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

doi:10.3233/FAIA220068

Parallel Scheduling of Complex Requests for a Constellation of Earth Observing Satellites

Samuel SQUILLACI [a,1], Stéphanie ROUSSEL [a] and Cédric PRALET [a]
[a] *ONERA/DTIS, Toulouse, France*

Abstract. Nowadays, the Earth observation systems involve multiple satellites, multiple ground stations, and multiple end-users that formulate various observation requests. These requests might be heterogeneous (stereoscopic observations, periodic observations, systematic observations, etc.), and one difficulty is that the search space defined by the possible ways of performing the requests given the multiple satellites and ground stations available is huge. This paper studies several combinatorial optimization techniques for solving such an operational problem, including a constraint programming approach and parallel scheduling techniques that take advantage of the problem structure. These algorithms are evaluated on realistic instances involving various request types and objective functions depending on the cloud cover conditions, that highly impact the quality of the images collected.

Keywords. Scheduling, Parallel processing, Constellation of satellites

1. Introduction

Earth Observing Satellites (EOSs) are used for various needs, including the surveillance of points of interest (*e.g.* for defence or environmental issues) or the punctual observations of areas of interest (*e.g.* for natural disaster monitoring or for mapping requirements). Over the years, many satellite systems were designed, and nowadays there is a growing development of constellations of EOSs, including several already active systems. In this paper, we consider a future Earth observation system designed by an industrial partner for this study. This system is composed of 16 low Earth orbit satellites and multiple ground stations. Based on this system, one ambition is to reduce the total time required to fulfil observation requests over relevant areas at the Earth surface. Another objective is to answer new kinds of requests formulated by the end-users, such as high frequency periodic requests (*e.g.*, observation of a given area four times per day) or systematic requests (observe an area of interest *a* each time a satellite of the constellation flies over *a*). For this new system, a key point concerns the development of a mission planning tool able to take as an input a set of candidate imaging requests and a cloud cover forecast, and return as an output a mission plan for each satellite of the constellation over the next time period, *e.g.* for the next 24 hours. This raises several technical challenges.

[1]Corresponding Author: S. Squillaci; E-mail: samuel.squillaci@onera.fr.

- As usual for Earth observing satellites, the problem is generally over-constrained, meaning that there is a need to select the subset of requests for which observation activities will be performed over the next planning period.
- The mission planner must deal with heterogeneous requests. This implies that it must be able to satisfy specific constraints associated with each request: for instance, for so-called stereoscopic requests, there is need to perform two acquisitions of a given area in a single pass of a satellite. This also implies that the planner must be capable of comparing two requests that might require very different numbers of images and that might have very different cloud cover conditions.
- There may exist numerous alternatives to satisfy complex requests such as periodic ones. For instance, if a user wants to observe a given ground area at 8am, 11am, 14pm, and 17pm each day, and if there are 5 observation opportunities around each time reference, we end up with 625 combinations *for a single request*.
- The satellites we consider must be pointed to targets during observations and pointed to ground stations for transferring data. As a result, observations and data transfers cannot be performed in parallel.
- In addition to the complexity of the problem, the mission planner must be very reactive, and we consider that we have approximately 5 minutes to build a plan that is updated several times per day.

To answer these challenges, we study several optimization techniques for defining the observation and data download activities of the satellites. Some of these techniques try to exploit parallel search, the main idea being to take advantage of the structure of the problem from the point of view of the temporal constraints. Indeed, the set of candidate observation and download activities can be partitioned since there are several independent satellites and usually several independent orbits for each satellite. For instance, with 16 satellites performing 15 revolutions per day, we obtain at least 240 subproblems that are independent from a temporal point of view, and there might even exist further decompositions within each orbit.

The rest of the paper is organized as follows. We first describe related works. We then formalize the problem considered. After that, we present several scheduling algorithms that exploit problem decomposition and parallel search. Last, we provide experimental results over realistic scenarios together with perspectives for this work.

2. Related Works

Several techniques were proposed in the last decade to define an observation selection and scheduling tool for a constellation of satellites. Some authors proposed to tackle a unique global optimization problem to define the activities of the satellites, based on *Branch & Price* [1,2] or MILP [3,4]. Several incomplete search methods were also tested such as tabu search [5], evolutionary algorithms [6,7], simulated annealing [8], or Pareto Search when multiple objective functions must be optimized [9,10].

Other approaches explicitly identify on one hand an assignment problem that consists in determining the satellite that should fulfil a given request among the satellites of the constellation, and on the other hand a scheduling problem that consists in ordering all the activities assigned to a given satellite so as to satisfy the temporal constraints. For instance, [11] handles the assignment problem by selecting at each step an observation job maximizing a given scoring function and assigning this job to a satellite whose workload is the lowest. [12] assigns jobs to satellites based on randomized heuristics that take into account various features, such as the size of the observation windows or the observation angle. It also couples this process with an Adaptive Large Neighborhood Search (ALNS) that uses destroy and repair operations to try and decrease the number of unscheduled jobs. In another direction, [13] assigns each job to the satellite that offers the highest scheduling success probability according to a neural network success prediction model and then performs tabu search to get a plan. Last, [14] studies assignment strategies where the difficulty of inserting an observation in the plan of a given satellite is taken into account. With regards to these previous works, one issue is that for the system we consider, the central mission planner must deal with more complex observation requests, such as the periodic imaging requests mentioned before.

A few studies have also been performed for complex observation requests (not limited to a single image acquisition). For instance, [15] optimizes a mission plan to reduce the error of a soil moisture estimator model based on observations made by different instruments. A beam search is proposed to build such a plan. [16] introduces *modes* coupling observations and data downloads to represent different ways to cover requests, together with an LNS algorithm based on successive insertions and removals of modes.

As for parallel search, problem decomposition has already been exploited for satellite constellations. [13] decomposes the problem into a prediction step that evaluates the probability to successfully insert an observation in a given satellite plan, and a parallel solving step that schedules the activities of every satellite. In this context, parallel solving is possible since the plan of each satellite is built *after* dispatching the observations. [17] proposes an auction-based algorithm to share the satellite resources between users computing bids in parallel. Parallel search techniques were also developed for planning problems in general, like parallel A*, but we focus our literature review on parallel search for satellite constellation planning.

3. Problem Modeling

We now describe the scheduling problem we have to solve given a set of satellites $\mathcal{S}$. We do not consider memory or energy capacity constraints since the main bottleneck for the specific satellites we are working on comes from the temporal constraints.

3.1. Observation Requests

An observation request is defined by a polygon at the Earth surface and by an observation pattern specified by the end-user posting the request. In the following, we consider polygons requiring a single picture from a satellite, together with four possible *request types*:

- *monoscopic requests*: need to take a single picture of the polygon;
- *stereoscopic requests*: need to take two pictures from two different angles, using a single pass of a satellite;
- *periodic requests*: need to cover the polygon periodically (*e.g.* 4 times per day);
- *systematic requests*: need to take a picture at each pass of a unique satellite over the polygon.

The set of observation requests is denoted by $\mathcal{R}$.

3.2. Observation Opportunities

For each request $r \in \mathcal{R}$, there is a set of observation opportunities $\mathcal{O}_r$. Each opportunity $o \in \mathcal{O}_r$ corresponds to a time window $[Start_o, End_o]$ during which a satellite Sat_o can take a picture of the corresponding ground target area. Taking a picture takes a duration δ_o within window $[Start_o, End_o]$, and there is a flexibility concerning the start time of the observation within this window. The set of observation opportunities associated with satellite s is referred to as $\mathcal{O}_s$ ($\mathcal{O}_s = \cup_{r \in \mathcal{R}} \{o \in \mathcal{O}_r \,|\, Sat_o = s\}$).

3.3. Request Modes

To fulfil a given request $r \in \mathcal{R}$, it is not necessary to use all observation opportunities in $\mathcal{O}_r$. In the following, a subset of $\mathcal{O}_r$ that suffices to complete request r is called a *mode* for r, following the terminology used in scheduling problems like RCPSPs with modes. The subset of observation opportunities associated with a mode m is denoted by $\mathcal{O}_m$ and the set of possible modes for r is referred to as $\mathcal{M}_r$. Set $\mathcal{M}_r$ can be large but it does not need to be defined in extension, and some methods studied thereafter are able to generate new modes on-the-fly. Additionally, modes may have different qualities. For instance, for monoscopic requests, there is a unique observation opportunity o in each mode and the quality of the mode depends on features like the predicted cloud coverage for o given the longitude and latitude of the area of interest and the cloud cover forecast. For periodic requests, a mode is composed of several observation opportunities and its quality might depend on the way the period is respected. To get a generic model, we simply associate a global reward ω_m with each mode m. Last, $\mathcal{M}$ denotes the set of all modes in the problem ($\mathcal{M} = \bigcup_{r \in \mathcal{R}} \mathcal{M}_r$).

3.4. Download Opportunities

Several ground stations are available to transfer acquisition data while the satellites move on their orbits, and the satellites can communicate with these ground stations only during some specific time periods. Formally, each download opportunity d associated with a given satellite is defined by a time window $[Start_d, End_d]$. To separate the download planning problem from the observation planning problem, we consider that an interval of duration Δ_d must be booked within each download window $[Start_d, End_d]$, the precise start time of this interval being flexible, as for the observation activities. The set of download opportunities associated with satellite s is referred to as $\mathcal{D}_s$.

3.5. Activities and Transition Times

The set of candidate activities $\mathcal{A}_s$ associated with a satellite s combines the set of observation opportunities $\mathcal{O}_s$ for s and the set of download opportunities $\mathcal{D}_s$ for s ($\mathcal{A}_s = \mathcal{O}_s \cup \mathcal{D}_s$). The activities in $\mathcal{O}_s$ are optional and their presences depend on the modes chosen for the requests, while the activities in $\mathcal{D}_s$ are mandatory. Also, for each pair of activities $a, b \in \mathcal{A}_s$, there is a minimum duration τ_{ab} required to perform a maneuver between the two pointing directions corresponding to a and b. The set of activities is $\mathcal{A} = \cup_{s \in \mathcal{S}} \mathcal{A}_s$.

3.6. Optimization Problem

A solution plan σ is defined by:

- a subset $\mathcal{R}(\sigma) \subseteq \mathcal{R}$ of requests that are selected in σ;
- for each request $r \in \mathcal{R}(\sigma)$, a mode $m_r(\sigma)$ chosen for r;
- for each satellite s, a sequence $\Pi_s(\sigma)$ containing all download opportunities in $\mathcal{D}_s$ and all observation opportunities in $\mathcal{O}_s$ that are involved in the selected modes.

A solution plan σ is valid if sequence $\Pi_s(\sigma)$ is feasible from a temporal point of view, taking into account the time windows and the minimum transition times between the activities. Our objective is to find a valid solution plan maximizing the sum of the rewards of the selected modes.

3.7. Problem Decomposition: Connected Components of Activities (CCAs)

We now introduce problem decompositions that will be exploited by the solving techniques defined in the next sections. Basically, we can identify groups of activities that are independent from a temporal point of view. To do this, it suffices to build, for each satellite s, a graph G_s containing one node per activity $a \in \mathcal{A}_s$ and one edge per pair of activities $a, b \in \mathcal{A}_s$ such that $Start_a \leq Start_b$ and $Start_b < End_a + \tau_{ab}$ hold (case of a direct possible temporal interaction between a and b).

The set of connected components of graphs G_s over all satellites s is denoted by $\mathcal{CCA}$ like “connected components of activities”, and each connected component in $\mathcal{CCA}$ is called a CCA. The main idea is then that *once the request modes are chosen*, the activities located in two distinct CCAs are independent and can be scheduled in parallel.

Moreover, in a solution σ, the global sequence of activities of satellite s can be equivalently represented as a set of sequences containing one sequence per CCA of s. More formally, in a solution σ, we can decompose $\Pi_s(\sigma)$ as $\Pi_s(\sigma) = \cup_{c \in \mathcal{CCA}_s} \Pi_c(\sigma)$ where $\mathcal{CCA}_s$ denotes the set of CCAs related to satellite s and $\Pi_c(\sigma)$ stands for the sequence of activities planned in CCA c.

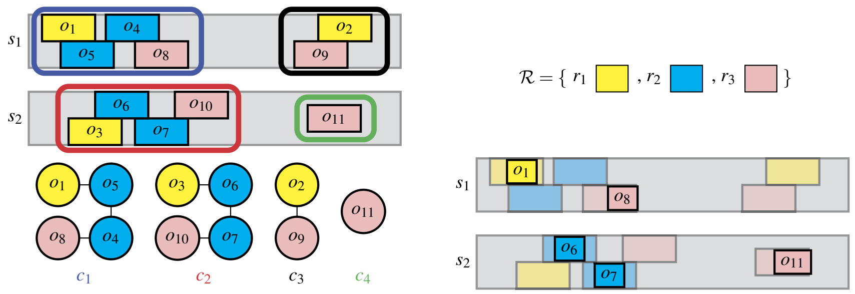

Figure 1. Example of CCAs c_1, c_2, c_3, c_4 (left) and example of a solution (right)

Figure 1 illustrates CCAs for a problem involving 2 satellites (s_1, s_2) and 3 requests (r_1, r_2, r_3). Request r_1 is monoscopic and has 3 modes $m_1^1 = \{o_1\}$, $m_2^1 = \{o_2\}$, $m_3^1 = \{o_3\}$. Request r_2 is stereoscopic so each mode is composed of two observations opportunities: $m_1^2 = \{o_6, o_7\}$, $m_2^2 = \{o_4, o_5\}$. Request r_3 is periodic and consists in observing a ground area twice. Its modes are $m_1^3 = \{o_{10}, o_{11}\}$, $m_2^3 = \{o_8, o_{11}\}$, $m_3^3 = \{o_8, o_9\}$, $m_4^3 = \{o_{10}, o_9\}$.

If we consider null transition times, there are 4 CCAs c_1, c_2, c_3, c_4. As o_1 and o_5 are observation opportunities that overlap, they are connected in G_{s_1} and belong to the same CCA (c_1). Opportunities o_5 and o_8 do not overlap, but they belong to the same CCA because they are indirectly connected through o_4.

4. Global CP Approach

To solve the problem defined in the previous section, a first idea is to directly reuse an existing combinatorial optimization engine. More precisely, we reuse a Constraint Programming (CP) tool, namely IBM Ilog CP Optimizer (CPO) that offers both a powerful declarative modeling language and efficient constraint-based scheduling techniques improved over the years. This approach is referred to as GCPS for *Global Constraint Programming Search*. The model exploited by GCPS considers several decision variables:

- $\forall m \in \mathcal{M}$, $\mathbf{y}_m \in \{0,1\}$: selection of mode m; as the number of possible modes can be very large, we reduce the number of modes in $\mathcal{M}$ by considering only the k best modes in terms of rewards for each request;
- $\forall s \in \mathcal{S}$, $\forall o \in \mathcal{O}_s$, $\mathbf{itv}_o$ in $[Start_o, End_o]$ and of duration δ_o: interval variable corresponding to observation opportunity o; in constraint-based scheduling, an interval variable is a composite object that covers the start time and end time of a task, its duration, and its presence (0 or 1) in the schedule, referred to as $presenceOf(\mathbf{itv}_o)$;
- $\forall s \in \mathcal{S}, \forall d \in \mathcal{D}_s$, $\mathbf{itv}_d$ in $[Start_d, End_d]$ and of duration Δ_d: interval variable representing the slot booked for satellite s in download opportunity d.

The problem to solve is then defined as follows. The goal is to maximize the total reward (1). At most one mode is selected for each request (2). The interval associated with each observation is present iff one mode containing it is selected (3). Intervals associated with download opportunities are necessarily selected (4). For each connected component of activities c in $\mathcal{CCA}$, there is no overlap between the intervals of activities in c given the required transition times expressed in τ (5). The latter constraint uses the specific *noOverlap* function available in CPO.

$$maximize \sum_{m \in \mathcal{M}} \omega_m \cdot \mathbf{y}_m \tag{1}$$

$$\forall r \in \mathcal{R}, \sum_{m \in \mathcal{M}_r} \mathbf{y}_m \leq 1 \tag{2}$$

$$\forall s \in \mathcal{S}, \forall o \in \mathcal{O}_s, presenceOf(\mathbf{itv}_o) = \sum_{m \in \mathcal{M} \,|\, o \in \mathcal{O}_m} \mathbf{y}_m \tag{3}$$

$$\forall s \in \mathcal{S}, \forall d \in \mathcal{D}_s, presenceOf(\mathbf{itv}_d) = 1 \tag{4}$$

$$\forall c \in \mathcal{CCA}, noOverlap(\{\mathbf{itv}_a \,|\, a \in c\}, \tau) \tag{5}$$

Thanks to the preprocessing performed at the level of modes, the model obtained using the CP technology is clear and compact, and as shown in the experiments it gives very good results. However, from an operational point of view, it has several drawbacks. First, for the end-users, there is a need to dispose of a robust anytime mission planner that quickly delivers good-quality solutions, and constructive solvers that start from an empty plan and insert requests one by one in the current solution are often preferred. Such methods are standard for operational satellite mission planning systems. Second, the model presented in this paper makes some simplifying assumptions with regards to the transition times between activities, which are actually time-dependent in the full version of the problem. To the best of our knowledge, even if there exist works on time-dependent constraints in CPO [18], the corresponding techniques are not available yet in the public version of the solver. For these reasons, we study constructive search techniques for which we are sure that they can be extended to a time-dependent context.

5. Parallel CCA-based Search (PCCAS)

We now study constructive search methods that insert requests one by one into the satellite plans. As mentioned before, a large number of CCAs increases the opportunity to perform computations in parallel, and the second kind of approaches studied is called *Parallel CCA-based Search* (PCCAS). Compared to GCPS, this approach does not require to compute all request modes beforehand. Instead, it starts with the best mode for each request and generates new modes on-the-fly when some activity insertions fail.

In PCCAS, we use a controller solver that maintains a global view of the problem together with worker solvers performing computations over a single CCA at each step. The controller solver initially computes all CCAs of the problem, based on all candidate activities. It then sends jobs to the worker solvers to ask for the insertion of a set of atomic activities within multiple CCAs in parallel. For each job, the set of atomic activities can be composed either by activities involved in a unique mode (so-called *Unit PCCAS*), or by activities involved in multiple modes (so-called *Batch PCCAS*). These two options lead to different algorithms defined in the two next sections.

6. Unit Parallel CCA Search (UPCCAS)

The first version of PCCAS is called *Unit Parallel CCA-based Search* (UPCCAS). In this approach, each worker solver attempts to schedule the activities of a single new mode in the CCA over which it works, and then returns the solution to the controller solver.

UPCCAS: overview The UPCCAS controller solver is composed of three main procedures (Figure 2). The first one, in the "*Initialization*" block, simply initializes data structures: for instance, it builds an empty sequence of activities for each CCA and generates the best mode associated with each request (one mode generation procedure per request type). After that, the "*Send jobs*" block consists in sending jobs to the worker cores, while there is still some job left and a free CPU. Then, the "*Analyze the results*" block reads the messages received from the worker cores and updates the current solution and the search state depending on the content of these messages. If time has not ran out and there are still some unplanned requests, the algorithm goes back to the "*Send jobs*" step. Last, when time is out, a message is sent to the worker cores to stop the search process. Further details on the different components of the algorithm are provided below.

UPCCAS: job definition For UPCCAS, the definition of the set of new modes to insert in a given CCA c is straightforward. This set is reduced to a singleton corresponding to the mode that has the highest reward among the request modes requiring activities in c.

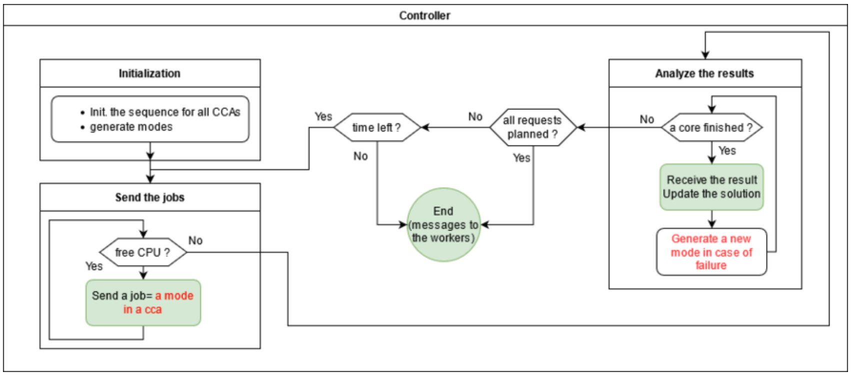
Controller
Initialization
• Init. the sequence for all CCAs
• generate modes
Send the jobs
free CPU ?
No
Yes
Send a job= a mode in a cca
time left ?
Yes
No
all requests planned ?
No
Yes
End (messages to the workers)
Analyze the results
a core finished ?
No
Yes
Receive the result Update the solution
Generate a new mode in case of failure

Figure 2. UPCCAS overview

UPCCAS: Send jobs At each step, the controller knows (1) the jobs being processed by the worker cores, (2) the jobs that have already been completed, and (3) the jobs that still need to be sent to a worker core. The "*Send jobs*" procedure consists in picking a job among the set of remaining ones and in sending it to a free worker core. Our goal is to insert modes into CCAs while respecting the greedy order defined by the mode rewards. Since modes are generated on-the-fly, future modes might appear in each CCA c , possibly interfering with the set of modes currently known for c.

To overcome this difficulty, we proceed as follows. First, a future mode for a request r might appear in a CCA c if some activities of r belongs to c and r is neither validated, nor rejected yet. Second, our mode generation process guarantees that for a given request r, the modes of r are generated by non-increasing reward (if inserting mode m fails for request r, then the next possible mode m' generated on-the-fly for r always satisfies $\omega_{m'} \leq \omega_m$). With these two points in mind, the algorithm is allowed to plan a mode m in c only if ω_m is higher than or equal to the reward of all current modes associated with requests that still have candidate activities in c, to be sure that m will never interfere with modes that have a higher reward.

UPCCAS: Analyze the results When the result of a job is analyzed by the controller, two cases can occur:

- Case 1: mode failure. If mode m of request r is rejected by a CCA required by m, then we cancel the mode in every CCA involving m (both for the CCAs where the insertion of m has already succeeded and for the CCAs still working on m). Then, if possible, we generate the next candidate mode for r, while taking in account the rejection explanation failures (the set of rejected activities) collected all along the search process. If there is no next possible mode for r, then r is rejected forever.
- Case 2: mode acceptance. If mode m of request r is accepted, then there is one less CCA for which an insertion validation is required. If there is no more validation waited for, the current mode of r is fully validated and r is completed. This might unlock some CCAs containing activities associated with request r.

UPCCAS: worker solver A job received by a worker solver is composed of a single mode m whose activities must be inserted in a given CCA c. This mode covers a set of activities A in c (possibility to have several activities in A, for instance if the corresponding request is stereoscopic). The worker solver then tries to add activities in A one by one to the current sequence of activities currently planned for c. When the insertion of one activity fails, the worker solver also tries to change the ordering of the activities already planned in c. If a feasible solution is found, it is sent to the controller, otherwise a failure message is returned. Note that adding one activity to the current plan of a CCA is actually equivalent to searching for a solution to a TSPTW (Traveling Salesman Problem with Time Windows), where the customers are the activities and the travel time between two customers corresponds to the transition time between two observations.

UPCCAS: optimizations In our implementation, we add optimizations to reuse some previous results. More specifically, when the mode of a request is changed from m to m', there might exist common observations associated with m and m'. If all observations associated with a given CCA c have already been validated for m and are still contained in m', then the corresponding solution is kept for c.

UPCCAS: example We illustrate UPCCAS on the example of Figure 1. We assume that for each request r_j ($j \in [1..3]$), we have $\omega_{m_i^j} > \omega_{m_{i'}^j}$ iff $i < i'$. Requests r_1, r_2, r_3 start with their best modes, namely $m_1^1 = \{o_1\}$, $m_1^2 = \{o_6, o_7\}$, and $m_1^3 = \{o_{10}, o_{11}\}$. Let us assume that m_1^1 is the mode that has the highest reward among m_1^1, m_1^2, m_1^3. All the CCAs containing observations for m_1^1 can be asked to insert the corresponding observations (here o_1 requested on CCA c_1). Then, we try to start scheduling activities for other CCAs while guaranteeing that the request reward order is always respected. For CCA c_2, it is not possible to start adding observations o_6 and o_7 belonging to m_1^2, since if the insertion of o_1 in c_1 fails, it might be the case that o_3 is part of a mode that has a better reward than $\omega_{m_1^2}$. For CCA c_3, no computation is triggered because no activity of c_3 belongs to the modes considered at this stage. For CCA c_4, observation o_{11} belonging to mode m_1^3 is not challenged by any another mode of another request, hence the insertion of o_{11} in c_4 can be requested to a worker solver.

We now assume that the insertion of o_1 in c_1 succeeds. In this case, mode m_1^1 is validated since all its observations have been successfully inserted. Consequently, all observations associated with request r_1 and that do not belong to m_1^1 can be deactivated. The deactivation of o_3 implies that request r_2 becomes the request with the highest reward in c_2 and the insertion of o_6 and o_7 in c_2 can now be requested to a worker solver.

In parallel, let us assume that the insertion of o_{11} succeeds. In this case, mode m_1^3 is not validated yet because o_{10} has not been scheduled. Suppose now that the insertion of o_6 and o_7 succeeds. Then, mode m_1^2 becomes validated and request r_3 now has the highest reward in c_2. As a result, the insertion of o_{10} in c_2 can be requested to a worker solver. If the insertion of o_{10} fails, we cancel mode m_1^3, *i.e.* we remove all observations of this mode that were planned before (o_{11} in the example), and we generate the next mode for r_3 taking into account the insertion failure for o_{10}. Here, the new mode generated is $m_2^3 = \{o_8, o_{11}\}$ and we try to fulfil this mode following the same process as before.

7. Batch Parallel CCA Search (BPCCAS)

In UPCCAS, modes are planned one by one in each CCA. This results in a lot of messages exchanged between the controller solver and the worker solvers, and the associated communication time is not negligible compared to the total computation time available. To overcome this issue, we introduce another approach called BPCCAS where larger jobs are sent to the worker solvers, by requesting to plan at each step a batch of modes within a given CCA instead of a single one.

BPCCAS: overview The BPCCAS controller procedure is composed of the same three blocks as UPCCAS. The first main difference is the connection between the blocks. Indeed, we iterate here and often go back to the *Initialization* step to simultaneously generate new modes for several requests. The definition of a job is also different here because we send batches of modes to each worker solver, and each worker plans activities in the order specified by the mode rewards, so that the worker somehow collaborate by inserting modes in the same order. Further details are given below.

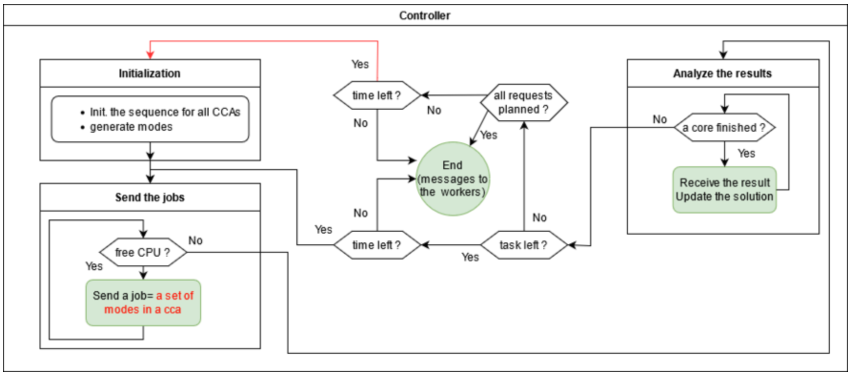

Figure 3. BPCCAS overview

BPCCAS: job definition For each CCA c, the goal is simply to schedule the batch of all modes m currently involved in c.

BPCCAS: Initialization The initialization procedure is mainly the same as in UPCCAS. The main difference is that it is called at the beginning of every iteration rather that once and for all, since in BPCCAS, we wait for the result of all jobs before generating new modes for the rejected requests. Note that at the beginning of each iteration, we start from an empty schedule because a new set of modes means a new greedy order.

BPCCAS: Send jobs Contrarily to UPCCAS, each job is a batch of modes.

BPCCAS: Analyze the results When a message is received from a worker solver concerning a CCA c, BPCCAS updates the set of rejected modes and records the observations whose insertion has failed. If all computations required over all CCAs are over and if some modes have been rejected, the set of candidate modes is updated. More precisely, for each request r whose current mode is m, three cases can occur after each iteration:

- mode acceptance: m is kept for the next iteration;
- mode failure: a new mode is generated for r for the next iteration;
- unknown status: one worker core has reached a certain amount of mode failures and did not try to schedule m. In this case, m is kept for the next iteration.

BPCCAS: worker solver The worker solver is mainly the same as in UPCCAS, and it basically solves TSPTWs to determine whether new activities can be planned in a CCA. The only difference is the definition of a job. Indeed, a set of modes are scheduled here, in the greedy order in terms of mode reward.

BPCCAS: optimizations In our implementation, to speed up the search process, we do not replan from scratch each time a single mode is rejected. More precisely, in each CCA c where a set of new modes M must be planned, we first determine the highest reward $\overline{\omega}$ associated with a mode in M. Then, we keep all modes m that were successfully inserted in c and such that $\omega_m \geq \overline{\omega}$, since these modes cannot be challenged by the new ones given the greedy nature of the mode insertion algorithm.

BPCCAS: example As for UPCCAS, we initially select the best mode for each request, that is $m_1^1 = \{o_1\}$, $m_1^2 = \{o_6, o_7\}$, and $m_1^3 = \{o_{10}, o_{11}\}$ for the example of Figure 1. Then, we simultaneously consider all the selected modes: in terms of atomic observation activities, we try to schedule $\{o_1\}$ in c_1, $\{o_6, o_7, o_{10}\}$ in c_2, nothing in c_3, and $\{o_{11}\}$ in c_4. Once all CCAs have been processed, we analyze the results to determine the requests for which some observations are rejected for their current mode. For every rejected requests, the next possible mode is generated, if any, and the whole scheduling process is triggered again. For instance, if the insertion of observation o_{10} fails (rejection of mode m_1^3 by c_2), a new mode $m_2^3 = \{o_8, o_{11}\}$ is generated. The sets of observations to schedule become $\{o_1, o_8\}$ in c_1, $\{o_6, o_7\}$ in c_2, nothing in c_3, and $\{o_{11}\}$ in c_4. As previously mentioned, previous results can be kept to avoid recomputing plans from scratch in each CCA.

8. Perturbation Process

In several situations, UPCCAS and BPCCAS terminate before the time limit. To exploit the full computation time available, once the first run is over, we randomize the greedy order and rerun the algorithm as many times as possible. To perform such an iterated greedy stochastic search, we generate, at the k^{th} run, a noise $\sigma_r^{(k)} \sim \mathcal{U}([-k\frac{\sigma_0}{2}, k\frac{\sigma_0}{2}])$ for each request r, with σ_0 a parameter of our method. Then, the reward of each mode m becomes $\omega_m^{(k)} = \omega_m + \sigma_r^{(k)}$ and we run the algorithm with the updated rewards.

9. Experiments

Benchmarks For the experiments, we consider a Walker constellation composed of 16 satellites (8 orbital planes and 2 satellites per plane) and a one-day long scheduling horizon. Request targets are randomly generated in Europe. We consider a unique ground station and book a download duration of 3 minutes within each download window.

Several instances composed of 50, 200, 500 and 1000 requests are generated. For each size, we generate several combinations of types of requests and each instance is denoted by "*m,s,y,p*" where m, s, y, p correspond to the number of monoscopic, stereoscopic, systematic, and periodic requests respectively. For each combination, we generate several instances with a different seed and calculate mean results. In accordance with the real catalogues of requests provided by our industrial partner for this study, there are less systematic and periodic requests than monoscopic and stereoscopic ones. Periodic requests all follow the same pattern: they are composed of 4 observations that should be performed around 8am, 12pm, 4pm, and 8pm. Each observation opportunity is given a duration and a score in $[0,1]$ depending on the cloud coverage forecast. We use historical weather data of January 2020 to get realistic observation scores. The reward of a mode corresponds to the sum of the scores of its observations. The observation durations follow a normal truncated distribution in range $[1,60]$ (in seconds) centered in 15 with a variance of 10. Long observation durations allow to capture videos of ground areas. The sizes of the connected components vary a lot. For one of the $0,1000,0,0$ instances, the components contain 40 activities on average, but the smallest and largest ones respectively contain 1 and 892 activities. For one of the $200,200,30,70$ instances, the component sizes range from 1 to 456 activities, with a mean equal to 29.

Experimental setup All the algorithms are implemented in Python 3.8.5. The Global Solver uses DOcplex 2.22.213. Two versions of this solver are tested, namely "GCPS 5 modes" and "GCPS 15 modes" that respectively take as an input the best 5 and 15 modes for each request. For UPCCAS and BPCCAS, the worker solvers use LKH3 to check whether a given set of activities can be scheduled in a CCA [19]. Experiments were run on a 20-core Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz, 62GB RAM, with a time out of 1 minute, 5 minutes, or 30 minutes per run. Each run is allowed to use 10 CPUs, and DOcplex automatically parallelizes search over the number of cores specified.

Comparisons Table 1 presents the results wrt the total rewards obtained for a representative set of instances, as well as results concerning mode generation. More precisely, for each request r, the solution produced uses mode number k_r for request r, and the table gives the maximum and mean values of k_r over all requests, averaged over the number of runs (columns *max* and *mn* respectively).

Table 1 first shows that for GCPS, the version using the 5 best modes is better than the version using the 15 best modes on average when the computation time equals 1 minute or 5 minutes. The main reason is that finding a good solution takes more time when the search space is larger.

Second, for the PCCAS algorithms, Table 1 shows that BPCCAS is much faster than UPCCAS, mainly because it directly handles batches of modes As shown in Figure 4, the usage of the worker cores on a single run (before the perturbation) is far better distributed

Time limit	Instance	#$\mathcal{A}$	GCPS 5 modes			GCPS 15 modes			UPCCAS			BPCCAS		
			rwd	mode		rwd	mode		rwd	mode		rwd	mode	
				max	mean		max	mean		max	mean		max	mean
	50,0,0,0	1347	**46.2**	0.0	0.0	**46.2**	7.0	1.0	**46.2**	0.0	0.0	**46.2**	0.0	0.0
	15,15,10,10	1110	**87.8**	4.0	0.8	**87.8**	11.4	1.3	**87.8**	0.0	0.0	**87.8**	0.0	0.0
	70,70,10,50	3505	**361.5**	4.0	0.9	361.0	13.0	1.9	356.8	2.4	0.2	355.6	3.0	0.2
	250,250,0,0	8499	512.5	4.0	1.7	475.0	14.0	3.9	497.0	18.0	1.4	**586.7**	20.0	1.4
1min	0,500,0,0	5702	445.6	4.0	1.7	448.0	7.0	2.0	462.5	6.2	0.8	**468.7**	7.0	0.7
	200,200,30,70	8544	548.5	4.0	1.7	507.0	14.0	4.1	530.2	21.2	1.8	**642.3**	20.6	1.5
	0,1000,0,0	11161	0.0	0.0	0.0	0.0	0.0	0.0	457.4	6.6	0.9	**507.8**	6.8	0.6
	400,400,60,140	16830	119.2	2.0	0.9	0.0	0.0	0.0	393.3	21.8	3.5	**710.8**	21.6	2.4
	440,440,40,80	16835	109.4	2.2	1.1	0.0	0.0	0.0	436.9	21.4	3.1	**745.4**	21.4	2.2
	70,70,10,50	3505	**365.0**	4.0	0.7	364.1	13.4	1.7	356.8	2.4	0.2	355.6	3.0	0.2
	250,250,0,0	8499	563.6	4.0	1.5	544.5	13.4	2.5	**598.9**	18.4	1.4	594.8	19.6	1.4
	0,500,0,0	5702	**517.3**	4.0	1.5	502.1	7.0	1.8	479.4	6.4	0.8	480.5	6.4	0.7
5min	200,200,30,70	8544	595.5	4.0	1.5	579.1	14.0	3.2	652.9	20.8	1.6	**653.9**	20.8	1.4
	0,1000,0,0	11161	**585.4**	4.0	1.8	554.7	6.8	2.0	528.3	6.8	0.9	519.7	6.8	0.6
	400,400,60,140	16830	760.1	4.0	1.8	627.9	14.0	5.2	683.0	21.8	3.5	**826.7**	21.6	2.4
	440,440,40,80	16835	766.1	4.0	1.8	666.4	14.0	5.3	750.5	21.4	3.1	**812.0**	21.4	2.2
	70,70,10,50	3505	**366.3**	4.0	0.7	365.9	13.4	1.5	356.8	2.4	0.2	355.6	3.0	0.2
	250,250,0,0	8499	578.1	4.0	1.4	578.3	13.2	2.1	**608.3**	18.8	1.5	595.4	20.0	1.4
	0,500,0,0	5702	**541.0**	4.0	1.4	533.6	7.0	1.6	485.5	6.6	0.8	481.2	6.6	0.7
30min	200,200,30,70	8544	611.4	4.0	1.4	625.7	14.0	2.5	**666.5**	21.6	1.8	656.4	20.6	1.5
	0,1000,0,0	11161	**662.5**	4.0	1.6	654.2	6.8	1.7	536.0	6.6	0.8	526.0	7.0	0.6
	400,400,60,140	16830	821.4	4.0	1.7	819.2	14.0	4.3	**834.0**	21.0	2.7	828.7	21.4	2.4
	440,440,40,80	16835	822.2	4.0	1.7	838.5	14.0	4.4	**840.1**	21.0	2.7	820.9	21.2	2.3

Table 1. Total reward, maximum and mean modes for the selected requests, for several instances

in BPCCAS. Nevertheless, Table 1 shows that UPCCAS catches up BPCCAS after 30 minutes. Theoretically speaking, UPCCAS and BPCCAS should reach the same solutions during the whole search process, so they should converge to the same best rewards. However, our implementation still involves some sources of non-determinism, e.g. on the way the rejection of activities are used to generate new modes on-the-fly, or on the way modes having the same reward are handled. This can explain the differences observed.

Last, when comparing GCPS with the PCCAS algorithms, we can see that all the approaches are equivalent on small instances (50 requests), where all the requests are satisfied with their best mode. On large instances (200 requests and more), the comparison depends on the time limit. After 1 minute, BPCCAS provides the best results. Although the difference tends to decrease while the time elapsed, it remains significant on specific very large instances (see $400, 400, 60, 140$ after 5 minutes) or on instances where the number of modes generated is high (see instances $250, 250, 0, 0$ and $200, 200, 30, 70$ after 30 minutes). In this case, GCPS deals with a limited number of modes whereas UPCCAS and BPCCAS are able to generate new modes on-the-fly. Therefore, UPCCAS and BPCCAS do not explore the same search space as GCPS. Additionally, GCPS seems to be slow at finding a first good-quality solution on the very large instances, but it provides the best results on several instances. Even if GCPS will not be able to handle the time-dependent aspects in our future works, the solutions found can still be used to test the efficiency of UPCCAS and BPCCAS on time-independent problems.

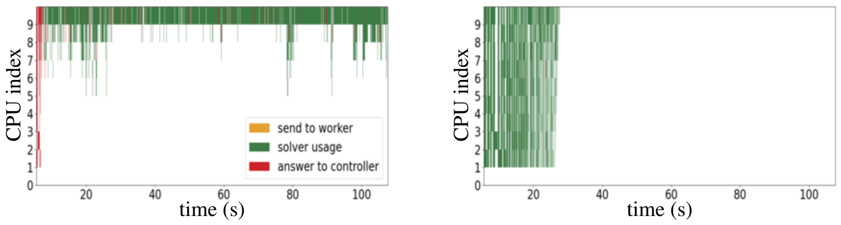

Figure 4. CPU usage on a single run of UPCCAS (left) and BPCCAS (right) on instance $200, 200, 30, 70$

10. Conclusion and perspectives

In this paper, we considered an application related to the management of complex user requests for a constellation of Earth observation satellites. From an AI technology point of view, we tested both on-the-shelf CP methods (GCPS approach) and ad-hoc constructive search procedures exploiting problem decompositions and parallelization (UPCCAS and BPCCAS). Both approaches have their strengths and weaknesses. To combine the best of the two worlds, the next step is to look for hybrid techniques combining on one side a constructive search procedure acceptable from an operational point of view, and on the other side AI combinatorial optimization methods. To get such an hybrid search scheme, Large Neighborhood Search would be a very good candidate. Overall, the long term objective for us is to show that it is possible to go beyond the simple requests managed by many existing satellite systems and bring new functionalities to the end-users.

Acknowledgement This work has been performed with the support of the French government, in the context of the BPI PSPC project "LiChIE" of the "Programme d'Investissements d'Avenir".

References

[1] Han C, Wang X, Song G, Leus R. Scheduling multiple agile Earth observation satellites with multiple observations. arXiv:181200203. 2018.

[2] Hu X, Zhu W, An B, Jin P, Xia W. A branch and price algorithm for EOS constellation imaging and downloading integrated scheduling problem. Computers & Operations Research. 2019;104:74-89.

[3] Kim J, Cho DH, Ahn J, Choi HL. Task scheduling of multiple agile satellites with transition time and stereo imaging constraints. arXiv:191200374. 2019.

[4] Xiao Y, Zhang S, Yang P, You M, Huang J. A two-stage flow-shop scheme for the multi-satellite observation and data-downlink scheduling problem considering weather uncertainties. Reliability Engineering & System Safety. 2019;188:263-75.

[5] Bianchessi N, Cordeau JF, Desrosiers J, Laporte G, Raymond V. A heuristic for the multi-satellite, multi-orbit and multi-user management of Earth observation satellites. European Journal of Operational Research. 2007;177(2):750-62.

[6] Wang J, Zhu X, Yang LT, Zhu J, Ma M. Towards dynamic real-time scheduling for multiple Earth observation satellites. Journal of Computer and System Sciences. 2015;81(1):110-24.

[7] Eddy D, Kochenderfer MJ. A maximum independent set method for scheduling Earth observing satellite constellations. arXiv:200808446. 2020.

[8] Holvoet N, Vongsantivanich W, Chaimatanan S, Delahaye D. Mission planning for non-homogeneous Earth observation satellites constellation for disaster response. In: 15th International Conference on Space Operations; 2018. p. 1-15.

[9] Sun K, Li J, Chen Y, He R. Multi-objective mission planning problem of agile Earth observing satellites. In: 12th International Conference on Space Operations; 2012. p. 2802-10.

[10] Li Z, Li X. A multi-objective binary-encoding differential evolution algorithm for proactive scheduling of agile Earth observation satellites. Advances in Space Research. 2019;63(10):3258-69.

[11] Dishan Q, Chuan H, Jin L, Manhao M. A dynamic scheduling method of Earth-observing satellites by employing rolling horizon strategy. The Scientific World Journal. 2013;3.

[12] He L, Liu X, Laporte G, Chen Y, Chen Y. An improved adaptive large neighborhood search algorithm for multiple agile satellites scheduling. Computers & Operations Research. 2018;100:12-25.

[13] Du Y, Wang T, Xin B, Wang L, Chen Y, Xing L. A data-driven parallel scheduling approach for multiple agile Earth observation satellites. IEEE Transactions on Evolutionary Computation. 2020;24:679-93.

[14] He Y, Chen Y, Lu J, Chen C, Wu G. Scheduling multiple agile Earth observation satellites with an edge computing framework and a constructive heuristic algorithm. Journal of Systems Architecture. 2019;95:55-66.

[15] Levinson R, Nag S, Ravindra V. Agile satellite planning for multi-payload observations for Earth science. In: International Workshop on Planning and Scheduling for Space; 2021. p. 89-97.

[16] Squillaci S, Roussel S, Pralet C. Managing complex requests for a constellation of Earth observing satellites. In: International Workshop on Planning and Scheduling for Space; 2021. p. 150-8.

[17] Picard G. Auction-based and distributed optimization approaches for scheduling observations in satellite constellations with exclusive orbit portions. In: International Workshop on Planning and Scheduling for Space; 2021. p. 108-16.

[18] Aguiar-Melgarejo P, Laborie P, Solnon C. A time-dependent no-overlap constraint: application to delivery problems. In: 12th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research; 2015. p. 1-17.

[19] Helsgaun K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research. 2000;126(1):106-30.