Hybridizing Machine Learning and Optimization for Planning Satellite Observations[*]

Romain Barrault[1], Cédric Pralet[1], Gauthier Picard[1], Eric Sawyer[2]

[1] DTIS, ONERA, Université de Toulouse, Toulouse, France
[2] CNES, Toulouse, France

{romain.barrault, gauthier.picard, cedric.pralet}@onera.fr
eric.sawyer@cnes.fr

Résumé

La planification des tâches d'un satellite d'observation terrestre est un problème hautement combinatoire. Cela consiste à calculer régulièrement la séquence d'observations à effectuer par un satellite pour collecter des photographies de points d'intérêt tout en tenant compte des manœuvres temporelles nécessaires pour diriger le satellite vers les points d'intérêts successifs. Pour résoudre un tel problème d'optimisation, nous proposons une nouvelle approche qui exploite des techniques d'apprentissage hors ligne pour approximer la faisabilité de l'ordonnancement pour des ensembles de tâches d'observation au sein d'un algorithme d'optimisation.

Mots-clés

IA hybride, Réseaux de Neurones Artificiels, Recherche à Voisinage Large, Planification, Satellite d'Observation Terrestre.

Abstract

Planning the activities of an Earth observation satellite is a highly combinatorial task. It consists in regularly computing the sequence of observations to be performed by a satellite to collect images of candidate points of interest (POIs), while taking into account the time-dependent maneuvers required to point the satellite to the successive POIs. To solve such a recurrent optimization problem, we propose a novel approach that exploits offline learning techniques to approximate scheduling feasibility for sets of observation tasks within an optimization algorithm.

Keywords

Hybrid AI, Neural Networks, Large Neighbourhood Search, Scheduling, Earth Observation Satellite.

1 Introduction

Earth observation satellites (EOSs) are vital for monitoring the planet, supporting diverse applications from climate studies to emergency response. *Agile* EOSs enhance responsiveness through advanced attitude control, enabling swift instrument reorientation towards targets of interest. However, planning the activities of agile EOSs presents significant challenges in terms of optimization. This explains why this problem has received attention over the last 25 years [24], leading to the use of various optimization techniques. The latter include, for example, greedy algorithms that iteratively insert observations one by one to build feasible plans, and several types of metaheuristics such as Large Neighborhood Search (LNS), where a current plan is iteratively optimized using destroy and repair operations. Among the state-of-the-art methods, we can cite an Adaptive LNS (ALNS) where six removal operators and three insertion operators are exploited [12]. Improvements of this algorithm using tabu search mechanisms have been proposed [10], as well as an extension that deals with the multi-satellite case by adding decisions related to the assignment of observations to satellites [9]. Another metaheuristic called GRILS for Greedy Randomized Iterated Local Search has also been recently developed and outperforms ALNS in the multi-satellite context [15]. In another direction, dynamic programming techniques have been studied to deal with cases where the profits collected depend on the time at which the observations are performed [16]. All of these approaches include a critical feasibility constraint that assesses the temporal viability of observation schedules based on satellite agility models. For example, operational satellites may use detailed models that take into account the peculiarities of their on-board momentum control gyroscopes. Frequent evaluation of this constraint during the optimization process can significantly slow down solvers. Consequently, many approaches simplify the agility model by making assumptions such as constant maneuver times or constant angular velocity/acceleration to reduce computational complexity. However, these simplifications can lead to less optimal or even impractical schedules. There are also approaches that first search for solution plans using constant maneuver times, and then use a more complex time-dependent agility model [22].

But a key point is that none of the previous contributions exploit the fact that observation scheduling for EOSs is a highly recurrent optimization problem that is solved every day during several years of operation. Therefore, to go beyond these existing approaches, we propose a set of techniques

[*]Cet article a été accepté à CPAIOR'25.

that allow us to *learn* some aspects of the EOS scheduling problems to be solved regularly. The models learned during an offline phase are then used during an online phase, where EOS scheduling problems have to be solved for precise sets of candidate observations. In the literature, the use of learning methods for satellite scheduling problems has only been tested by a few authors, with attempts to build a decision policy that returns the next observation to insert in the current plan given the features of the candidate requests and the observations already selected [2, 11], and attempts to train a neural network that predicts the probability that an observation can be inserted in the plan of a given satellite [5]. The approach we propose is original compared to existing works because instead of learning an insertion heuristic or an observation insertion success estimator, we try to learn a so-called *scheduling feasibility function*. The latter is a Boolean function that takes as an input a set of POIs I to be observed and returns the value 1 if and only if the execution of all observations in I is considered as feasible from the point of view of the temporal constraints. More specifically, for the offline phase, the idea we promote is to learn a surrogate model that provides a good approximation of the temporal feasibility constraints, which is one of the main bottlenecks of the EOS schedulers. To do this, we first map the temporal feasibility problem to a binary classification problem for a neural networks (NN). We also approximate capacity constraints over the number of POIs that can be observed within orbit sections of different sizes. Since we are trying to infer constraints corresponding to lower bounds on the set of feasible solution plans, obtaining these capacity constraints can be seen as a form of approximate knowledge compilation [21].

In the end, we adopt a hybrid AI and combinatorial optimization approach combining ML-based constraint modeling and classical decision algorithms [13, 1]. Globally, this paper brings several contributions with regard to the existing works:

(i) We show how a complex routing/scheduling problem (the visit of a set of POIs during some time windows) can be approximated as a simpler binary classification problem, by exploiting the geographical nature of the problem;

(ii) We propose an NN-based feasibility classifier that has a surprisingly low average error rate on average in terms of scheduling feasibility;

(iii) We propose an approximation of capacity constraints to help the optimization process, to tackle some challenging POI distributions that the NN struggles with;

(iv) We develop a hybrid AI approach, HySSEO (*Hybrid Selection and Scheduling for Earth Observation*), where the feasibility constraints learned offline are exploited within an online optimization model that searches for an optimal selection of POIs.

Section 2 provides background on the EOS scheduling problem. Section 3 presents the offline learning methods we propose to approximate the scheduling feasibility constraints.

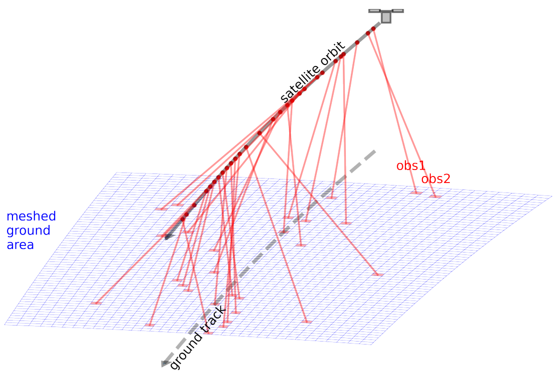


Figure 1: Satellite-centered discretization of the field of view (in blue) and illustration of a set of selected meshes (in red) that can be successively observed by the satellite over its orbit.

Section 4 details the online algorithms used to schedule EOS activities, including the hybrid AI approach proposed. Section 5 provides experimental results concerning the precision of the learned feasibility model and the quality of the schedules obtained. Section 6 concludes and provides perspectives on multi-satellite scenarios.

2 Problem Formalization

We consider a satellite in low Earth orbit that needs to collect acquisitions over points of interest on the ground during one pass over a specific area (during one orbit). Our goal is to select and schedule observations so as to maximize a total reward. The problem is overconstrained in the sense that capturing all candidate acquisitions is usually infeasible.

2.1 Space Discretization and Candidates POIs

For the sake of the learning approach mentioned in the introduction, we define the candidate observations from the point of view of the satellite (satellite-centered representation of the candidate observations), instead of using a fixed meshing of the Earth surface as usually done in the literature. This choice allows us to learn a unique scheduling feasibility model that can be reused for all the satellite revolutions over the Earth, whatever the precise longitude at which the satellite crosses the equatorial plane.

Formally, as illustrated in Figure 1, we build a grid of meshes between two latitudes $LatMin, LatMax$. To compute this grid for a North-to-South pass over all potential observation targets, we follow the process described below (the case of a South-to-North pass is symmetric). First, we consider different rolling angles (right-left rotations of the satellite allowing to capture areas on the left and right of its ground track). More precisely, we consider an angular step δ_r derived from the width of the field of view of the satellite sensor, and a maximum number of steps N_r related to the maximum observation angle usable to capture images. The set of candidate rolling angles is then $R = \{k \cdot \delta_r \mid k \in [-N_r, N_r]\}$. To define different positions of the satellite on its orbit, we consider an initial time t_0 at which the satellite

reaches latitude $LatMax$ and a time-step δ_t derived from the length of the field of view of the sensor and the speed of the satellite on its orbit. We also define the number of time-steps N_t required to reach latitude $LatMin$. The set of candidate positions of the satellite on its orbit is then derived from the set of times $T = \{t_0 + k \cdot \delta_t \,|\, k \in [0, N_t - 1]\}$ and the space mechanic equations. From the previous sets of candidate rolling angles and candidate satellite positions, we build a grid of meshes by projecting, for each configuration, the field of view of the sensor on the Earth ellipsoid. This grid contains H rows and W columns, where $H = N_t$ and $W = 2 \cdot N_r + 1$.
In the following, we denote by $\mathcal{M}$ the set of meshes obtained from this discretization process, and each potential POI of the problem is mapped to the center of a mesh in $\mathcal{M}$.

2.2 Optimization Problem

As an input, we consider a set of candidate observations $M \subseteq \mathcal{M}$. This set is strictly included in $\mathcal{M}$ when the end-users require images only on specific points on Earth. For each candidate observation $i \in M$, we have: (i) a position $P_i \in [1..H] \times [1..W]$ in the grid; (ii) a time window $[S_i, E_i]$, during which observation i can be performed, derived from a fixed maximum observation angle and from the ability of the satellite to point towards right-left and/or forward-backward directions; (iii) a reward Rw_i depending on the user requirements and the weather forecast (the higher the cloud coverage prevision, the lower the reward).
Additionally, the satellite must maneuver between successive observations in order to point its sensor towards the right direction at each step. In the following, we denote by tt the transition function such that $tt(i, j, t)$ returns the duration required by a maneuver from a configuration where the satellite is pointed to observation $i \in M$ to a configuration where the satellite is pointed to observation $j \in M$, when the maneuver starts at time t. This transition function is *time-dependent* (it depends on t) mainly due to the motion of the satellite on its orbit along time.

Definition 1 (EOSP). *The* Earth Observation Scheduling Problem *(or EOSP) consists in finding a sequence of observations $\sigma = [\sigma_1, \ldots, \sigma_K]$ such that: (i) each candidate observation in M appears at most once in σ; (ii) the successive observations can be performed during the allowed time windows; formally, the earliest start time of the first observation is $s_{\sigma_1} = S_{\sigma_1}$, the earliest start time of the kth observation is given by $s_{\sigma_k} = \max(S_{\sigma_k}, s_{\sigma_{k-1}} + tt(\sigma_{k-1}, \sigma_k, s_{\sigma_{k-1}}))$, and condition $s_{\sigma_k} \leq E_{\sigma_k}$ must be satisfied for every observation σ_k involved in σ; (iii) the total reward collected ($\sum_{i\in\sigma} Rw_i$) is maximized.*

In terms of Operations Research, the problem obtained is a Time-Dependent Orienteering Problem with Time Windows (TD-OPTW) [20]. This problem is known to be NP-hard [8]. Therefore, it is usually addressed using incomplete methods, such as ant colony optimization [23], iterated local search [7], or large neighborhood search (LNS) [20]. The mapping between EOSP and TD-OPTW has been studied in previous works and attacked using large neighborhood search specifically [12]. Most of these local search or metaheuristic methods start searching from good quality solutions found by a heuristic greedy algorithm. We also use this approach as a baseline in our investigations.

3 Learning the Schedule Feasibility

For an EOS, schedules for successive orbits must be optimized several times a day over several years of operation. To solve this highly recurrent problem, we propose to learn, during an offline phase, a set of constraints called *feasibility constraints* that should be satisfied by any selection of observations.

3.1 Global Approach

To learn feasibility constraints during an offline phase, we combine two distinct learning approaches: (1) based on machine learning methods, we learn a globally feasibility function *feasibilityNN* evaluating whether a set of POIs can be observed during a single satellite pass; (2) based on operations research methods, we learn hard capacity constraints limiting the number of POIs that can be observed within specific areas. More specifically, for the second point, we consider a set of rectangular areas A in the grid of meshes, and for each area $a \in A$ we try to compute the maximum number of POIs that can be observed within a during a single satellite pass, referred to as $Capacity_a$.
During the online phase where a set of candidate observations M is available, the idea is to exploit the previous feasibility constraints to compute in a coarse-grain fashion an optimal selection of observations, before working on detailed scheduling decisions. Equation (1), Equation (2), Equation (3) and Equation (4) give the corresponding coarse-grain optimization model, where Boolean variable x_i takes value 1 if the ith candidate mesh is selected, and value 0 otherwise. In Equation (2), function *feasibilityNN* exploits the global feasibility function expressed as a neural network classifier to determine whether an assignment of the x_i variables is feasible. In Equation (3), the hard capacity constraints enforce upper bounds about the selection. As a result, the hybrid approach proposed combines on the one hand a constraint-based optimization model that must be solved online, several times per day, to compute a plan for the satellite over the next decision horizon, and on the other hand learning techniques through the offline computation of function *feasibilityNN* and bounds $Capacity_a$.

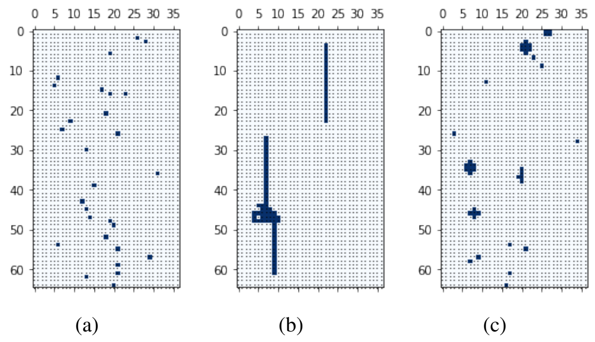
$$\text{maximize } \textstyle\sum_{i\in M} Rw_i \cdot x_i \tag{1}$$

$$\mathit{feasibilityNN}(x_i, \ldots, x_{|M|}) = 1 \tag{2}$$

$$\forall a \in A, \sum_{i\in M \,|\, P_i \in a} x_i \leq Capacity_a \tag{3}$$

$$\forall i \in M,\ x_i \in \{0, 1\} \tag{4}$$

One of the main advantages here is that the optimization model proposed is focused on the selection aspect and does not directly use the time-dependent transition function tt that is associated with costly computations. As the calls to the transition function are the main bottleneck of the



(a) (b) (c)

Figure 2: Examples of maximum sets of meshes observable by the satellite during a single pass (mesh indices on the x-axis and y-axis): (a) sparse instance, (b) dense instance, (c) mixed instance

search methods in practice, building a surrogate model of the feasibility function is highly relevant. In the following we successively detail how the global feasibility function $feasibilityNN$ and the capacities $Capacity_a$ are learned.

3.2 Learning a global feasibility function

3.2.1 From a Scheduling Feasibility Function to a Classification Problem.

From a machine learning point of view, if we consider the grid of candidate meshes as an image where each pixel is black if the corresponding mesh is activated and white otherwise, our goal is to classify the images depending on whether they correspond to feasible sets of observations. On such an image, the satellite ground track would consist in a straight vertical line crossing the middle of the x axis. We will see later how we consider other local and global features, derived from this initial image.

To illustrate the approach, Figure 2 provides images corresponding to sets of meshes for which it is possible to find feasible observation schedules given a realistic maneuver model. More specifically, Figure 2(a) gives the image associated with the set of meshes activated in Figure 1. In this case, the set of activated meshes is sparse and only a reduced number of meshes are observed due to the time consumed by the satellite maneuvers between the corresponding acquisition tasks. On the other side, Figure 2(b) gives an example of an image where the set of activated meshes is dense. In this case, many contiguous meshes are activated and many small maneuvers are used, which is why the satellite is able to observe many meshes during a single pass. Figure 2(b) also shows standard patterns where the satellite observes strips of contiguous meshes that are parallel to its ground track. Last, Figure 2(c) corresponds to a mixed instance involving both clusters of meshes and individual meshes spread over the observation area.

3.2.2 Input features and NN architecture.

In our study, three kinds of features are considered to classify the images according to the feasibility of the mesh selection:

- *Image*: raw features corresponding to the activation of individual pixels in the mesh selection image;
- *Locals*: features of a set of predefined slices of the grid. Globally, we consider successive slices of height h separated by r rows, h and r being two parameters of the method. These slices are defined as $Slice_{k\cdot r+1,k\cdot r+h} = [k\cdot r+1..k\cdot r+h] \times [1..W]$ for different values of k, so as to cover the whole grid. For each slice $Slice_{k\cdot r+1,k\cdot r+h}$, we compute two features: $nMeshesInSlice_k$ that represents the number of meshes activated in the slice, and $dispersionInSlice_k$ that approximates the total maneuver time required to successively visit the columns where meshes are activated in the slice. On the last point, if we denote by $Columns_k = [c_1,\dots,c_Q]$ the indices of the successive columns where meshes are activated, we have $dispersionInSlice_k = \sum_{i\in[1..Q-1]} TI\_ttCols(c_{i+1}-c_i)$ where $TI\_ttCols(\delta)$ stands for a Time-Independent approximation on the minimum transition time required by any maneuver traversing δ columns; such a transition time is non-linear in δ, especially for small maneuvers requiring an acceleration phase for the satellite;
- *Globals*: features representing global metrics over the set of selected meshes. Two global features are exploited: $nActive$, the total number of meshes activated in the grid, and $costMST$, the cost of a minimum spanning tree covering all the activated meshes, given that the cost of an edge between two activated meshes i and j is defined from a Time-Independent approximation $TI\_tt(i,j)$ of the (time-dependent) transition time between i and j. Note that the cost of a minimum spanning tree is also used as an efficient lower bound in works on traveling salesman problems with time windows [6].

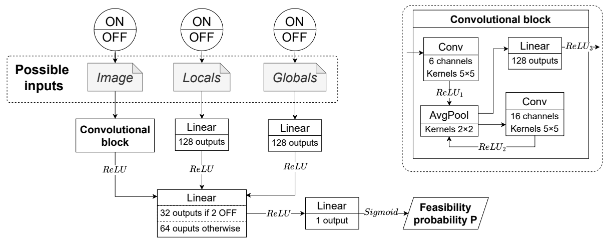


Figure 3: Common architecture of designed neural networks

Figure 3 displays the architecture shared by all neural networks designed in this work. The first feature to point out is that as a binary classifier, it returns a single number between 0 and 1 (thanks to the sigmoid activation function) embodying probability ; in this case, it is a feasibility probability for a given set of meshes. For the inputs, the three kinds of features listed before can be activated or deactivated, which gives us eight configurations ranging from a case where the neural network is empty to a case where all the features are used.

3.2.3 Feasibility Classifier and Generation of Training Instances.

To train the NN classifier, we first build a set of feasible mesh selections $\mathcal{P}_{max}$ that are maximal in terms of inclusion. This means that for each maximal set $P \in \mathcal{P}_{max}$, each mesh selection $P' \subseteq P$ is feasible and the scheduling algorithm estimates that it is not possible to activate one more mesh in P while keeping the feasibility of the selection. The examples provided in Figures 2(a) à 2(c) are maximal positive instances obtained from a realistic time-dependent maneuver model. On this point, the maximal instances built are actually *approximately maximal* in the sense that the feasibility of a mesh selection is tested based on an approximate greedy algorithm that inserts the observations one by one in the current sequence of visits, each time at a position that is considered as the best one. Other efficient scheduling algorithms could be used [18], but it is worth mentioning that determining whether a single mesh can be added to a mesh selection while preserving the scheduling feasibility is NP-hard, due to the NP-hardness of the Traveling Salesman Problem with Time Windows [19]. This is why we only use approximate scheduling algorithms to estimate whether a set of mesh activations is maximal.

Then, for each maximum instance $P \in \mathcal{P}_{max}$, we generate N positive (resp.) instances by randomly deactivating (resp. activating) meshes. To better approximate the feasibility-infeasibility frontier, the number of positive (resp. negative) instances generated is higher for small numbers of meshes deactivations (resp. activations). Details about the instance generation protocol are given in the experimental section. All the positive and negative instances generated are used to train the neural network, together with all the maximum instances.

3.3 Learning capacity constraints

We now describe how the areas A and the bounds $\{Capacity_a, \, a \in A\}$ used in Equation (3) are defined. We recall that H and W respectively denote the height and width of the grid of meshes. The areas manipulated correspond to slices of meshes of various heights. More precisely, for each $h \in [1..H]$, we consider a slice $Slice_{1,h} = [1..h] \times [1..W]$ covering h successive rows and the entire grid width. Then, we solve an EOSP as defined in Definition 1 for a set M containing one candidate observation i per mesh in $[1..h] \times [1..W]$, with a unit reward $Rw_i = 1$. The best total reward $TotalRw^*_h$ found for area $a = Slice_{1,h}$ is then used to define capacity $Capacity_a$, that is $Capacity_a = TotalRw^*_h$. Due to the invariance of the problem along the grid rows, reward $TotalRw^*_h$ is exploited to define a maximum capacity not only over slice $Slice_{1,h}$, but also over all slices $Slice_{k+1,k+h} = [k+1..k+h] \times [1..W]$ of height h for $k \in [0..H-h]$. Note that for $h = H$, total reward $TotalRw^*_H$ gives the maximum number of meshes that can be selected over the whole grid.

From a theoretical point of view, the previous process guarantees that any solution of an EOSP problem defined over the grid must satisfy the capacity bounds, otherwise the optimal rewards found above would not be optimal. However, one difficulty is that solving the previous EOSP problem in NP-hard, because it contains the standard Traveling Salesman Problem with Time Windows as a special case [19]. As a result, finding the optimal total reward $TotalRw^*_h$ is not necessarily easy for large grid slices.

To overcome this difficulty, we exploit two main ideas. First, we use an incomplete LNS solver instead of a complete search engine to optimize the total reward. This LNS solver is run during an arbitrary CPU time set to h minutes. This implies that the capacities computed are not guaranteed to correspond to actual hard constraints, but it is worth mentioning that LNS is a state-of-the-art incomplete method for Time-Dependent Orienteering Problems with Time Windows [20]. Second, when optimizing the total reward for slice $Slice_{1,h+1}$, we reuse the solution found for slice $Slice_{1,h}$. Basically, the best solution found for a slice $Slice_{1,h}$ selects customers within column range $[cmin_h, cmax_h]$. In practice, these customers are always placed around the ground track of the satellite because such customers have time windows that are large and contain the windows of customers placed near the borders of the grid. To simplify the resolution for height $h+1$, we consider only the customers located in area $[1..h+1] \times [cmin_h, cmax_h]$. For high values of h, this heuristic approach strongly simplifies the problem because the number of columns to take into account is quickly reduced to only 3 or 4.

4 Integrating the Learned Model into a Scheduler

This section describes how to exploit such trained models within an EOSP solver. We first recall conventional methods.

4.1 Greedy Scheduler

In order to solve a planning and scheduling problem such as EOSP or TD-OPTW, greedy algorithms are candidates of choice. Basically, greedy algorithms tackle problems by making the choice that seems the best at each decision step. While this usually fails to lead to the globally optimal solutions, it often creates solutions that have a quite good quality [3]. In the Space domain, for operational satellites, greedy algorithms are the baseline when it comes to scheduling tasks, because (i) they are fast (polynomial time in general), (ii) they can be guided by many efficient heuristics exploiting information about the observation tasks and the available time windows.

In the case of EOSP, Algorithm 1 sketches a greedy algorithm, looping over candidate observations. At each step, the idea is to select the best observation that can be added to the current schedule and to insert this observation at its best possible position in the sequence of observations. The process continues until all the observations are scheduled or there is no more place left in the current solution.

Selecting the next observation to schedule and its insertion position is a decision that strongly impacts the quality of the final solution. For this, the algorithm relies on two subfunctions that instantiate specific greedy heurist-

Algorithm 1 : GreedySch

```
Input : An EOSP $p$
Output : An admissible solution $\sigma$
$\sigma \leftarrow []$
$continue \leftarrow true$
while $|\sigma| \neq |M|$ and $continue$ do
    i ← selectObs($\sigma$,$M$)
    if i $\neq \emptyset$ then $\sigma \leftarrow$ scheduleObs(i, $\sigma$, $M$)
    else $continue \leftarrow false$
return $\sigma$
```

Algorithm 2 : Generic LNS

```
Input : A problem $p$
Input : An admissible solution s
Output : An admissible solution s*
s* ← s
while stop criterion is not met do
    s′ ←repair(destroy(s))
    if accept(s′,s) then s ← s′
    if $\sum_{i \in s'} Rw_i > \sum_{i \in s^*} Rw_i$ then s* ← s′
return s*
```

ics. Function `selectObs()` first selects an observation from set M given the current solution σ. Here, we consider that the best observation is the observation $i \in M$ that is not already contained in σ and maximizes the ratio between the reward Rw_i and the additional time Δ_i required to maneuver between its predecessors and successors σ_k and σ_{k+1} at the best feasible insertion position in σ. By denoting as s_{σ_k} the earliest start time of observation σ_k in the current schedule and by $s_i = \max(S_i, s_{\sigma_k} + tt(\sigma_k, i, s_{\sigma_k}))$ the earliest time at which observation i can be performed after σ_k, this additional maneuver time is $\Delta_i = tt(\sigma_k, i, s_k) + tt(i, \sigma_{k+1}, s_i) - tt(\sigma_k, \sigma_{k+1}, s_k)$. The observation selection heuristic then chooses an observation in $\arg\max\{\frac{Rw_i}{\Delta_i} \mid i \in M, i \notin \sigma\}$, the idea being to favor observations that have the best "reward over maneuver time" ratio given the current schedule, or in other words the best yield. Function `scheduleObs()` positions the observation selected at the right place in the schedule. Note that `selectObs()` and `scheduleObs()` can be optimized by sharing some computations and data, but we omit these details for sake of conciseness.

4.2 Large Neighborhood Search (LNS)

LNS is a metaheuristic method employed in optimization problems, particularly those involving combinatorial optimization [17]. It is an advanced version of the simpler local search technique. Local search begins with an initial solution to a problem, followed by an exploration of "neighboring" solutions that are generated by making minor adjustments to the current solution. The aim is to discover a solution that optimizes a specific objective function (e.g., minimizing expenses or maximizing productivity). However, local search can become trapped in local optima, where all surrounding solutions are dominated by the current one. LNS addresses this limitation by examining a much larger set of neighboring solutions. In this extended neighborhood, the solution modifications can be more significant, potentially involving larger alterations of the current solution. This wider search allows LNS to potentially escape from local optima and discover superior solutions overall.
An LNS algorithm for the EOSP is sketched in Algorithm 2. It mainly alternates between `destroy()` and `repair()` operations to find a better solution. The `destroy()` operation consists in deactivating some meshes from the current solution, and the `repair()` operation consists in refilling the solution plan through mesh activations. To decide whether the next destroy and repair operations have to work on the new solution obtained, we use function `accept()` that can be implemented in different ways. The simplest choice is to accept any solution after the repair process.
In our investigations we consider two LNS solvers:

LNS for Selecting Observations (LNSSel): in this case, solutions are subsets of M, function `destroy()` removes a portion of observations from the current solution with a priority for the deactivation of the observations that have a small ratio between the reward and the distance to the set of activated meshes, and function `repair()` adds as many observations as possible in a greedy manner while checking the feasibility with $feasibilityNN$. Since checking the feasibility using the surrogate is very fast, we can also afford multiple restarts until reaching a given time limit. A restart is performed after a fixed number of LNS iterations, and to perform a restart we generate a new initial solution containing 15 randomly chosen observations, enriched with as many observations as possible, in a greedy manner.

LNS for solving an EOSP (LNSSch): in this case, solutions are sequences of observations from M. Function `destroy()` randomly removes observations from the current sequence with a priority to observations close to the ones that just got removed, while function `repair()` adds and schedules as many observations as possible in a greedy manner, by calling the transition function. Here, each observation activation involves more computations than in LNSSel.

4.3 Using the Trained Models when Solving EOSP

Figure 4 illustrates our hybrid approach. It is composed of an offline phase and an online phase. During the offline phase, feasibility classifier is trained, and capacity constraints are determined, as explained in Section 3. The online phase first selects a subset of good candidate observations M' using an LNS based on our $feasibleNN$ model and capacity constraints. Then, this subset is used as an input by an LNS-based scheduler, whose initial point is obtained by a greedy algorithm that attempts to schedule as many observations from M' as possible. This algorithm is developed to answer two objectives. (i) The first objective is to take advantage of

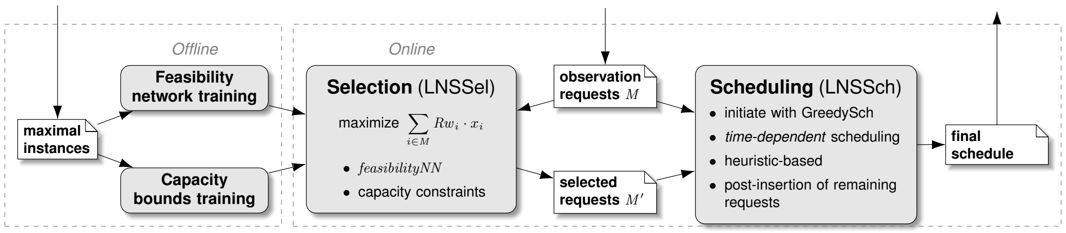


Figure 4: HySSEO components: (i) training on maximal instances is performed offline; (ii) from the initial set of observation requests M, the selection component selects a subset M' by using the trained feasibility model and capacity constraints; (iii) finally M' is used in priority by an LNS-based scheduler, that could add more requests from M if there is room for.

the surrogate feasibility function $feasibilityNN$ and capacity constraints to quickly compute good-quality solutions by exploring many observation selection strategies, instead of directly using complex time-dependent scheduling operations. (ii) The second objective is to challenge the robustness of the surrogate model $feasibilityNN$, by first searching for an optimal set of selected meshes $M' \subseteq M$ according to this model and then checking, based on the standard LNS algorithm for EOSP, whether all meshes in M' can actually be simultaneously observed. From a machine learning point of view, this objective is more challenging than just classifying some randomly generated instances.

5 Experimental Evaluation

Now, we provide an experimental analysis on both the precision of the learned classifier and its capacity to select good sets of candidate observations within an EOSP solving process. The EOSP solvers are implemented in Java and executed on 20-core Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz, 62GB RAM, Ubuntu 18.04.5 LTS, with an OpenJDK 11.0.9 JVM. The machine learning models have been developed and pre-trained using pytorch library in Python 3.11 [14]. These models have been serialized in ONNX and then loaded and called by our solvers using the ONNX Runtime for Java [4].

5.1 Generation of the positive and negative instances

From each maximal set of meshes $P^* \in \mathcal{P}_{max}$, we generate 35 *positive instances* by deactivating m meshes belonging to P^* (random choice of the meshes deactivated). To better approximate the feasibility-infeasibility frontier, we generate more positive instances for small values of m: we generate $K_1 = 6$ random positive instances for $m = 1$, and for $m \in [2, 10]$, we generate $K_m = round(5/6 \cdot K_{m-1})$ positive instances. Additionally, we generate $K_{>10} = 6$ positive instances where more than 10 meshes are deactivated. Similarly, for each maximal instance $P^* \in \mathcal{P}_{max}$, we generate 35 *negative instances* by randomly activating at least one mesh that does not belong to P^* (random choice of the meshes activated). Again, to better approximate the feasibility-infeasibility frontier, we generate more negative instances for small values of m, using the values of K_m introduced before. For last, each instance belongs to one out of five archetypes: *dense* instances show a huge amount of activated meshes that are close to each other, implying small maneuvers; *mixed* instances have many isolated activated meshes with a few clusters of close meshes; *sparse* instances only have isolated meshes, and therefore a smaller amount of observable meshes due to higher maneuver durations; *flat* instances show meshes displayed mostly horizontally, which is less easy to capture for the EOS; *borders* instances are characterized by a high amount of meshes near the borders of the meshing, which have smaller visibility time windows. We aim to cover realistic cases and distinguish the ease of observing a whole set of POIs based on the geometry of the instance.

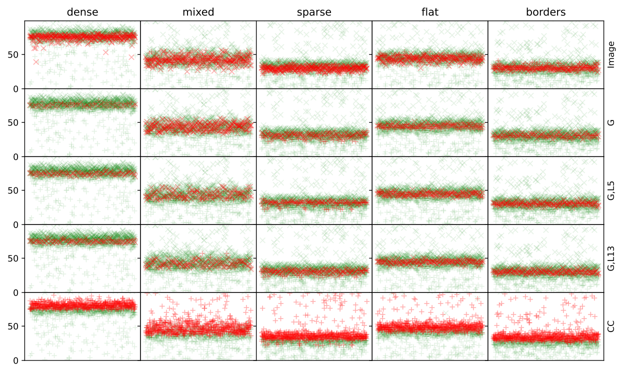


Figure 5: Classification results for five classifiers (Image, G, {G,L5}, {G,L13}, and CC) on five types of instances (*dense*, *mixed*, *sparse*, *flat* and *borders*). On each graph, each dot corresponds to a test instance classified by the model, green if correctly classified and red otherwise. Negative instances are represented by crosses and positive instances by pluses. The x-axis is for the instance number while the y-axis is the amount of activated meshes in the instance.

5.2 Precision of the learned classifier

As explained in Section 3.2.2, several NN models have been designed for different input features. Figure 5 shows the results of several such models. Models Image and G take as input only the corresponding features described in Section 3.2.2. {G,L5} and {G,L13} are based on global and local features computed according to the description of Section 3.2.2. {G,L5} considers successive slices of height $h = 5$ separated by $r = 1$ row, while {G,L13} considers

Table 1: Error rates for the classifiers and each instance type: *dense* (D), *mixed* (M), *sparse* (S), *flat* (F), *borders* (B).

NN model	Image					G					{G,L5}					{G,L13}					CC				
Instance type	D	M	S	F	B	D	M	S	F	B	D	M	S	F	B	D	M	S	F	B	D	M	S	F	B
Error rate (%)	25	24	25	21	20	4.6	19	11	11	9.8	6.6	11	12	13	13	6.6	9.1	12	12	13	42	45	46	45	46

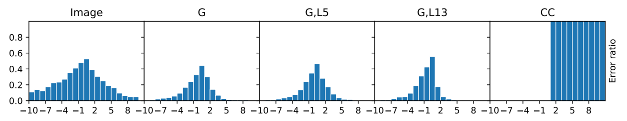


Figure 6: Error rates of designed neural networks for different numbers of meshes added (positive value) or removed (negative values) from maximal instances.

successive slices of height $h = 13$ separated by $r = 6$ rows, implying much less NN inputs. Finally, a classifier based only on the capacity constraints (CC) claims that an instance is feasible if and only if it satisfies these constraints. The error rates of all these models are shown in Table 1, which indicates that models with global features perform best. Models with the image input do not show good performance in the current architecture, which is why our study is focused on the three other NN models. In addition, since capacity constraints are necessary feasibility conditions, they classify instances poorly on their own. On this point, only a minor part of the negative instances in the dataset are well classified by the CC approach, since activating less than 10 additional meshes rarely leads to a violation of the capacity constraints. Figure 6 provides the error rates of the models as a function of the number of meshes activated and deactivated in the positive and negative instances respectively. It shows that the G and {G,L5} models appear to be slightly more robust while {G,L13} gives fewer false positives. Finally, the CC classifier returns no false negative. This makes sense since it classifies the instances based on necessary feasibility conditions, and it returns only false positives. Consequently, only the three most promising models are kept for testing in our HySSEO method.

5.3 Performances of HySSEO

Table 2 shows the results of the runs performed to evaluate the online algorithm. The learned model used in the LNSSel part of HySSEO varies between G, {G,L5}, {G,L13}, and all of the feasibility checks used in this part also verify the capacity constraints learned offline. As preliminary results have shown that neural networks alone are not sufficiently constrained on dense instances and may return large sets of observations whose cardinality exceeds the capacity constraints, models G, {G,L5}, and {G,L13} are combined with CC in all of our tests. A last HySSEO algorithm is used to be compared with and does not include any NN: it only checks the capacity constraints. The algorithms are run for 1 and 5 minutes in order to compare their ability to converge faster or produce better results with enough time. For a fixed timespan, in the HySSEO algorithms, 16% of that

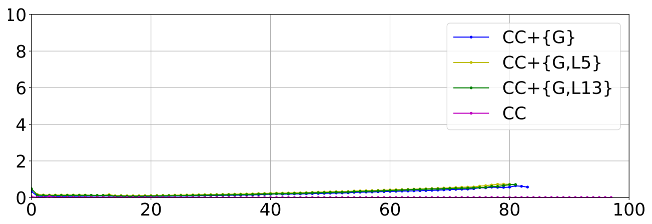


Figure 7: Average inference time (in ms) for each number of activated meshes in the current solution.

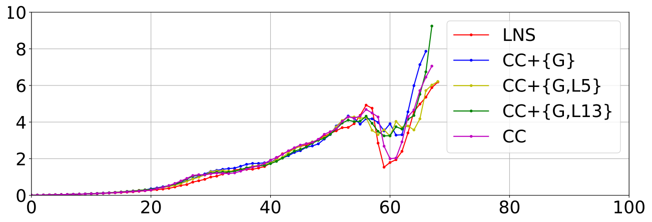


Figure 8: Average insertion time (in ms) of an observation in the current schedule as a function of its current size.

timespan is used by LNSSel, 34% is used by LNSSched on the set of observations returned by LNSSel, and the remaining 50% of the computation time is used to optimize the insertion of other remaining requests. Table 2 shows the performance of the algorithms with $mean \pm std$ on 100 instances equally composed of dense, mixed and sparse instances, with 200 observation requests. The optimality gap is obtained by comparing to an LNSSched algorithm working on the full set of requested observations during a longer computation time of 1 hour. In the table, *nSites* is the number of observations involved in the final schedule, and LNS_1 (resp. LNS_2) refers to the first (resp. the second) use of LNSSched in HySSEO. The results show that the baseline LNS is beaten on average by all of our HySSEO algorithms, regardless of the allowed computation time. In addition, except for dense instances where CC + { } is highly efficient, using one of our three neural networks to compute feasibility checks improves the solution quality. In the experiments, the capacity constraint checks tend to return much too large request sets on mixed and sparse instances since it is blind with regards to the actual maneuver times required within a slice of meshes. However, we can note that configuration CC returns the best solutions after the first LNSSched, which is quite natural since it is easier to obtain a good solution when requests are less filtered. On the other hand, for the three NN-based HySSEO algorithms, the post-insertion process has a higher impact since the initial LNSSel phase is more constrained in this case. These three NN-based approaches show similar results in terms of solution quality. Figures 7

Table 2: Scheduling performance metrics.

Instance type		1 minute					5 minutes				
		LNS	CC+{G}	CC+{G,L5}	CC+{G,L13}	CC+{ }	LNS	CC+{G}	CC+{G,L5}	CC+{G,L13}	CC+{ }
Dense	$nActive$	n/a	61.6 ± 13.6	59.6 ± 11.3	60.6 ± 12.5	62.0 ± 14.2	n/a	61.5 ± 13.5	60.1 ± 11.8	60.6 ± 12.6	62.1 ± 14.2
	$nSitesLNS_1$	n/a	42.0 ± 9.0	41.8 ± 8.7	$\mathbf{42.1 \pm 9.0}$	41.5 ± 8.8	n/a	42.4 ± 8.9	42.4 ± 9.0	$\mathbf{42.4 \pm 9.4}$	41.9 ± 8.9
	$gapLNS_1(\%)$	n/a	7.6 ± 3.9	8.3 ± 3.6	7.7 ± 4.2	7.2 ± 4.0	n/a	5.9 ± 2.7	6.6 ± 2.7	6.6 ± 3.5	6.1 ± 3.1
	$nSitesLNS_2$	$\mathbf{47.2 \pm 9.1}$	45.6 ± 9.1	45.9 ± 8.6	45.7 ± 9.2	45.6 ± 8.9	$\mathbf{47.8 \pm 9.0}$	46.8 ± 9.1	46.9 ± 9.0	47.0 ± 9.1	46.7 ± 8.7
	$gapLNS_2(\%)$	5.5 ± 3.6	3.6 ± 2.7	3.9 ± 2.0	3.5 ± 2.2	$\mathbf{3.3 \pm 2.0}$	2.8 ± 2.2	2.0 ± 1.4	1.9 ± 1.5	2.0 ± 1.3	$\mathbf{1.8 \pm 1.5}$
Mixed	$nActive$	n/a	66.5 ± 4.9	59.3 ± 3.5	56.8 ± 4.0	97.5 ± 0.7	n/a	67.4 ± 4.0	60.2 ± 3.4	58.1 ± 4.2	97.5 ± 0.7
	$nSitesLNS_1$	n/a	46.4 ± 5.3	$\mathbf{47.5 \pm 3.8}$	47.4 ± 5.2	46.9 ± 2.7	n/a	46.5 ± 5.5	47.9 ± 4.0	$\mathbf{48.3 \pm 4.5}$	48.1 ± 2.9
	$gapLNS_1(\%)$	n/a	18.8 ± 6.8	17.3 ± 6.7	17.8 ± 6.9	10.0 ± 3.9	n/a	17.6 ± 6.7	14.8 ± 5.7	15.3 ± 5.7	7.0 ± 3.2
	$nSitesLNS_2$	52.1 ± 3.3	54.0 ± 2.9	53.9 ± 3.0	$\mathbf{54.1 \pm 3.0}$	51.8 ± 2.8	53.4 ± 2.7	$\mathbf{54.8 \pm 2.6}$	54.3 ± 2.9	54.7 ± 2.7	53.5 ± 2.4
	$gapLNS_2(\%)$	7.4 ± 3.9	4.8 ± 2.5	4.8 ± 2.4	$\mathbf{4.4 \pm 2.3}$	5.4 ± 3.2	4.0 ± 3.4	$\mathbf{1.7 \pm 1.5}$	2.2 ± 1.6	2.0 ± 1.6	2.4 ± 2.2
Sparse	$nActive$	n/a	49.2 ± 1.7	48.9 ± 2.1	46.3 ± 2.1	97.8 ± 0.4	n/a	49.1 ± 1.3	49.5 ± 2.2	46.7 ± 2.6	97.7 ± 0.5
	$nSitesLNS_1$	n/a	39.3 ± 4.2	43.1 ± 3.0	42.6 ± 2.8	$\mathbf{43.5 \pm 1.6}$	n/a	39.6 ± 3.8	43.9 ± 3.0	43.6 ± 2.9	$\mathbf{44.7 \pm 1.9}$
	$gapLNS_1(\%)$	n/a	18.1 ± 7.6	13.1 ± 4.6	16.1 ± 4.6	7.7 ± 3.0	n/a	16.2 ± 7.1	11.1 ± 4.3	13.3 ± 3.8	5.2 ± 2.9
	$nSitesLNS_2$	46.8 ± 1.6	47.2 ± 1.7	47.5 ± 1.7	$\mathbf{47.9 \pm 1.9}$	46.5 ± 1.5	47.5 ± 1.4	47.7 ± 1.6	48.0 ± 1.3	$\mathbf{48.2 \pm 1.5}$	47.4 ± 1.8
	$gapLNS_2(\%)$	6.6 ± 2.7	$\mathbf{3.1 \pm 2.0}$	4.3 ± 2.0	5.0 ± 2.5	5.0 ± 2.4	3.7 ± 2.0	2.1 ± 1.3	$\mathbf{2.0 \pm 1.7}$	2.5 ± 1.9	2.6 ± 2.1

et 8 display the average inference time for feasibility checks in `LNSSel` and the average insertion time of a request into the current schedule in `LNSSched`, respectively. As expected, our learned model allows us to check the feasibility of a set of requests much faster than the feasibility of an actual observation insertion into a schedule. Note that the first kind of checks slightly increases with the number of activated meshes due to the operations required to compute features such as $costMST$, while the second kind of checks is much slower since it requires testing each insertion position in the worst case. In Figure 8, the behavior around 60 requests is due to the different kinds of instances involved in our dataset. More precisely, for many instances, the maximum number of selectable observations is around 60, and in this case testing all the possible insertion positions is long, while for other instances (typically the dense ones), the maximum number of selectable observations is much higher, and in this case finding a feasible insertion position is not that hard.

6 Conclusion

This paper proposes a novel approach to approximate, using a neural network, a feasibility constraint for a set of observation tasks by a satellite. The main idea is to map the feasibility check to a classification task, based on local and global features of the set of POIs. We evaluated the classification performance of several combinations of features, and used them to select good subsets of candidate observation tasks upstream of a conventional scheduling process based on greedy search and large neighborhood search. Combined with this classifier we also propose approximate capacity constraints, to help the NN discriminate unfeasible observation sets. The resulting feasibility classifiers have a small error rate, even on very hard instances, and HySSEO is better than baseline methods in terms of solution quality. Computing local features for feasibility does not seem to improve the solution quality, and combining global features and capacity constraints is sufficient to provide results among the best.
One of the next steps is to apply this hybrid AI approach to plan the observations of several satellites. In this case, the problem to be solved is similar to a team orienteering problem. The latter is much harder to solve since it involves decisions on the allocation of observation tasks to the satellites, instead of just observation selection decisions. On this line, we believe that the techniques proposed and tested in a mono-satellite context can be highly relevant in multi-satellite scenarios, to explore large sets of observation dispatching decisions contrarily to traditional EOS planning methods that are slowed down by the computation of candidate maneuvers for many candidate observation insertion positions across all the satellites of the constellation. In another direction, we are also considering using our feasibility model as a constraint in constraint programming or linear programming frameworks, as in the Empirical Model Learning approach, which consists in encoding neural networks into linear constraints, so as to approximate non-linear or costly constraints [13].

References

[1] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[2] Eric Bensana, G Verfaillie, C Michelon-Edery, and Nicolas Bataille. Dealing with uncertainty when managing an earth observation satellite. In *Fifth International Symposium on Artificial Intellgence, Robotic and Automation in Space (i-SAIRAS'99)*, pages 120–124, 01 1999.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[4] ONNX Runtime developers. Onnx runtime. `https://www.onnxruntime.ai`, 2024. Version 1.17.3.

[5] Yonghao Du, Tao Wang, Bin Xin, Ling Wang, Yingguo Chen, and Lining Xing. A data-driven parallel

scheduling approach for multiple agile earth observation satellites. *IEEE Transactions on Evolutionary Computation*, 24(4):679–693, 2020.

[6] Sylvain Ducomman, Hadrien Cambazard, and Bernard Penz. Alternative filtering for the weighted circuit constraint: Comparing lower bounds for the tsp and solving tsptw. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016.

[7] Ander Garcia, Olatz Arbelaitz, Pieter Vansteenwegen, Wouter Souffriau, and Maria Teresa Linaza. Hybrid approach for the public transportation time dependent orienteering problem with time windows. In Emilio Corchado, Manuel Graña Romay, and Alexandre Manhaes Savio, editors, *Hybrid Artificial Intelligence Systems*, pages 151–158, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[9] Lei He, Xiaolu Liu, Gilbert Laporte, Yingwu Chen, and Yingguo Chen. An improved adaptive large neighborhood search algorithm for multiple agile satellites scheduling. *Computers & Operations Research*, 100:12–25, 2018.

[10] Lei He, Mathijs Weerdt, and Neil Yorke-Smith. Time/sequence-dependent scheduling: the design and evaluation of a general purpose tabu-based adaptive large neighbourhood search algorithm. *Journal of Intelligent Manufacturing*, 31(4):1051–1078, 2020.

[11] Jason T. Lam, François Rivest, and Jean Berger. Deep reinforcement learning for multi-satellite collection scheduling. In Carlos Martín-Vide, Geoffrey Pond, and Miguel A. Vega-Rodríguez, editors, *Theory and Practice of Natural Computing*, pages 184–196, Cham, 2019. Springer International Publishing.

[12] Xiaolu Liu, Gilbert Laporte, Yingwu Chen, and Renjie He. An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. *Computers & Operations Research*, 86:41–53, 2017.

[13] Michele Lombardi, Michela Milano, and Andrea Bartolini. Empirical decision model learning. *Artificial Intelligence*, 244, 01 2016.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[15] Guansheng Peng, Guopeng Song, Yongming He, Jing Yu, Shang Xiang, Lining Xing, and Pieter Vansteenwegen. Solving the agile earth observation satellite scheduling problem with time-dependent transition times. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(3):1614–1625, 2022.

[16] Guansheng Peng, Guopeng Song, Lining Xing, Aldy Gunawan, and Pieter Vansteenwegen. An exact algorithm for agile earth observation satellite scheduling with time-dependent profits. *Computers & Operations Research*, 120:104946, 03 2020.

[17] David Pisinger and Stefan Ropke. Large neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, pages 99–127, Cham, 2019. Springer International Publishing.

[18] Cédric Pralet. Iterated maximum large neighborhood search for the traveling salesman problem with time windows and its time-dependent version. *Computers & Operations Research*, 150:106078, 2023.

[19] M.W.P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[20] Verena Schmid and Jan Fabian Ehmke. An effective large neighborhood search for the team orienteering problem with time windows. In Tolga Bektaş, Stefano Coniglio, Antonio Martinez-Sykora, and Stefan Voß, editors, *Computational Logistics*, pages 3–18, Cham, 2017. Springer International Publishing.

[21] Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *J. ACM*, 43(2):193–224, March 1996.

[22] Samuel Squillaci, Cédric Pralet, and Stéphanie Roussel. Comparison of time-dependent and time-independent scheduling approaches for a constellation of earth observing satellites. In *13th International Workshop on Planning and Scheduling for Space (IWPSS)*, 2023.

[23] Cédric Verbeeck, Pieter Vansteenwegen, and El-Houssaine Aghezzaf. The time-dependent orienteering problem with time windows: a fast ant colony system. *Annals of Operations Research*, 254(1):481–505, July 2017.

[24] Xinwei Wang, Guohua Wu, Lining Xing, and Witold Pedrycz. Agile earth observation satellite scheduling over 20 years: Formulations, methods, and future directions. *IEEE Systems Journal*, 15(3):3881–3892, 2021.