level $k \in [1..K-1]$ in $[\Lambda_1,\ldots,\Lambda_{K-1}]$, 0 otherwise. $\mathsf{u}_r$ is a real variable in $[0, \mathbf{u}_r^{max}]$ representing the utility of request $r$. The optimization of $\Lambda_K$ can be performed using the program given below:

$$\textbf{max} \quad \lambda \tag{8}$$
$$\text{s.t.} \quad (2),(3),(4),(5),(6)$$
$$\forall r \in \mathscr{R}, \quad \mathsf{u}_r = \sum_{m\in\mathscr{M}_r} \Omega_m \cdot \mathsf{x}_m \tag{9}$$
$$\forall k \in [1..K-1], \quad \sum_{r\in\mathscr{R}} \mathsf{y}_{rk} = 1 \tag{10}$$
$$\forall r \in \mathscr{R}, \quad \sum_{k\in[1..K-1]} \mathsf{y}_{rk} \leq 1 \tag{11}$$
$$\forall r \in \mathscr{R}, \quad \lambda \leq \mathsf{u}_r + M \sum_{k\in[1..K-1]} \mathsf{y}_{rk} \tag{12}$$
$$\forall r \in \mathscr{R}, \quad \mathsf{u}_r \geq \sum_{k\in[1..K-1]} \Lambda_k \cdot \mathsf{y}_{rk} \tag{13}$$

Constraint (9) expresses the utility of each request. Utility values computed at previous iterations are all allocated (10) and to exactly one request (11). In Constraint (12), $M = \max_{r\in\mathscr{R}} \mathbf{u}_r^{max}$ is used to ignore requests associated with levels strictly lower than $K$ when optimizing $\lambda$ (big-M formulation). Constraint (13) ensures that the utility obtained for the request associated with level $k$ is not less than $\Lambda_k$. To implement the leximin rule, it then suffices to solve a sequence of such problems for $K \in [2..|\mathscr{R}|]$ to optimize the value of each component of the utility profile.

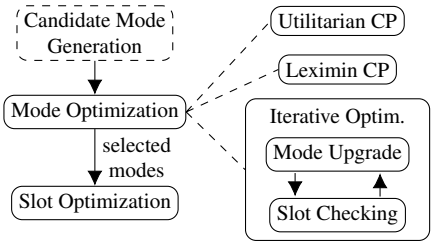



**Figure 3.** Optimization architecture

## 4. Optimization Framework and Algorithms

This section presents optimization algorithms we investigate to solve OSAP. As shown in Figure 3, the optimization architecture is composed of several modules that are executed sequentially. The first module, namely *Candidate Mode Generation*, consists in generating candidate modes to consider for each request. We present such a generation in the experiments section. The second module, *Mode Optimization* aims at optimally selecting modes wrt the utilitarian criteria or to the leximin criteria. Modes selected by this module are then given as an input to the third module, namely *Slot Optimization*, that aims at finding the best slot allocation wrt the window utility. As the mode utility is (lexicographically) more important than the window utility, splitting the optimization module into a mode dedicated one and a slot dedicated one does not remove optimal solutions from the search space and is therefore not an approximation of the problem.

*Candidate Mode Generation.* The first step consists in the generation of mode candidates. We assume that the modes are generated so that the allocation where all requests $r$ are assigned their worst modes $\mathscr{M}_{r,1}$ is a feasible solution.

*Mode optimization.* We consider three approaches for the Mode Optimization module, as illustrated on Figure 3. The *Utilitarian CP* and the *Leximin CP* approaches correspond to the encodings presented previously. As expected and as highlighted by the experimental results, Utilitarian CP (resp. Leximin CP) performs quite well wrt the utilitarian (resp. the leximin) criterion but has a lower quality wrt the leximin (resp. the utilitarian) criterion. Moreover, these approaches can struggle finding good quality solutions for large instances.

In order to handle the balance between the two criteria, we introduce a third approach, namely *Iterative Optimization*, that is composed of two layers. The first layer is entitled *Mode Upgrade* and aims at producing allocations for the requests so that Constraint (2) is satisfied and the two Criteria (7) and (8) are optimized. The mode allocation produced by this layer is then checked by the *Slot Checking* layer. This layer checks whether Constraints (2)– (6) are satisfied.

Algorithm 1 details the Mode Upgrade layer. For each request $r$, we maintain an index $idx_r$ that represents the index of the mode currently assigned to $r$. The algorithm starts by assigning the worst mode (*i.e.* mode index equal to 1) to each request (lines 2-3). Then, the set of requests that are candidates for upgrade $\mathscr{C}$ is initialised with the set of requests $\mathscr{R}$ (line 4). While there is at least one candidate, a request $r$ is chosen in $\mathscr{C}$ using a heuristic $h$ given as an input (line 6). Such a heuristic is discussed in the following. If the best mode is currently assigned to the request then it is removed from the set of

candidates (lines 7-8). Otherwise, the mode request is upgraded (line 10). If there does not exist a feasible solution with this upgrade (line 11), the mode request is downgraded and the request is removed from the set of candidates (lines 12-13). The algorithm returns the set of selected modes for each request (line 14).

**Algorithm 1:** The Mode Upgrade layer

```
1  Function modeUpgrade(𝒮, ℛ, h)
2      for r ∈ ℛ do
3          idx_r ← 1 ;
4      𝒞 ← ℛ ;
5      while 𝒞 ≠ ∅ do
6          r ← choose(𝒞, h, {ℳ_{r,idx_r} | r ∈ ℛ}) ;
7          if idx_r = |ℳ_r| then
8              𝒞 ← 𝒞 \ {r} ;
9          else
10             idx_r ← idx_r + 1 ;
11             if !check(𝒮, ℛ, {ℳ_{r,idx_r} | r ∈ ℛ}) then
12                 idx_r ← idx_r − 1 ;
13                 𝒞 ← 𝒞 \ {r} ;
14     return {ℳ_{r,idx_r} | r ∈ ℛ};
```

The Slot Checking layer is called through the check function (line 11). Such a function takes as an input the set of satellites, the set of requests and a current mode allocation. These inputs are encoded following the utilitarian CP encoding presented in Section 3 except that variables of type $x_m$ have a fixed value that depends on the selected modes. Formally, $\forall r \in \mathscr{R}, \forall m \in \mathscr{M}_r$, we fix $x_m = 1$ iff $m$ is equal to $\mathscr{M}_{r,idx_r}$.

In this paper, we consider two different heuristics for choosing request $r$ in the set of candidates. The first heuristic, denoted $h^{util}$, selects the request whose next mode increases the most the global utility of the allocation. Formally, $choose(\mathscr{C}, h^{util}, \{\mathscr{M}_{r,idx_r} | r \in \mathscr{R}\}) = argmax_{r \in \mathscr{C}} \Delta_{util}(r)$ where $\Delta_{util}(r) = \Omega_{\mathscr{M}_{r,idx_r+1}} - \Omega_{\mathscr{M}_{r,idx_r}}$ if $idx_r < |\mathscr{M}_r|$, 0 otherwise. Such a heuristic tends to favour the utilitarian criterion. The second heuristic, denoted $h^{fair}$, selects the request with the smallest utility. Formally, $choose(\mathscr{C}, h^{fair}, \{\mathscr{M}_{r,idx_r} | r \in \mathscr{R}\}) = argmin_{r \in \mathscr{C}} \Omega_{\mathscr{M}_{r,idx_r}}$. Such a heuristic allows to favour the leximin optimality as it upgrades the request having the lowest reward.

*Slot optimization.* The final step of the optimization architecture consists in optimizing the slots for the modes that have been selected at the Mode Optimization step. To do so, we consider that for each mode $m$, variable $x_m = 1$ iff $m$ is selected by the Mode Optimization module. Then, we solve the following problem, where Criterion 14 consists in maximizing the window utility as defined in Definition 11:

$$\textbf{max} \quad \sum_{r \in \mathscr{R}} \sum_{v \in \mathscr{V}_r} \omega_v \cdot presenceOf(\mathsf{itv}_v) \tag{14}$$

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

## 5. Experimental Evaluation

We evaluate the approaches presented in the paper on realistic benchmarks. We first present the experimental setup and then analyze results obtained on several instances.

We consider a Low-Earth Orbit constellation (500km altitude). The constellation is composed of 8 orbital planes with a 60 degrees inclination. Each orbital plan contains one satellite. This constellation setting is used for determining the orbit slots that can be requested for a given point on Earth, using a spatial mechanics library. We consider allocation requests over Europe national capitals. The generation protocol first consists in (1) randomly selecting a subset of national capitals $\mathscr{C}$ (10 in the experiments presented in the paper), (2) specifying the respective numbers of time-tagged and global allocation requests for each instance, and (3) randomly picking a national capital in $\mathscr{C}$ as a ground station for each of these requests. Then, the request features are generated as follows. For each global allocation request $r$, the most preferred mode global duration is randomly chosen in interval [2 hours, 4 hours]. Then, the less preferred modes are generated by iteratively retrieving 30 minutes to this global duration, until reaching 0 for the less preferred mode $\mathscr{M}_{r,1}$. The minimum slot duration is randomly chosen in interval [2 minutes, 4 minutes]. For each time-tagged allocation request $r$, we consider two time references patterns. The first one is composed of time references 8am, 12pm, 4pm, 8pm. The second one contains references 9am, 1pm, 5pm. For each time reference $t$, we consider only reservation windows contained in interval $[t-1\text{hour}, t+1\text{hour}]$. The less preferred mode $\mathscr{M}_{r,1}$ has an empty set of time references. Preferred modes are generated by iteratively adding one time reference (randomly picked) until reaching the complete set of time references. We generated 5 different order book instances per configuration (defined by $|\mathscr{R}_G|$ and $|\mathscr{R}_T|$), and present the averaged values in the following tables. For slot optimization, the reward is in [0,1] and is linear wrt to the distance to the time reference for time-tagged requests or wrt the number of slots used for fulfilling modes for global requests. The number of requests we consider is larger than current realistic data but allows to assess the capacity for the approach to scale up.

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. The version of CP Optimizer included in IBM ILOG CPLEX Studio 20.1 is used by the solvers through the Java API. CP Optimizer is called by all the approaches we consider and can use the 20 cores. We set different timeouts, as defined in the following table:

| Method | Mode opt. | Slot opt. | Slot check |
|---|---|---|---|
| upgrade − util | n/a | 300s | 120s |
| upgrade − fair | n/a | 300s | 300s |
| cp − util / cp − fair | 300s×$\lvert\mathscr{R}\rvert$ | 300s | 300s |

Table 1 and Table 2 respectively present the results in terms of utility and computation time for each algorithm and order book configuration. We display results for order books containing only time-tagged allocation requests (lines 1-5), only global allocation requests (lines 6-10), and as many time-tagged requests and global ones (lines 11-15).

In Table 1, we show the overall mode utility $u$ and the window utility $u^{slot}$. For configurations with only time-tagged allocation requests, cp − util and upgrade − util provide the best mode utilitarian allocation. Moreover, utility-optimal and fairness-optimal allocations are respectively returned by cp − util and cp − fair for the small instances. For

| configurations | | | | cp – fair | | cp – util | | upgrade – fair | | upgrade – util | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lvert\mathscr{R}_G\rvert$ | $\lvert\mathscr{R}_T\rvert$ | $\lvert\mathscr{M}\rvert$ | $\lvert\mathscr{V}\rvert$ | $u$ | $u^{slot}$ | $u$ | $u^{slot}$ | $u$ | $u^{slot}$ | $u$ | $u^{slot}$ |
| 0 | 5 | 22.0 | 107.0 | **1980.20**† | **4.43** | **1980.20**∗ | **4.43** | **1980.20** | **4.43** | **1980.20** | **4.43** |
| 0 | 10 | 44.6 | 218.2 | 3925.00† | **8.85** | **3953.40**∗ | 8.66 | 3925.00 | **8.85** | **3953.40** | 8.66 |
| 0 | 15 | 67.2 | 326.2 | 6260.40† | 13.15 | **6288.80**∗ | 12.96 | 6260.40 | **13.16** | **6288.80** | 12.96 |
| 0 | 20 | 90.0 | 439.6 | 8294.00† | **17.27** | **8322.40** | 17.06 | 8294.00 | 17.25 | **8322.40** | 17.03 |
| 0 | 25 | 112.0 | 549.8 | 10313.20 | 21.09 | **10341.60** | 20.94 | 10313.20 | **21.16** | 10276.60 | 20.78 |
| 5 | 0 | 31.4 | 198.6 | 39874.00 | **4.64** | 39911.20 | 4.50 | **42394.00** | 4.31 | 42034.00 | 4.31 |
| 10 | 0 | 63.8 | 405.0 | **44646.60** | 9.20 | 42953.60 | 8.20 | 44286.60 | **9.32** | 44286.60 | 9.27 |
| 15 | 0 | 96.4 | 606.2 | 42109.20 | 13.29 | 42730.20 | 10.76 | 44291.60 | 13.51 | **44420.00** | **13.90** |
| 20 | 0 | 129.6 | 814.6 | 27927.20 | 9.80 | 40992.60 | 9.32 | 43131.20 | **14.14** | **43409.00** | 13.86 |
| 25 | 0 | 161.4 | 1018.2 | 28864.80 | 9.80 | 40489.20 | 9.30 | 39645.40 | **13.87** | **43117.40** | 13.23 |
| 5 | 5 | 53.8 | 311.0 | 39515.60 | **8.97** | 40998.60 | 8.43 | 42395.60 | 8.30 | **44388.00** | 7.23 |
| 10 | 10 | 109.6 | 627.0 | 43594.40 | 15.93 | 42664.40 | 15.52 | 44674.40 | **15.98** | **47071.60** | 13.98 |
| 15 | 15 | 165.2 | 944.0 | 34171.40 | **23.29** | 39015.00 | 20.00 | 46368.80 | 21.74 | **47244.60** | 19.10 |
| 20 | 20 | 219.4 | 1258.2 | 31823.00 | **26.31** | 41759.80 | 24.18 | 45223.60 | 25.16 | **47728.40** | 19.45 |
| 25 | 25 | 274.6 | 1572.8 | 29788.40 | **28.87** | 41641.00 | 26.05 | 46824.60 | 27.16 | **47474.80** | 21.24 |

**Table 1.** Utility results (∗ and † respectively indicate utility-optimality and fairness-optimality results)

| configurations | | | | cp – fair | | cp – util | | upgrade – fair | | upgrade – util | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lvert\mathscr{R}_G\rvert$ | $\lvert\mathscr{R}_T\rvert$ | $\lvert\mathscr{M}\rvert$ | $\lvert\mathscr{V}\rvert$ | mode | slot | mode | slot | mode | slot | mode | slot |
| 0 | 5 | 22.0 | 107.0 | 6.34 | 2.09 | 12.96 | **1.63** | **4.24** | 1.96 | 4.89 | 2.32 |
| 0 | 10 | 44.6 | 218.2 | 77.79 | **3.93** | 45.96 | 3.94 | **9.73** | 4.00 | 10.45 | 3.79 |
| 0 | 15 | 67.2 | 326.2 | **164.48** | 242.04 | 439.87 | 242.63 | 9.41 | 243.74 | 9.63 | **241.06** |
| 0 | 20 | 90.0 | 439.6 | 195.42 | 300.38 | 6000.21 | 300.09 | **9.50** | 300.13 | 10.19 | 300.36 |
| 0 | 25 | 112.0 | 549.8 | 759.76 | 300.16 | 7500.19 | 300.09 | **237.50** | 300.25 | 294.00 | 300.25 |
| 5 | 0 | 31.4 | 198.6 | 1500.57 | 300.04 | 1500.14 | 300.06 | 390.88 | 300.03 | **339.86** | 300.04 |
| 10 | 0 | 63.8 | 405.0 | 3002.12 | 300.08 | 3700.13 | 300.13 | **1249.88** | 300.06 | 1279.68 | 300.06 |
| 15 | 0 | 96.4 | 606.2 | 4502.34 | 300.06 | 4500.43 | 300.12 | 1849.59 | 300.11 | **1834.56** | 300.05 |
| 20 | 0 | 129.6 | 814.6 | 6003.48 | 300.07 | 6000.23 | 300.07 | **2496.88** | 300.08 | 2526.24 | 300.08 |
| 25 | 0 | 161.4 | 1018.2 | 7504.20 | 300.07 | 7500.22 | 300.07 | 3137.79 | 300.10 | **3074.31** | 300.09 |
| 5 | 5 | 53.8 | 311.0 | 1504.79 | 300.05 | 3000.13 | 300.04 | **420.16** | 300.09 | 433.25 | 300.06 |
| 10 | 10 | 109.6 | 627.0 | 3029.96 | 300.08 | 6000.21 | 300.09 | **1206.61** | 300.08 | 2026.38 | 300.06 |
| 15 | 15 | 165.2 | 944.0 | 8926.65 | 300.09 | 9000.24 | 300.14 | **2410.81** | 300.10 | 3530.65 | 300.08 |
| 20 | 20 | 219.4 | 1258.2 | 12011.00 | 300.15 | 12000.43 | 300.12 | **3438.31** | 300.16 | 4494.19 | 300.10 |
| 25 | 25 | 274.6 | 1572.8 | 15014.64 | 300.19 | 15000.44 | 300.14 | **5187.83** | 300.17 | 6019.73 | 300.11 |

**Table 2.** Computation time results (values in seconds)

order books containing more than 5 time-tagged requests, utilitarian and fair approaches clearly converge to different optima. Configurations with global allocation requests are more rewarding but difficult to solve by cp – fair and cp – util (larger timeouts did not allow to get optimal results), and upgrade – util performs better in terms of utility in most of the settings. upgrade – fair also outputs quite good utilitarian allocations. Thus, the two proposed upgrade schemes are relevant approaches wrt utilitarianism. However, there is no clear winner for the windows utility optimization.

Looking at Table 2, upgrade-based heuristic methods clearly outperform optimal constraint-programming-based ones (which both require all the time budget we set, even with much larger timeouts not presented here). Even for smaller instances, all methods achieve the slot optimization timeout we set, which explains why there is no clear winner concerning the window utility criteria. On large instances containing both types of requests, it is clear that upgrade – fair is the fastest.

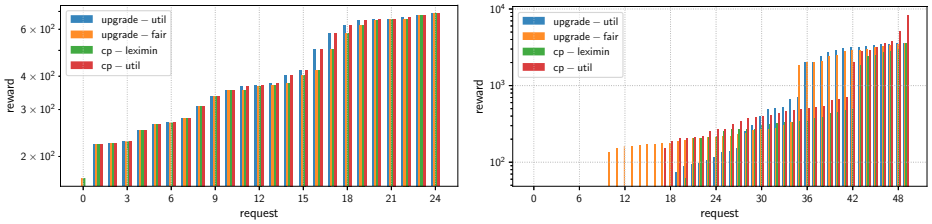



**Figure 4.** Utility profiles for two sample instances. Left: an instance with 25 time-tagged allocation requests. Right: an instance with 25 time-tagged allocation requests and 25 global allocation requests

To analyze the fairness of the resulting allocations, we provide in supplementary materials the utility profiles obtained for all instances. Figure 4 presents the utility profiles obtained for two sample instances for each algorithm. Looking at the instance with only time-tagged allocation requests (left), we can observe that all the methods behave quite similarly. Indeed, such problems are not too constrained, given the constellation configuration. It is always possible to serve all the requests, and the fair approaches only serve one more request (#0). The situation on the right, with 25 global allocation requests and 25 time-tagged allocation requests is far more informative about the behavior of each method. Let's note that the most rewarding requests (25 to 49) are the global allocation ones. We can remark that some requests cannot be fulfilled: 10 requests are not served by the fairest method. upgrade – fair serves more requests; even more than cp – fair, since the latter rapidly reaches the time budget without being able to serve some requests. upgrade – fair is even beating cp – util on some requests (35 to 42). On its side, upgrade – util is also better than cp – util on most of the high reward requests. All in all, both upgrade – fair and upgrade – util behave very well on the fairness side. Such a behaviour can be observed for most of profile utilities in supplementary materials. Therefore, upgrade – fair represents a very good trade-off on both utility and fairness optimization criteria.

## 6. Conclusions

In this paper, we modelled a novel problem (OSAP) consisting in allocating orbit slots from an Earth observation constellation to several users, with some utilitarian and fairness objectives. We considered two types of requests: time-tagged requests (asking for observations at some time plots at a given frequency) and systematic requests (asking for orbit slots every time it is possible). We proposed a two-level optimization framework, sequencing mode optimization and slot optimization. Four solution methods have been evaluated on randomly generated order books requesting orbit slots on a realistic constellation. Experimental evaluation shows that global allocation requests are the hardest ones to fulfil, and that cp – util and cp – fair do not scale on larger instances with both global and time-tagged allocation requests, while iterative upgrading methods result in good quality solutions (wrt. utility for upgrade – util and fairness for upgrade – fair) and are 3 times faster on larger instances.

This study paves the way for future research. We notably aim at investigating other types of requests and mode selection. Indeed, we only looked at requests targeting single point on Earth, while some clients may have interest in imaging large areas. Moreover,

we only considered simple mode generation in our paper, while the number of modes is exponential, and may require a dedicated effort for searching in the mode space, and thus guiding the mode upgrade procedure. Finally, we will also explore a complementary approach consisting in degrading maximum utility mode selections until they become feasible, instead of upgrading bad quality mode selections.

## Acknowledgements

This work has been performed with the support of the French government in the context of the Programme dInvertissements dAvenir, namely by the BPI PSPC LiChIE project (Lion Chaine Image Elargie), coordinated by Airbus Defence and Space.

## References

[1] Barbulescu L, Watson JP, Whitley LD, Howe AE. Scheduling space-ground communications for the Air Force satellite control network. Journal of Scheduling. 2004;7:7-34.
[2] Gooley TD. Automating the Satellite Range Scheduling Process; 1993.
[3] Schalck SM. Automating Satellite Range Scheduling; 1993.
[4] Parish DA. A Genetic Algorithm Approach to Automating Satellite Range Scheduling; 1994.
[5] Johnston MD, Tran D. Automated Scheduling for NASA's Deep Space Network. In: Proc. of the 7th International Workshop on Planning and Scheduling for Space (IWPSS-11); 2011. p. 1-10.
[6] Bell CE. Scheduling deep-space network data transmissions: a Lagrangian relaxation approach. In: Proc. of SPIE 1963, Applications of Artificial Intelligence. Orlando, FL, USA; 1993. p. 330-40.
[7] Chien S, Lam R, Vu Q. Resource scheduling for a network of communication antennas. In: Proc. of the IEEE Aerospace Conference. Aspen, CO, USA; 1997. p. 361-73.
[8] Chien S, Rabideau G, Knight R, Sherwood R, Engelhardt B, Mutz D, et al. ASPEN: Automated Planning and Scheduling for Space Mission Operations. In: Proc. of the 6th International Symposium on Space Operations at the Start of the 3rd Millennium (SpaceOps-00). Toulouse, France; 2000. .
[9] Guillaume A, Seugnwon L, Wang YF, Zheng H, Hovden R, Chau S, et al. Deep Space Network Scheduling Using Evolutionary Computational Methods. In: Proc. of IEEE Aerospace Conference; 2007. p. 1-6.
[10] Johnston MD, Clement BJ. Automating Deep Space Network Scheduling and Conflict Resolution. In: Proc. of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06); 2006. p. 14831489.
[11] Damiani S, Dreihahn H, Noll J, Niézette M, Calzolari GP. Automated Allocation of ESA Ground Station Network Services. In: Proc. of the 5th International Workshop on Planning and Scheduling for Space (IWPSS-06); 2006. p. 1-10.
[12] Hoffmann A, Dreihahn H, Niézette M, Theis G. Improving Performance and Interoperability of the ESTRACK Planning System. In: Proc. of the 6th International Workshop on Planning and Scheduling for Space (IWPSS-09); 2009. p. 1-9.
[13] Schmidt M, Schilling K. A Scheduling System with Redundant Scheduling Capabilities. In: Proc. of the 6th International Workshop on Planning and Scheduling for Space (IWPSS-09). Pasadena, CA, USA; 2009. p. 1-6.
[14] Schmidt M, Schilling K. Enhanced Redundant Scheduling Capability for Low Cost Ground Station Networks. In: Proc. of the 7th International Workshop on Planning and Scheduling for Space (IWPSS-11). Darmstadt, Germany; 2011. p. 1-6.
[15] Ganian R, Hamm T, Mescoff G. The Complexity Landscape of Resource-Constrained Scheduling. In: Bessiere C, editor. Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20. International Joint Conferences on Artificial Intelligence Organization; 2020. p. 1741-7. Available from: https://doi.org/10.24963/ijcai.2020/241.
[16] Kurokawa D, Procaccia AD, Shah N. Leximin Allocations in the Real World. ACM Transactions on Economics and Computation. 2018;6(34). Available from: https://doi.org/10.1145/3274641.