Integer and Constraint Programming for the Offline Nanosatellite Partition Scheduling Problem

Julien Rouzot, Mickaël Pereira, Christian Artigues, Romain Boyer, Frédéric Camps, Philippe Garnier, Emmanuel Hebrard, Pierre Lopez

▶ **To cite this version:**

Julien Rouzot, Mickaël Pereira, Christian Artigues, Romain Boyer, Frédéric Camps, et al.. Integer and Constraint Programming for the Offline Nanosatellite Partition Scheduling Problem. 22th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2025), Melbourne (Australia), November 10-13, 2025., Nov 2025, Melbourne, Australia. ⟨hal-05058427⟩

HAL Id: hal-05058427

https://laas.hal.science/hal-05058427v1

Submitted on 15 May 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons CC0 1.0 - Universal - International License

Integer and Constraint Programming for the Offline Nanosatellite Partition Scheduling Problem

Julien Rouzot[1,2], Mickaël Pereira[1], Christian Artigues[1], Romain Boyer[1], Frédéric Camps[1], Philippe Garnier[1,2], Emmanuel Hebrard[1], and Pierre Lopez[1]

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
firstname.lastname@laas.fr
[2] IRAP, Université de Toulouse, CNRS, UT3, CNES, Toulouse, France
firstname.lastname@irap.omp.eu

Abstract. Effective scheduling of tasks for nanosatellites is essential, given their limited onboard resources and capabilities. In a typical nanosatellite mission, the onboard computer has to run payload tasks linked to the mission objective, such as observations and measurements for science campaigns, but also communication and avionic tasks needed for navigation and safety. We consider a partitioned real-time system where a partition is a job that embeds a set of elementary tasks implementing one of the above-described functions. The offline nanosatellite partition scheduling problem consists in scheduling a set of partitions on a single-core processor within a fixed time frame that will be repeated cyclically, while maximizing the number and duration of scheduled payload partitions. The paper first establishes similarities and differences with related scheduling problems. We then prove that the problem is strongly NP-hard. A Mixed Integer Linear Programming (MILP) matheuristic and a Constraint Programming (CP) model are proposed. We compare the MILP and CP approaches in a multi-objective context and demonstrate the relevance of the latter to solve the problem efficiently both on randomly generated instances and on a real nanosatellite case study of the University Space Center of Toulouse: the NIMPH project. The CP model is embedded in NanoSatScheduler, an open source software with a user interface designed for the offline nanosatellite partition scheduling problem. For the NIMPH real-case study, the proposed solution outperforms the semi-manual approach used so far. As a result, the NIMPH team has adopted NanoSatScheduler as an operational tool for their mission.

Keywords: Constraint Programming · Mixed Integer Linear Programming · Nanosatellites · Offline Partition Scheduling · Periodic Scheduling

1 Introduction

In scientific nanosatellite missions, the onboard computer has to orchestrate scientific and avionic tasks related to the mission objective and the proper functioning of the nanosatellite. We consider the case where isolated threads, also

called partitions, are run on a single-core processor. Each partition encapsulates a segment of the application code. The principle of a partitioned system ensures space and time isolation of applications with different levels of criticality [27]. Typically, some partitions correspond to payload tasks with low criticality but high interest for the mission, while other partitions correspond to avionic tasks with high criticality for nanosatellite safety. In this paper, we introduce the offline nanosatellite partition scheduling problem (ONPSP) and we refer to the partitions as *jobs* for a better match with the scheduling literature. These jobs are decomposed into elementary tasks of variable duration. The schedule is built on a short time frame called the cycle time (e.g. 1 second) and is repeated indefinitely until further notice. There are various intra- and inter-job constraints that complicate the scheduling process. The time lag between two executions of tasks of the same job can be constrained to have a maximum value. The latter constraint is used to ensure the regularity of critical tasks within the time frame. There are also minimum time-lag constraints within the tasks of the same job, which represent a minimum amount of time required between two executions of a payload task, that can be needed to save data, resetting an instrument between two observations or just to spread the scientific observations over time. There are also precedence constraints between tasks of different jobs, to represent a data flow between tasks. Finally, some jobs may have a set of fixed times at which a task must start, allowing the user to build upon an existing incomplete plan.

Although the models proposed in this paper are valid for many onboard partitioned systems, our work takes place in a practical study, the Nanosatellite to Investigate Microwave Photonics Hardware (NIMPH) mission, to be launched in 2025 by the University Space Center of Toulouse (Centre Spatial Universitaire de Toulouse, CSUT)[3]. This mission is an academic endeavor that is part of the larger NanoLab Academy project launched by the French National Center for Space Studies (Centre National d'Etudes Spatiales, CNES) in an effort to bring students to develop and exploit their own spacecraft to aid in novel scientific experiments. The NIMPH main mission is designed to prove the viability of microwave photonics hardware, i.e. a new kind of radio frequency transmitter that merges fiber optic and photonic devices, in space. The NIMPH nanosatellite is based on the CubeSat architecture [10] and the tasks to be performed must be scheduled on a single-core processor through the hypervisor XTratuM [15], used for all NanoLab Academy missions.

Several objectives can be considered depending on the mission priorities. In [21], we proposed a method to solve a variant of the problem aimed at minimizing the context switches [6], i.e. the number of times the processor switches from one job to another. The number of tasks and their durations were considered as fixed parameters for each job, the fixed start times were not considered and the problem complexity was left open. With the evolution of the real project, handling fixed start times and maximizing the number and duration of science

[3] https://www.csut.cnrs.fr/en/project/nanosatellite-to-investigate-microwave-photonics-hardware/

jobs have become strict necessities, while context switches became of secondary importance and are thus ignored in this study.

We propose a MILP-based matheuristic that adapts the MILP proposed in [18] for a related nanosatellite scheduling problem and a CP model. Both aim at generating an optimized schedule based on user-defined multiple objectives, considering the number of task occurrences and durations across different partitions. The CP model is integrated into an open-source software called NanoSatScheduler that includes a user interface where the user can create/modify instances and review/modify the solutions provided by the solver.

The remainder of the paper is organized as follows. In Section 2, we define the problem and its objectives. In Section 3, the related work in the literature is briefly presented. In particular, the similarities and differences with the previously considered offline nanosatellite task scheduling problem, and other (partition) scheduling problems are discussed. Section 4 establishes that the ONPSP is strongly NP-hard. The MILP model and the matheuristic are detailed in Section 5. The CP model and the user interface embedded in NanoSatScheduler are presented in Section 6. Section 7 provides computational experiments. The two approaches are compared on randomly generated instances and on the real-case study. Conclusions are drawn in Section 8.

2 Problem Definition

In the ONPSP under consideration, each job $i \in \mathcal{J} = \{i = 1, \ldots, n\}$ consists of elementary tasks. For each job, there is a minimum number of tasks $\underline{m}_i$ and a maximum number of tasks $\overline{m}_i$ such that the actual number of scheduled tasks $m_i \in [\underline{m}_i, \overline{m}_i]$ is a decision variable. Set $\mathcal{M}_i = \{1, \ldots, \overline{m}_i\}$, denotes the indices of all potential tasks of job $i \in \mathcal{J}$ and once m_i is chosen, the scheduled tasks are assumed to have indices $\{1, \ldots, m_i\}$. All scheduled tasks must not overlap, as they run on a single-core processor, and within the time frame $[0, h]$, where h is the cycle time. Each task in each job i has a minimum duration $\underline{p}_i$ and a maximum duration $\overline{p}_i$ and the actual duration $p_{i,j} \in [\underline{p}_i, \overline{p}_i]$ of a scheduled task j of job i is also a decision variable . The start time of each scheduled task and the start time of its immediate successor within the job, if it is scheduled, must be separated by at least lag_i^{min} time units (minimum time lag). Additionally, the end time of each scheduled task and the start time of its successor, if it is scheduled, must be separated by no more than lag_i^{max} time units (maximum time lag)[4]. Due to periodicity, the time-lag constraints have also to be enforced between the last scheduled task of the job and the first scheduled task of the job. More precisely, if the first task of job i starts at time t, its last task starts at time t'_{start} and ends at t'_{end}, then $h - t'_{start} + t \geq lag_i^{min}$ and $h - t'_{end} + t \leq lag_i^{max}$. For each job i, we have a set of Q_i fixed start times $T_i = \{t_{i,1}, \ldots, t_{i,Q_i}\}$. There must be exactly one task of job i scheduled to start at each of the fixed start times in T_i. Moreover, there is a set E of precedence constraints between different jobs.

[4] Modeling start-to-start and end-to-start time lags is a request from the NIMPH team, but this can be easily modified for other nanosatellite missions.

A precedence constraint $(i, i') \in E$ states that if task (i', j) is scheduled, then task (i, j) should be scheduled before it in the time frame.

Depending on the mission contexts, the effort of one or more jobs must be optimized, either by maximizing the number of tasks or the overall duration of the corresponding job. Those multiple – and conflicting – objectives are user defined. The weights u_i and v_i respectively define the importance of maximizing the number of tasks and total duration for each job, thus allowing users to finely tune their objective. The objective function of the ONPSP is given by (1). We provide a solution for an ONPSP instance as an illustrative example of the problem in Figure 1.

$$\sum_{i=1}^{n} \sum_{j=1}^{m_i} u_i + v_i p_{i,j} \tag{1}$$

Fig. 1. In this toy example, we have 3 jobs (A,B,C), with $\underline{m}_A = 1$, $\overline{m}_A = 3$; $\underline{m}_B = \overline{m}_B = 1$; $\underline{m}_C = \overline{m}_C = 2$. Job A is the only one with variable duration, $\underline{p}_A = 2$, $\overline{p}_A = 4$. The only precedence constraint is (A, B), so at least k tasks of job A must occur before the k-th task of job B. We have $lag_A^{min} = 5$ and $lag_C^{max} = 5$. $T_B = \{2\}$, so the only task of job B must start at $t = 2$. In this example, the objective is to maximize the number of tasks for job A first ($u_A = M$, with M a large number), then we want to maximize the total duration for A ($v_A = 1$). For the other jobs $i \neq A$, we have $u_i = v_i = 0$.

3 Related Works

The literature on scheduling for nanosatellites is divided between online (onboard) and offline scheduling methods, such as in our study.

Although Rigo et al. [19] claim that online scheduling is not adapted to CubeSat nanosatellites due to their limited onboard capacity and the need to get close to optimal solutions, online scheduling approaches have recently been investigated, in particular to evaluate online task scheduling strategies aimed at minimizing power consumption [3, 7, 12–14, 24, 25, 28].

The literature on nanosatellite offline scheduling [4, 18–20, 23] focuses on mathematical programming approaches for the offline nanosatellite task scheduling problem (ONTSP). The ONTSP considers in the same model all kinds of tasks performed by the nanosatellite: payload, communication, or operating system tasks. The goal is to schedule them on a much larger time horizon than in our problem, corresponding to at least an orbit (about 100 min) with a time discretization of 1 min or 30 sec. A task must be scheduled within a time window and can be started and stopped as required, but with a minimum and maximum number of startups until the required duration reaches a value in a predefined interval and the tasks can be performed in parallel. A limitation comes from energy requirements, and the required power cannot exceed a given threshold to

save energy. The ONTSP can be viewed as a coarse-grained scheduling problem for the satellite mission, while the ONPSP is a fine-grained scheduling problem that can be seen as a lower-level problem whose solution is needed to implement the high-level plan defined by the ONTSP solution.

The ONPSP is close to other offline (partition) scheduling problems in real-time systems encountered in the literature. However, many of the proposed methods, including integer and constraint programming, deal with strictly periodic scheduling on multi-core processors [1, 2, 5, 11, 16, 22]. The ONPSP involves a single-core processor and minimum/maximum time lags offer more flexibility than strict periodicity.

In the general scheduling literature, there are related single-machine problems, especially the single-machine problem with minimum and maximum time lags that is NP-hard in the general case. In [26], several complexity results have been established, but our problem appears as a special case due to the specific time lag between the last and first scheduled tasks of each job, as well as the presence of fixed start times. Similarly, our problem is a special case of NP-hard cyclic scheduling problems [9], since there is no overlap of task instances between two time frames. A specific study is therefore needed to formally establish the complexity of the problem.

4 Problem Complexity

We establish the complexity of the ONPSP when the number of tasks for each job i is fixed ($m_i = \underline{m}_i = \overline{m}_i$) and the duration of each task of each job i is also fixed ($p_{i,j} = \underline{p}_i = \overline{p}_i$), the weight of the job in terms of number of scheduled tasks is $u_i = 1$ and its weight concerning its total scheduled duration is $v_i = 0$. There are no precedence constraints ($E = \emptyset$). The considered decision variant of the problem asks whether we can find a schedule with the objective $\sum_{i=1}^{n} m_i$, i.e. whether we can schedule all tasks in the cycle time h.

The NP-hardness proof of the ONPSP follows from straightforward reductions from 3-Partition (see Figures 2 and 3). The 3-Partition problem asks whether a set of $3m$ values $\{e_k \mid k = 1, \ldots, 3m\}$ of total value $\sum_{k=1}^{3m} e_k = mQ$ with $m \geq 1$ integer and $\frac{Q}{4} < e_k < \frac{Q}{2}$ can be partitioned into a set of m triplets, each of total value Q. The following theorems show that the problem difficulty comes from the fixed start times and the minimum/maximum time lags, even considered separately.

Theorem 1. *The ONPSP is strongly NP-hard, even if* $m_i = 1$, $lag_i^{min} = 0$ *and* $lag_i^{max} \geq h$, $\forall i = 1, \ldots, n$.

Proof. Consider the 3-Partition problem from which we build an instance of the ONPSP as follows. Let $h = m(Q + 1) - 1$. We have $3m + 1$ jobs where for $i = 1, \ldots, 3m$, $m_i = 1$, $lag_i^{min} = 0$, $lag_i^{max} = h$, $p_i = e_i$ and for $i = 3m + 1$, $m_i = m - 1$, $lag_i^{min} = 0$, $lag_i^{max} = h$, $p_i = 1$. Job $3m + 1$ has a set of $m - 1$ fixed start times $t_{3m+1,k} = \{(Q + 1)k - 1\}$ for $k = 1, \ldots, m - 1$. The fixed tasks of job $3m+1$ leave only m idle intervals, each of length Q, on which the unfixed

jobs must fit exactly since $\forall i = 1, \ldots, 3m$, $p_i > \frac{Q}{4}$, it is not possible to schedule 4 jobs in a single interval. If strictly less than three jobs are scheduled within an interval, then an idle time must be present since $\forall i = 1, \ldots, 3m$, $p_i < \frac{Q}{2}$, which does not allow to schedule all tasks in the m intervals. Hence, the 3-Partition problem built this way has a solution if and only if the ONPSP has a solution. □

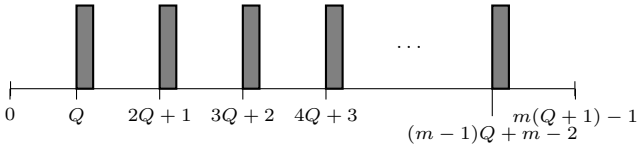

Fig. 2. Illustration of the reduction of 3-Partition to the ONPSP without lag. Grey rectangles represent the $m-1$ tasks with fixed start times.

Theorem 2. *The ONPSP is strongly NP-hard, even if* $T_i = \emptyset$, $\forall i = 1, \ldots, n$.

Proof. Consider the 3-Partition problem from which we build an instance of the ONPSP as follows. Let $h = m(Q+1)$. We have $3m+1$ jobs where, for $i = 1, \ldots, 3m$, $m_i = 1$, $lag_i^{min} = 0$, $lag_i^{max} = h$, $p_i = e_i$ and for $i = 3m+1$, $m_i = m$, $lag_i^{min} = lag_i^{max} = Q+1$, $p_i = 1$. It is easy to show that if the problem is feasible, there is a solution that schedules the first task of job $3m+1$ at time Q and the last job at time $h-1$. The m tasks of job $3m+1$ partition the cycle time in m idle time intervals of length Q where the other jobs must fit. The 3-Partition problem built this way has a solution if and only if the ONPSP has a solution. □

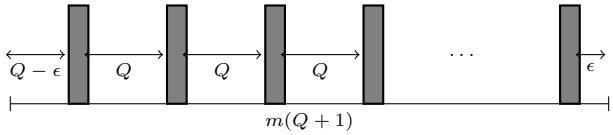

Fig. 3. Illustration of the reduction of 3-Partition to the ONPSP without fixed start times. Grey rectangles represent the m tasks with constant lag.

Without fix start times or time lags, the problem becomes polynomial if we only consider maximizing the number of tasks without precedence constraints. If we consider duration maximization, we can reduce the problem to Subset-Sum with or without precedence constraints (we need to select tasks with duration a_i such that $\sum a_i = C$, C being the time remaining when all mandatory tasks are inserted). The only case that remains is when we only want to maximize the number of tasks with precedence constraints, which corresponds in the 3-field scheduling notation to $1|d_i = d, prec|\sum U_i$. We are not aware of complexity results in this case.

5 MILP Model and Scaling Matheuristic

5.1 Time indexed MILP

We propose a MILP of the problem directly transposed from the one proposed in [18] for the related offline nanosatellite task scheduling problem, incorporating a new modeling framework for the precedence constraints that are absent in [18].

Let $y_{i,t}$ be a binary decision variable that states whether job i is in process at time period t and let $z_{i,t}$ denote a binary decision variable indicating whether a task of job i starts at time t. For ease of notation, we denote as $\rho(t)$ the time period immediately following t in a cyclic schedule, i.e. $\rho(t) = t + 1$ if $t < h - 1$ and $\rho(h - 1) = 0$ otherwise. Symmetrically, we denote as $\rho^{-1}(t)$ the time period immediately preceding t in a cyclic schedule. We have $\rho^{-1}(t) = t - 1$ if $t \geq 1$ and $\rho^{-1}(0) = h - 1$. We denote as $I(t, \delta)$ the cyclic interval of length δ starting at time period t that can be defined by $I(t, \delta) = \{\tau_1, \ldots, \tau_\delta\}$ with $\tau_1 = t$ and $\tau_q = \rho(\tau_{q-1})$ for $q = 2, \ldots, \delta$. Our MILP model is described below:

$$\max \sum_{i=1}^{n} \sum_{t=0}^{h-1} u_i z_{i,t} + v_i y_{i,t} \tag{2}$$

subject to:

$$z_{i,t} \geq y_{i,t} - y_{i,\rho^{-1}(t)} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{3}$$

$$z_{i,t} \leq y_{i,t} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{4}$$

$$\sum_{\tau=t}^{t+\underline{p}_i-1} y_{i,\tau} \geq \underline{p}_i z_{i,t} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-\underline{p}_i \tag{5}$$

$$\sum_{\tau=t}^{t+\underline{p}_i-1} z_{i,\tau} \leq 1 \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-\underline{p}_i \tag{6}$$

$$\sum_{\tau=t}^{t+\overline{p}_i} y_{i,\tau} \leq \overline{p}_i + \sum_{\tau=t+1}^{t+\overline{p}_i} z_{i,\tau} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-\overline{p}_i-1 \tag{7}$$

$$\underline{m}_i \leq \sum_{t=0}^{h-1} z_{i,t} \leq \overline{m}_i \qquad \forall i \in \mathcal{J} \tag{8}$$

$$\sum_{\tau \in I(t, lag_i^{min})} z_{i,\tau} \leq 1 \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{9}$$

$$\sum_{\tau \in I(t, lag_i^{max})} y_{i,\tau} \geq 1 \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{10}$$

$$\sum_{i \in \mathcal{J}} y_{i,t} \leq 1 \qquad \forall t = 0, \ldots, h-1 \tag{11}$$

$$\pi_{i,i',0} = z_{i,0} - z_{i',0} \qquad \forall (i, i') \in E \tag{12}$$

$$\pi_{i,i',t} = \pi_{i,i',t-1} + z_{i,t} - z_{i',t} \qquad \forall (i, i') \in E, \forall t = 1, \ldots, h-1 \tag{13}$$

$$z_{i,t} = 1 \qquad \forall i \in \mathcal{J}, \forall t \in T_i \tag{14}$$

$$\pi_{i,i',t} \geq 0 \qquad \forall (i,i') \in E, \forall t = 0, \ldots, h-1 \tag{15}$$

$$z_{i,t} \in \{0,1\} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h - \underline{p}_i \tag{16}$$

$$y_{i,i'} \in \{0,1\} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{17}$$

The objective (2) is to maximize the weighted sum of the number of tasks and the total duration of each job. Constraints (3) and (4) link variables $z_{i,t}$ and variables $y_{i,t}$, in the sense that $z_{i,t}$ marks the start of a task of job i and must be equal to 1 if the job is in process at time t but not at time $\rho^{-1}(t)$, the previous time period in the cyclic sense. Constraints (5) and (6) enforce the minimum duration for each task of each job. Constraint (5) states that if a job i is (re)started at time t, the task must be in process during at least $\underline{p}_i$ time periods and constraints (6) prevent a job from starting twice during an interval of length $\underline{p}_i$. Constraints (7) prevent a job from being scheduled more than $\overline{p}_i$ time periods within each interval of length $\overline{p}_i + 1$ unless it is restarted within this interval. Constraints (8) set the number of scheduled tasks of each job within the required bounds. Constraints (9) and (10) enforce the minimum and maximum time lags taking account of the cyclic context. Constraints (11) prevent tasks from being scheduled in parallel. Constraints (12-13) are discussed in more detail below. Constraints (14) are the fixed start time constraints. Constraints (15–17) give variable domains (the variable $\pi_{i,i',t}$ is also introduced below).

The MILP model proposed in [18] does not allow to tackle the precedence constraints. Modeling the precedence constraint between task (i,j) and task (i',j) in standard time-indexed MILP would require to index the "start" and "in-progress" variable by the task index j, such that constraint $\sum_{\tau=t}^{h-1} y_{i,j,t} + \sum_{\tau=0}^{t} z_{i',j,t}$, $\forall t = 0, \ldots, h-1$ models the precedence relation. However, this would create a much bigger MILP model with $2h\sum_{i=1}^{n} \overline{m}_i$ binary variables instead of $2hn$ binary variables.To overcome this difficulty we propose a new way of modeling the precedence constraints without increasing the number of binary variables, by introducing a continuous decision variable $\pi_{i,i',t}$ for each precedence constraint $(i,i') \in E$ and each time period t.

Theorem 3. *Constraints (12-13) enforce the precedence relation* $(i,i') \in E$.

Proof. Variable $\pi_{i,i',t}$ is incremented as soon as a task of job i starts and is decremented when a task of job i' starts. Thanks to the non-negativity of the variables, a task of job i' can only start if a task of job i' without an already scheduled successor has started. □

5.2 Scaling Matheuristic

Unfortunately, for practical applications, even with this reduced model, the time-indexed MILP formulation may have a huge number of binary variables, particularly in our use case, where the cycle time lasts a second while task accuracy is of the order of a microsecond. To address this issue, we propose a scaling matheuristic that works as presented in Algorithm 1. Steps 1–3 downscales and rounds the

data using a scale factor S to obtain an ONPSP aiming at being more restrictive than the original one. However, for fixed start times, no rounding choice is a priori more restrictive. We apply the following procedure, determined experimentally, to favor feasibility. We arbitrarily round up each fixed start time (step 3), but we prevent any task to be scheduled in the previous time period, except if the earliest completion time of another fixed task is precisely this time period (Constraint added at step 4). The resulting MILP may be infeasible while the original problem is feasible, which is a clear drawback of this method, prone to further improvements. In the case of success in solving the downscaled problem, we build and solve an LP (step 8) to obtain the start times and durations in the original scale. The tasks are ordered in the order obtained by the downscaled MILP (constraint 18) and all constraints are linear. Note, however, that this LP still may have no solution if the decisions taken by the downscaled MILP are incompatible with the original scale.

Algorithm 1: Scaling Matheuristic

Data: an ONPSP instance P, a scaling factor $S > 0$
Result: an ONPSP solution $(m_i)_{i\in\mathcal{J}}, (s_{i,j}, p_{i,j})_{i\in\mathcal{J}, j=1,\ldots;m_i}$ or `fail`
`// Build a downscaled ONPSP instance` (P')
1 $h' \leftarrow \left\lfloor \frac{h}{S} \right\rfloor$;
2 $\underline{p}'_i \leftarrow \left\lceil \frac{\underline{p}_i}{S} \right\rceil$, $\overline{p}'_i \leftarrow \left\lfloor \frac{\overline{p}_i}{S} \right\rfloor$, $lag_i^{max\prime} \leftarrow \left\lceil \frac{lag_i^{min}}{S} \right\rceil$, $lag_i^{min\prime} \leftarrow \left\lfloor \frac{lag_i^{max}}{S} \right\rfloor$, $\forall i \in \mathcal{J}$;
3 $T'_i = \{t'_{i,k} = \left\lceil \frac{t_{i,k}}{S} \right\rceil \, | i \in \mathcal{J}, k = 1, \ldots, Q_i\}$;
4 add to (P') additional constraints:
$y_{j,t} = 0, \forall j \in \mathcal{J}, \forall t \in \cup_{i\in\mathcal{J}}\{\rho^{-1}(t') | t' \in T'_i\} \setminus \cup_{i\in\mathcal{J}, t'\in T'_i} I(t', \underline{p}_i)$;
`// Solve` (P') `and build solution to` (P)
5 solve (P');
6 **if** `success` **then**
7 Get number of tasks m_i, start and duration $s'_{i,j}$, $p'_{i,j}$, $\forall i \in \mathcal{J}, j = 1, \ldots, m_i$;
8 Solve LP with variables $s_{i,j} \in [0, h]$, $p_{i,j} \in [\underline{p}_i, \overline{p}_i], \forall i \in \mathcal{J}, j = 1, \ldots, m_i$:

$$\max \sum_{i\in\mathcal{J}} \sum_{j=1}^{m_i} v_i p_{i,j} \tag{18}$$

$$s_{a,b} \geq s_{i,j} + p_{i,j} \qquad \forall a, b, i, j : s'_{a,b} > s'_{i,j} \tag{19}$$

$$s_{i,j} = t_{i,k} \qquad \forall i, j, i, k : s'_{i,j} = t'_{i,k} \tag{20}$$

$$s_{i,j+1} - s_{i,j} - p_{i,j} \leq lag_i^{max} \qquad i \in \mathcal{J}, j = 1, \ldots, m_i - 1 \tag{21}$$

$$h + s_{i,0} - s_{i,m_i} - p_{i,m_i} \leq lag_i^{max} \qquad i \in \mathcal{J} \tag{22}$$

$$s_{i,j+1} - s_{i,j} \geq lag_i^{min} \qquad i \in \mathcal{J}, j = 1, \ldots, m_i - 1 \tag{23}$$

$$h + s_{i,0} - s_{i,m_i} \geq lag_i^{min} \qquad i \in \mathcal{J} \tag{24}$$

if `success` **then**
9 return the solution found
10 return `fail;`

6 NanoSatScheduler

In this section, we present NanoSatScheduler, an open source software tailored to solve ONPSP. This software embeds a CP model and a user interface allowing the user to create/modify their ONPSP instances and visualize/edit/save the solutions provided by the solver.[5]

The variables $s_{i,j} \in [0, h - \underline{p}_i]$ and $e_{i,j} \in [\underline{p}_i, h]$ indicate the start and end time of the j-th task of job i, respectively, and the variables $p_{i,j} \in [\underline{p}_i, \overline{p}_i]$ represent their duration. The variables $x_{i,j} \in [0, 1]$ indicate whether the j-th task of job i is actually present in the schedule. Finally, the variables $f_{i,k} \in [1, \overline{m}_i]$ will allow us to select which task will be associated with each date fixed by the user. Our objective is a combination of the occurrences and the total time used in the schedule for each partition (25).

Constraints (26) order the start variables in increasing chronological order. Constraints (27) link the presence variables so that the first variables are always used to indicate the presence of a task in the schedule. Those two constraints are both helpful to model the problem, but also to prevent the solver from exploring symmetrical solutions, thus improving its efficiency. We enforce the minimum number of tasks with constraints (28). The global constraints NoOverlap (29) prevent the *present* tasks – task (i, j) is present if and only if $x_{i,j} = 1$ – from overlapping. This global constraint is available in most CP solvers. The respect of *max-lag* is ensured with constraints (30), which enforce a maximum delay lag_i^{max} between the end and start of two consecutive present tasks. The consistency between the last present task and the first task is ensured with constraint (31). In the same fashion, the *min-lag* constraints are respected thanks to constraints (32-33). For each precedence between partitions in E, a *predecessor* task is required to be present and placed before a *successor* task if it is present. Finally, the global constraint Element$(x, vars, value)$ allows us to enforce that the variable in the list $vars$ at index x takes the value $value$. Here, we use this global constraint to ensure that each fixed time value $t_{i,k}$ is assigned to a start variable $s_{i,j}$. For this, we define variables $f_{i,k}$ representing the index of the variable in the list s_i that will be assigned the value $t_{i,k}$.

$$\max \sum_{i=1}^{n} \sum_{j=1}^{\overline{m}_i} u_i x_{i,j} + v_i p_{i,j} x_{i,j} \tag{25}$$

subject to:

$$s_{i,j} < s_{i,j+1} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{26}$$

$$x_{i,j+1} \implies x_{i,j} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{27}$$

$$x_{i,\underline{m}_i} = 1 \qquad \forall i \in \mathcal{J} \tag{28}$$

$$\text{NoOverlap}(s_{i,j}, p_{i,j}, x_{i,j})_{i \in \mathcal{J}, j \in \mathcal{M}_i} \tag{29}$$

$$x_{i,j+1} \implies s_{i,j+1} - e_{i,j} \leq lag_i^{max} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{30}$$

[5] Git repository: https://gitlab.laas.fr/roc/julien-rouzot/onpsp.

$$!(x_{i,j+1}) \wedge x_{i,j} \implies h + s_{i,0} - e_{i,j} \leq lag_i^{max} \quad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \quad (31)$$

$$x_{i,j+1} \implies s_{i,j+1} - s_{i,j} \geq lag_i^{min} \quad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \quad (32)$$

$$!(x_{i,j+1}) \wedge x_{i,j} \implies h + s_{i,0} - s_{i,j} \geq lag_i^{min} \quad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \quad (33)$$

$$x_{i',j} \implies x_{i,j} \wedge (s_{i,j} < s_{i',j}) \quad \forall (i,i') \in E,\ j \in \mathcal{M}_i \quad (34)$$

$$\textsc{Element}(f_{i,k}, s_i, t_{i,k}) \quad \forall i \in \mathcal{J},\ k \in 1, \ldots, Q_i \quad (35)$$

To make our scheduling tool accessible to the nanosatellite design teams and for wider distribution, we implemented a graphical user interface within NanoSatScheduler. This interface has the following features:

- *Instance Creation*: Users can enter the constraints of their problem by specifying parameters for each constraint for each partition, and indicate their objective function. These instances can be saved/loaded from an existing instance file.
- *Solution Visualization and Editing*: Once the instance is solved, the solution is displayed on the interface, along with solution details such as the objective function value, solution time, and the time used by each partition on the schedule. A solver-independent test tool verifies that all constraints are satisfied. Users can independently visualize each partition, and it is also possible to directly modify the solution through the interface, by setting fixed start times and/or changing tasks duration. This will trigger the solve process again and provide the user with a new solution. If the solution is satisfactory to the user, it can be saved in the appropriate format for direct use with the XTratuM hypervisor.

7 Experimental Results

XTratuM hypervisor [15] comes with a planning tool called XoncretE developed by the company FentISS. This software includes both a graphical interface and a console, and can be used by the NanoLab Academy mission teams. However, XoncretE has several drawbacks: Each occurrence (task) of each partition (job) must be manually declared in the interface; It is not possible to apply *min-lag* and *max-lag* constraints, we can only provide an exact frequency for each partition; The tool does not perform optimization (it does not seek to maximize the number of tasks or their durations). The planning for the hypervisor is therefore usually done as follows: CNES provides the teams with a baseline plan with the partitions common to all nanosatellite missions, and each mission team attempts to manually insert their own partitions into the plan. This represents a laborious task, leading to suboptimal solutions for the teams. We will not compare ourselves here with XoncretE, which is not an optimization software, but we will use the NIMPH mission instance (which is the real-case study) to illustrate the contribution of our method compared to a solution constructed according to the method in place at NanoLab Academy. This instance has 13 partitions and up to 60 tasks to schedule. For the statistical analysis, we also generate

random instances based on the NIMPH instance. Our instance generator can be configured to vary the size of the instances, the minimum/maximum number of each type of partition (i.e. with different type of constraints), and the average proportion of the schedule allocated to each. We ensure that the minimum occurrences are proportional to the given cycle time, in order to generate realistic and non-trivial instances. If, however, some instances generated this way are proven to be unsatisfiable in less than one second by our solver, these instances are discarded.

For our experiments, we use the CP-SAT solver from OR-Tools [17]. All experiments are run on a single-core Intel E5-2695 v3 2.3 Ghz CPU with 32 GB of RAM. For all experiments, the time limit is set to one hour for each instance. As maximizing the number of tasks is often the main goal, we run most of our experiments with this criterion only, but we also optimize both the number of tasks and durations for the NIMPH instance.

We begin by comparing the performance of our two models, MILP and CP, on a set of 100 synthetic instances derived from NIMPH, following the methodology described earlier. For the MILP model, we solve the instances using various time-step sizes (i.e. precision), while the CP model does not require such time scaling. In Table 1, the number of feasible solutions found over the 100 instances is shown. We observe that the matheuristic sometimes fails to build feasible solutions even if a feasible solution to the downscaled problem is found, due to the impact of time-step scaling. Increasing precision (i.e. reducing the step size) may lead to less failures in the reconstruction step but significantly slows the MILP resolution and, in many cases, even prevents it from converging to a solution. The 'gap' column reports the average gap to the best upper bound known when a valid solution is found. Our results show that the MILP model performs poorly on these instances, primarily because of the high number of time steps in NIMPH-like scenarios.

In contrast, the CP model does not require a variable for each job at every time step, allowing for faster solving without sacrificing precision. The CP model finds feasible solutions for all 100 instances and proves optimality for 82 of them. Among the optimally solved instances, 74 of them are solved in less than 1 minute. As the MILP model proves impractical for our use case, we focus the remainder of this section on further experimentation with the CP model.

Table 1. Number of feasible solutions found by the downscaled MILP, number of feasible solutions (after the reconstruction step for the MILP method), mean optimality gap to the best bound known for the valid solutions, for our CP and MILP model with different time step size on 100 synthetic instances.

Method	Step size	MILP Feasible	Feasible	Gap
MILP	5 ms	78	72	5.1 %
MILP	2 ms	66	60	7.6 %
MILP	1 ms	35	34	12.9 %
CP	1 μs	-	100	0.7 %

We now demonstrate the interest of our CP model on the real instance, which corresponds to the NIMPH mission partitions. Here, we maximize two science partitions – namely EDMON and M2M – using lexicographical optimization, as NIMPH team has strict preferences on which one they want to favor. For each stage corresponding to the current partition to optimize, we first maximize the number of tasks, then the total duration of the partition[6]. For this real-world case study, the NIMPH team provided the initial schedule generated by CNES using XoncretE, where the payload partitions must be inserted. To ensure a fair comparison, we created a new instance in which the base schedule consists of tasks with fixed start times and duration. We then apply our lexicographical optimization to insert the payload tasks. The resulting solution represents the best schedule the NIMPH team could have achieved manually, assuming the solution is optimal[7]. Table 2 shows that the primary objective (i.e. maximizing the number of occurrences of the EDMON partition) is significantly worse when starting from a baseline schedule. We generate those solutions in less than 30 seconds, while minutes and generally hours are needed to manually achieve similar solutions. This highlights the limitations of the current scheduling workflow used at the NanoLab Academy, at least for the NIMPH mission, and demonstrates how our method can both improve schedules quality and significantly reduce the time required to generate them.

Table 2. Objective value for each objective ranked in a lexicographical order on NIMPH instance with and without using the baseline schedule provided by XoncretE.

Objective	XoncretE + handcrafted	NanoSatScheduler
Number of tasks EDMON	2	**5**
Duration EDMON	130.0	**307.2**
Number of tasks M2M	**4**	2
Duration M2M	**106.0**	40.0

Even if for NIMPH mission the objective is a strict lexicographical maximization, it might not be the case for other missions. As tuning the weights of the objective function can be a difficult process for the user, we propose to generate Pareto fronts, so the user can directly choose from a set of non-dominated solutions. To compute the Pareto solutions, we use the well-known epsilon-constraint method [8]. Figure 4 presents the Pareto front for NIMPH instance, if we want to maximize the number of tasks for EDMON partition against the number of tasks for M2M, and Figure 5 presents the Pareto front when maximizing the durations. The user can then select the best solution for her/his use case. Note that any other objective combination can be generated (e.g. number of tasks against total duration for the same job). For NIMPH mission, we can see that we can generate a solution that dominates the one reached using lexicographical optimization when using the base schedule in terms of number of tasks.

[6] Maximizing the number of tasks before the total duration is a request from the NIMPH team.

[7] This is the case for the NIMPH instance.

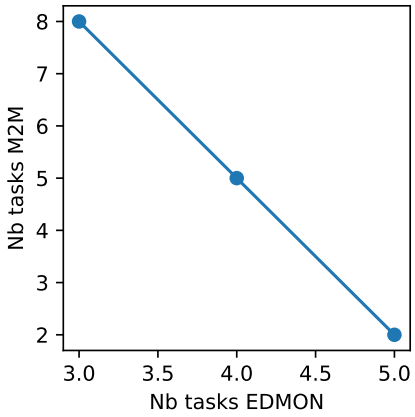

Fig. 4. Pareto front for the number of tasks of partition M2M against EDMON.

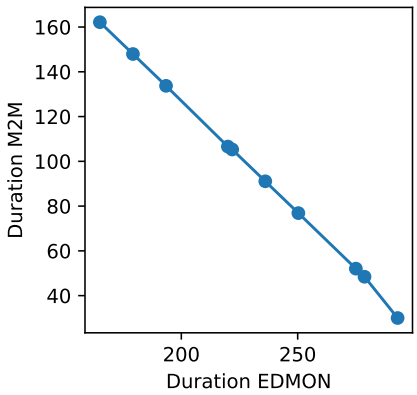

Fig. 5. Pareto front for the total duration of partition M2M against EDMON.

When we started working with NIMPH team, one of our first questions was about the cycle time length, as a single second cycle seemed quite small. According to the NIMPH team, semi-manually scheduling a second-long plan is already laborious, so scheduling longer plans is not considered, due to an overwhelming complexity. As our software handles one-second plans easily, we now analyze the impact of cycle time on the solutions quality. To that end, we generate 100 more synthetic instances and successively solve them and increase the cycle time and number of minimum/maximum tasks of each job accordingly. The time limit for each iteration is fixed to one hour and we add the previous objective function value as a lower bound for the next stage. Table 3 shows the mean number of tasks over a second, the mean solution time and the proportion of optimal solutions for 100 instances, for different cycle time lengths.

We observe that extending the cycle time allows for a slight increase (+0.6%) in the number of tasks over a second. Note that in this case, we would be able to perform more than 2000 additional tasks every hour. Additionally, the proportion of optimal solutions significantly declines as the size of the instances increases.

Table 3. Mean number of tasks (per second), mean solution time, and proportion of optimal solutions, for different cycle time lengths, on 100 synthetic instances.

Cycle time	Number of tasks (per s)	Solution time	Prop. optimal
1 s	110.6	6 min	91 %
2 s	111.1	22 min	66 %
4 s	111.2	45 min	32 %

Another problem of this arbitrary cycle time is the potential sub-optimality of the generated solutions. Indeed, the computed schedules can generally be

compressed to fit a shorter cycle time, thus increasing the average number of tasks performed per second. To that end, we first compute a solution with the baseline cycle time and the given objective (maximize the number of tasks and/or durations of a given set of jobs). Then we solve the instance again, but we add the previous objective value as a constraint and minimize the cycle time. In Table 4, we display the mean cycle time, the mean number of tasks per second, the mean solution time. The second line correspond to solutions where the cycle time is minimized. Note that the proportion of optimal solutions is not compared in Table 4, as the objectives are different for the two stages. For this experiment, the number of tasks per second significantly increases (+3%), which shows that minimizing the cycle time is more efficient than extending it.

Table 4. Mean cycle time, mean number of tasks (per second), mean solution time, with and without minimizing the cycle time, on 100 synthetic instances.

Cycle time	Number of tasks (per s)	Solution time
1 s	110.6	6 min
0.969 s	114.0	34 min

8 Conclusion

In this paper, we presented both a MILP approach and a CP model to tackle the ONPSP, complemented by a user interface that enables visualization and interaction with the generated solutions. Additionally, we provide users with a set of non-dominated solutions, allowing them to explore and analyze the trade-offs between different objectives. Our results demonstrate that the MILP matheuristic faces scalability issues. In contrast, the CP model proves to be highly efficient without compromising precision. For our real-case study, our method performs much better than the current workflow of semi-manually building the schedule, which convinced the NIMPH team to switch to NanoSatScheduler for the mission. Additionally, we show that extending the cycle time has a small impact on the solution quality, whereas minimizing it significantly increases the number of tasks that can be scheduled over time.

For future work, our primary objective is to distribute the software within the CNES NanoLab Academy to gather feedback from other nanosatellite mission teams and improve our approach. In [21], we proposed a CP model that aims at minimizing the context switches only, but this cannot be directly incorporated in our new model. We plan to enhance NanoSatScheduler by accounting for the context switches in the task durations, which will improve solution quality, as the context switch time can be ignored when tasks of the same partitions are continuous[8]. Lastly, it would be valuable to explore whether our method could be extended to broader use cases, particularly in the context of energy-aware scheduling for nanosatellite missions, as discussed in [4, 18–20, 23].

[8] Currently, context switches are accounted for by adding a constant margin to the minimum task duration, so tasks always include the context switch time penalty.

References

1. Al Sheikh, A., Brun, O., Hladik, P.E.: Partition scheduling on an IMA platform with strict periodicity and communication delays. In: 18th international conference on real-time and network systems. pp. 179–188 (2010)
2. Balashov, V.V., Balakhanov, V.A., Kostenko, V.A.: Scheduling of computational tasks in switched network-based IMA systems. In: Proceedings of International Conference on Engineering and Applied Sciences Optimization. pp. 1001–1014 (2014)
3. Bernardo, V.P., Seman, L.O., Bezerra, E.A., Ribeiro, B.F.: Hardware-in-the-loop simulation of an on-board energy-driven scheduling algorithm for cubesats. IEEE Embedded Systems Letters **16**(1), 69–72 (2023)
4. Camponogara, E., Seman, L.O., Rigo, C.A., Morsch Filho, E., Ribeiro, B.F., Bezerra, E.A.: A continuous-time formulation for optimal task scheduling and quality-of-service assurance in nanosatellites. Computers & Operations Research **147**, 105945 (2022)
5. Chen, J., Du, C., Xie, F., Lin, B.: Scheduling non-preemptive tasks with strict periods in multi-core real-time systems. Journal of Systems Architecture **90**, 72–84 (2018)
6. Comer, D., Fossum, T.V.: Operating system design: Internetworking with Xinu. Prentice Hall (1987)
7. Dobiáš, P., Casseau, E., Sinnen, O.: Fault-tolerant online scheduling algorithms for cubesats. In: Proceedings of the 11th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures/9th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms. pp. 1–6 (2020)
8. Haimes, Y.: On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE Transactions on Systems, Man, and Cybernetics **SMC-1**(3), 296–297 (1971)
9. Hanen, C., Munier, A.: Cyclic scheduling on parallel processors: an overview. In: Chrétienne, P., Coffman, E., Lenstra, J., Liu, Z. (eds.) Scheduling theory and its applications. J. Wiley & Sons (1994)
10. Heidt, H., Puig-Suari, J., Moore, A., Nakasuka, S., Twiggs, R.: Cubesat: A new generation of picosatellite for education and industry low-cost space experimentation. In: 14th Annual/USU Conference on Small Satellites, Utah, USA (2000)
11. Karlsson, E., Rönnberg, E., Stenberg, A., Uppman, H.: A matheuristic approach to large-scale avionic scheduling. Annals of Operations Research **302**(2), 425–459 (2021)
12. Kessler Slongo, L., Vega Martinez, S., Vale Barbosa Eiterer, B., Augusto Bezerra, E.: Nanosatellite electrical power system architectures: Models, simulations, and tests. International Journal of Circuit Theory and Applications **48**(12), 2153–2189 (2020)
13. Liubimov, O., Turkin, I.: Optimizing the cubesat on-board computer power consumption under hard real-time constraints. In: Conference on Integrated Computer Technologies in Mechanical Engineering–Synergetic Engineering. pp. 404–414. Springer (2023)
14. Martinez, S.V., Morsch Filho, E., Seman, L.O., Bezerra, E.A., de Paulo Nicolau, V., Ovejero, R.G., Leithardt, V.R.Q.: An integrated thermal-electrical model for simulations of battery behavior in cubesats. Applied Sciences **11**(4), 1554 (2021)

15. Masmano, M., Ripoll, I., Crespo, A., Metge, J.: Xtratum: a hypervisor for safety critical embedded systems. In: 11th Real-Time Linux Workshop. vol. 9 (2009)
16. Minaeva, A., Hanzálek, Z.: Survey on periodic scheduling for time-triggered hard real-time systems. ACM Computing Surveys (CSUR) **54**(1), 1–32 (2021)
17. Perron, L., Didier, F.: CP-SAT. https://developers.google.com/optimization/cp/cp_solver/ (2024)
18. Rigo, C.A., Seman, L.O., Camponogara, E., Morsch Filho, E., Bezerra, E.A.: A nanosatellite task scheduling framework to improve mission value using fuzzy constraints. Expert Systems with Applications **175**, 114784 (2021)
19. Rigo, C.A., Seman, L.O., Camponogara, E., Morsch Filho, E., Bezerra, E.A., Munari, P.: A branch-and-price algorithm for nanosatellite task scheduling to improve mission quality-of-service. European Journal of Operational Research **303**(1), 168–183 (2022)
20. Rigo, C.A., Seman, L.O., Morsch Filho, E., Camponogara, E., Bezerra, E.A.: MPPT aware task scheduling for nanosatellites using MIP-based ReLU proxy models. Expert Systems with Applications **234**, 121022 (2023)
21. Rouzot, J., Gobert, J., Artigues, C., Boyer, R., Camps, F., Garnier, P., Hebrard, E., Lopez, P.: Scheduling onboard tasks of the NIMPH nanosatellite. In: Liberatore, F., Wesolkowski, S., Parlier, G.H. (eds.) Proceedings of the 13th International Conference on Operations Research and Enterprise Systems, ICORES 2024, Rome, Italy, February 24-26, 2024. pp. 277–284. SCITEPRESS (2024)
22. Schild, K., Würtz, J.: Scheduling of time-triggered real-time systems. Constraints **5**, 335–357 (2000)
23. Seman, L.O., Rigo, C.A., Camponogara, E., Munari, P., Bezerra, E.A.: Improving energy aware nanosatellite task scheduling by a branch-cut-and-price algorithm. Computers & Operations Research **158**, 106292 (2023)
24. Turkin, I., Liubimov, O., Zakharenko, V.: Optimization of energy consumption of the cubesat on-board computer under real-time limitations. Aerospace Technic and Technology **0**(3), 126–137 (2024)
25. Vega Martinez, S., Seman, L.O., Morsch Filho, E., Slongo, L.K., Bezerra, E.A.: Onboard energy scheduling optimization algorithm for nanosatellites. International Journal of Circuit Theory and Applications **51**(8), 3915–3937 (2023)
26. Wikum, E.D., Llewellyn, D.C., Nemhauser, G.L.: One-machine generalized precedence constrained scheduling problems. Operations Research Letters **16**(2), 87–99 (1994)
27. Windsor, J., Hjortnaes, K.: Time and space partitioning in spacecraft avionics. In: 2009 Third IEEE International Conference on Space Mission Challenges for Information Technology. pp. 13–20. IEEE (2009)
28. Zhang, W., Behbahani, A.S., Eltawil, A.M.: Joint frequency scheduling and power allocation for cubesat communication. In: 38th International Communications Satellite Systems Conference (ICSSC 2021). vol. 2021, pp. 184–188 (2021)