Earth Observation Satellite Scheduling with Graph Neural Networks

Antoine Jacquet
Jolibrain
Toulouse, France

Guillaume Infantes
Jolibrain
Toulouse, France

Nicolas Meuleau
Jolibrain
Toulouse, France

Emmanuel Benazera
Jolibrain
Toulouse, France

Stéphanie Roussel
Onera
Université de Toulouse
Toulouse, France

Vincent Baudoui
Airbus DS
Toulouse, France

Jonathan Guerra
Airbus DS
Toulouse, France

Abstract

The Earth Observation Satellite Planning (EOSP) is a difficult optimization problem with considerable practical interest. A set of requested observations must be scheduled on an agile Earth observation satellite while respecting constraints on their visibility window, as well as maneuver constraints that impose varying delays between successive observations. In addition, the problem is largely oversubscribed: there are much more candidate observations than what can possibly be achieved. Therefore, one must select the set of observations that will be performed while maximizing their weighted cumulative benefit, and propose a feasible schedule for these observations. As previous work mostly focused on heuristic and iterative search algorithms, this paper presents a new technique for selecting and scheduling observations based on Graph Neural Networks (GNNs) and Deep Reinforcement Learning (DRL). GNNs are used to extract relevant information from the graphs representing instances of the EOSP, and DRL drives the search for optimal schedules. Our simulations show that it is able to learn on small problem instances and generalize to larger real-world instances, with very competitive performance compared to traditional approaches.

1 Introduction

An Earth observation satellite (EOS) must acquire photographs of various locations on the surface of Earth to satisfy user requests. An *agile* EOS has degrees of freedom allowing it to target locations that are not exactly at its vertical in an earth-bound referential ("nadir"). The satellite we consider is in low orbit; as a consequence, each observation is available in a visible time window (VTW) that is significantly larger than its acquisition duration. Maneuvering the satellite between two observations consists of modifying its pitch, yawn and roll triple—also called its *attitude*—and thus implies delays that depend on the start and end observation, as well as on the time of the transition [18]. In addition, an agile EOS is typically oversubscribed: there is more observations to be performed that can possibly be achieved in the given time of operation. As different acquisitions may be associated with different priorities or utilities, the Earth observation satellite planning problem (EOSP) consists in selecting a set of acquisitions that maximize their weighted cumulative values, and designing a schedule for acquiring these observations while respecting the operational constraints.

The most complex instances of the EOSP involve several satellites orbiting Earth over multiple orbits, and dependencies between targets [22]. For instance, an acquisition may consist of several pictures of the same earth-bound location to be taken by different satellites, and/or in different

17th European Workshop on Reinforcement Learning (EWRL 2024).

time-windows [19]. In this paper, we limit our study to single-satellite, single-orbit, and single-shot problems, where there is only one satellite to control over a single orbit, and each acquisition is made of a single picture to be taken in its given VTW. Nevertheless, the problem is NP-complete [13], and there is no practical solution to compute optimal schedules for problems of realistic size which contain a few thousands of candidate acquisitions, thus focusing previous work towards approximate, heuristic and random search algorithms [22]. Variants of the greedy randomized adaptive search procedure (GRASP) [6] are commonly deployed in practical applications. At the same time, the field of combinatorial optimization is currently the subject of an accrued interest from researchers in deep learning. In particular, Deep Reinforcement Learning (DRL) offers a framework to learn heuristics for NP-complete problems that has been successfully applied to a wide range of problems [24]. Following this trend, we build on previous work using a state-of-the-art combination of graph neural networks (GNN) and DRL. This approach code-named *Wheatley* was developed to address Job Shop Scheduling Problems with duration uncertainty [11]. Here we adapt it to the EOSP and prove its efficiency in solving deterministic but largely over-subscribed problems. Our simulation results show that we outperform the currently deployed techniques.

Our main contributions are threefold: (1) propose a graph search representation of the problem without discretizing time; (2) use deep reinforcement learning to solve the problem; (3) use directly graph representation as observations thanks to graph neural networks. The main outcomes are: (1) very competitive results compared to baselines; (2) good generalization abilities allowing to train on small instances an solve efficiently large instances.

The paper is organized as follows. First we give a quick survey of previous work using deep learning to solve the EOSP. Then, in Section 3 ,we introduce the problem and discuss various representations used in this work. Section 4 is dedicated to the description of the machine learning architecture used for optimization. We provide simulation results on large size real-world instances of the problem in Section 5. We conclude and discuss further research directions in Section 6.

2 Related Work

The EOSP has been subject to a large body of research, from communities as varied as aerospace and engineering, operational research, computer science, remote sensing and multipliscinary sciences. We refer the reader to [22] for a survey of non-machine learning approaches to the problem, and we focus our attention on DRL based approaches to the EOSP. Note that we address the time-dependent EOSP, where the duration of a transition between two observations varies with time. This contrasts with most of previous literature that assumes constant, time-independent transition duration.

Peng et al. [14] address a slightly different problem where observations are scheduled on board. A LSTM-based encoding network is used to extract features, and a classification network is used to make the a decision. Dalin et al. [4] solve multi-satellite instances by modeling them as a Multi-Agent Markov Decision Processes, then use a DRL actor-critic architecture. The actor is decentralized, each satellite using a relatively shallow network to select its action. The critic is centralized and implemented as a large recurrent network taking input from all satellites. Hermann et al. [9] also address the multi-satellite problem: a policy is trained in a single satellite environment on a fixed number of imaging targets, and then deployed in a multi-satellite scenario where each spacecraft has its own list of imaging targets. Local policies are learned using a combination of Monte Carlo Tree Search (MCTS) to produce trajectories, and supervised learning to learn Q-values using the trajectories produced by MCTS as training examples. Finally, Chun et al. [3] present a very similar approach; the main difference is that the transitions durations approximated during training phase instead of being precisely computed on discrete dates before training.

3 Problem Representation

An instance of the EOSP is defined by a set of candidate observations, or acquisitions, and a time-horizon τ (in this work, the duration of an orbit). Each observation i is associated with its fixed duration d_i and its VTW $[e_i; l_i]$ such that $l_i \leq \tau$. The transition duration between two acquisitions i and j is a function $\Delta_{i,j}(t_i)$ of the starting time t_i of the first observation. A schedule is a sequence of observations with associated starting time. It is represented as a single mapping that associates with each candidate observation i its starting time t_i, such that $t_i = -1$ for all observations i *not* included

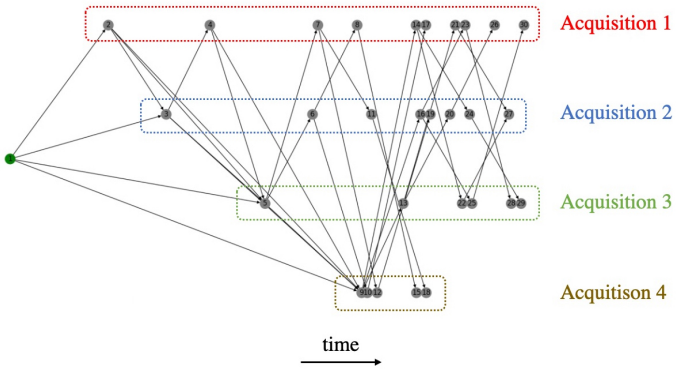

Figure 1: Discrete graph for 4 candidate acquisitions. Corresp. continuous graph is a 4-node clique.

in the schedule. A schedule is feasible if: (i) each scheduled observation starts and ends within its VTW: $t_i \in [e_i; l_i - d_i]$;(ii) the time-gap between two successive observations is greater or equal to the transition delay: $\forall i, j$ such that $t_i \neq -1$, $t_j \neq -1$ and $t_i \leq t_j$ then $t_j - t_i \geq d_i + \Delta_{i,j}(t_i)$

Each observation is associated with a utility value u_i in $\mathbb{R}^+$. The goal is then to find a feasible schedule that maximizes the cumulative utility of the observations it includes: $maximize \left(\sum_{i:t_i \neq -1} u_i\right)$.

3.1 Classical Approach : Time Discretization

In the EOSP, the start time t_i of each observation is a continuous value, and the transition durations $\Delta_{i,j}(t_i)$ are continuous functions of continuous variables. Therefore, the EOSP is not a completely discrete problem. However, it is often re-framed as such, either by making assumptions on the start time of transitions (for instance, every transition starts as soon as it is available), or by crudely discretizing the time variable.

In this work, the problem is provided under the form a very large time-discretized graph that represents the EOSP as a sequential decision problem. In this graph—later called the "discrete graph"—every candidate acquisition is represented once for each possible (discrete) start time. A node is typically denoted (i, t_i) and represents starting observation i at time $t_i \in [e_i, l_i - d_i]$. An edge between two nodes is present if the end observation is a possible immediate successor of the start observation. The acquisition and transition durations are accounted for while defining the edges: an edge between (i, t_i) and (j, t_j) is such that t_j is the smallest discrete time satisfying $t_i + d_i + \Delta_{ij}(t_i) \leq t_j$ and $t_j \geq e_j$. Note that there is an (implicit) mutual exclusion between two nodes (i, t_i) and (i, t'_i), $t_i \neq t'_i$, to indicate that observation i must not be performed twice. Finally, some edges are pruned using considerations of optimality: if an observation k can be inserted between (i, t_i) and (j, t_j) without breaking the constraints of the problem (that is, the transition delays), then the edge between (i, t_i) and (j, t_j) is removed. In this case, every path between the two nodes must go through one node (k, t_k). The reasoning is that every *optimal* solution that includes both (i, t_i) and (j, t_j) would also include observation k. Therefore, (j, t_j) should not be an *immediate* successor of (i, t_i). Figure 1 provides an example of discrete graph for a problem containing 4 candidate acquisitions.

The discrete graph is convenient for a typical state-space approach such as using an implicit enumeration algorithm (Dikstra, A*). A schedule can be built by starting from a node representing the origin of the graph and following the edges to add observations in chronological order (the earliest are added to the scheduled before the latest). Let m denote the last node scheduled. When a new node $n = (i, t_i)$ is scheduled, the graph is updated in the following way: (1) all edges outgoing from node m are removed, except the one leading to the newly scheduled node n; (2) all other nodes candidate for observation i (nodes (i, t'_i) with $t'i \neq t_i$) are deleted; (3) all nodes that have become unreachable from the origin node are removed.

3.2 Our Approach : Continuous-Time Graph

Although convenient for state-space approaches, the discrete graph has the drawback of quickly becoming huge as the number of candidate acquisition grows, making it unsuitable as an input to a GNN. For this reason, we derive from the discrete graph a (much) more compact graph that we call the "continuous graph". In this graph, each candidate observation is represented exactly once, with no mention of its exact starting time. A node is simply defined by a candidate observation i. An edge (i, j) is present in the continuous graph if and only if there is an edge $((i, t_i), (j, t_j))$ in the discrete graph. Note that this may lead to two nodes pointing at each other, if the corresponding observations may be performed in any order.

At each iteration of our algorithm, the agent is fed with the compact continuous graph and selects the next acquisition to be inserted in the schedule. To track which candidate observations are disallowed by this decision, and what are the possible next actions, we implement this decision in the discrete graph by assuming the selected observation is started at its earliest possible (discrete) time. The discrete graph is updated as described above, and these changes are reflected in the continuous graph. The resulting continuous graph is fed to the GNN at the next iteration. At each step, the set of candidate acquisitions that can be added to the current schedule is the set of immediate successors of the last node scheduled in the discrete graph. The learning algorithm is responsible for selecting one particular acquisition among them, using the procedure detailed later.

As stated above, in this work, we rely on the discrete graph. We use it both for building the continuous graph, and as a simulator of the built strategy. Pre-building this discrete graph could be avoided by building directly the continuous graph from the problem data, and using a satellite simulator to get the attitudes and transition times on-the-fly, without significant change in our approach. At the time of writing, the discrete graph is precomputed by calling a closed-source proprietary satellite simulator, and there is no simple legal way to switch to the “on-the-fly” approach.

3.3 Sequential Decision Model

Our solution technique is based on representing the process of *building an optimal schedule* for an instance of EOSP as a reinforcement learning problem, then using DRL algorithms to solve it. Reinforcement learning is concerned with learning the solution of a *Markov Decision Process* (MDP), which is a discrete-time sequential decision model. An MDP is defined as a tuple (S, A, T, R) where S is a state space, A the action space, T the transition matrix, and R the reward function [15]. The definition of these elements flows directly from the representation of the EOSP as a discrete graph:

- A state $s \in S$ is a discrete EOSP graph as defined before. It can be either the initial graph where no task has been selected yet, or an intermediate graph where some nodes and edges have been selected to account for the fact that the beginning of the schedule has been decided;
- Given a state s, the set of available actions a is the set of all possible successors of the last selected node in s (thus leading to *chronological insertions*);
- MDPs naturally handle uncertainty in the problem. I the general case, it is represented in the transition matrix: $T(s, a, s') = \Pr(s(t+1) = s' \mid s(t) = s, a(t) = a)$. However, our formulation of the EOSP is deterministic, therefore, the MDP we derive contains no uncertainty. Given an initial state s (discrete graph) and selected action a (the next observation to add to the schedule), the transition matrix gives probability 1 to the state s' representing the discrete graph after the addition of a to the schedule, following the update procedure described in the previous section.
- As every inserted observation is feasible by definition, we use as immediate reward the utility value associated with the selected observation ie $R(s, a, s') = u_a$. The aim is to maximize the undiscounted sum of immediate rewards.

While these components define a fully observable MDP, we do not provide a perfect description of the state s to the deep network. As explained above, we do not feed the huge discrete graph to the neural network. Instead, we use a continuous graph that is not a perfectly accurate representation of the problem, because transition duration are not reported exactly (see section 4.2). Therefore, the learner solves partially-observable MDP (POMDP) [12], where partial observability concerns only the transition durations, and thus plays a minor role.

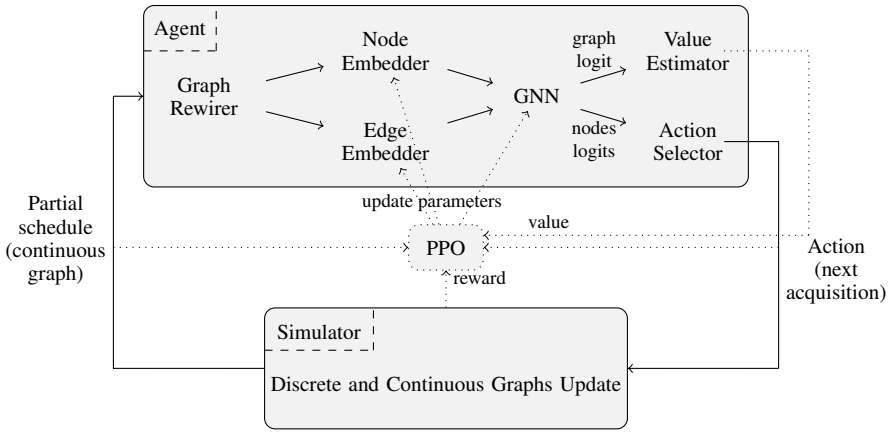

Figure 2: General Architecture

4 Solution

Following [11], we use a reinforcement learning setup where the agent receives continuous graphs representing partial schedules as input, selects the next observations to schedule, and updates its parameters based on the reward representing the cumulative utility of the schedules it produces. For a given problem, a simulator is in charge of managing the different graphs and feeding the learner with the appropriate data. The learner implements a policy, that is, a stochastic mapping from states s to actions a as defined above. It learns a policy that maximizes the reward function over a base of real-world problems used as training set. The policy has to be able to generalize to test problems, that is, exhibit good performances without further learning on a set of instances not seen before.

An overview of the architecture is shown in Fig. 2. The graphs provided as input are processed by several elements. First, the graph is transformed in order to allow bidirectional message-passing by the GNN, then simple networks produce node and edge embeddings. Next the graph is processed through a Message-Passing Graph Neural Network (MP-GNN) to extract features capturing relevant information, and produce action probabilities. Finally, the RL algorithm updates the parameters of the whole system, embedders and GNN, based on the rewards received. For ease of presentation, we first discuss the RL algorithm, then the embedders and GNN.

4.1 Reinforcement Learning

As our core algorithm, we use the Proximal Policy Optimization (PPO) algorithm [16] with action masking [10].

A peculiar aspect of the EOSP instances we have to solve is that the utility of different acquisitions may vary by up to 8 degree of magnitude. In fact, acquisitions are grouped in 7 priority classes with utility value ranging from 1 to 10^8. The utility of the observations within a class of priority is equal to the value of that class, plus a small term depending on the predicted cloud coverage at the location of the acquisition (in order to favor acquisitions that are likely to happen with a clear sky). This generates instability in DRL algorithms (and in MDPs in general), as the low priority observations provide a reward that might be difficult to distinguish form the noise in the algorithm. In addition, the critic must learn very large values, starting from very low values at initialization, and following tiny gradient steps. This makes learning slow and inefficient.

We tried several approaches to handle this, including using a logarithmic scale and 2-hot encoding [7]. In our current implementation, we simply divide each individual reward by the average utility of all candidates observations in the problem. This is a simple way to remedy the issue of having to learn very large values, but it does not fix the problem of the discrepancy between rewards (unless some extreme priority classes are not represented in the problem instance). We are currently examining optimization with lexicographic preferences [17].

4.2 GNN Implementation

Node attributes: To inform the learner, we label each node of the continuous graph with the window of visibility of the corresponding observation. We note that the continuous graph still does not contain any information about the transitions duration, so the learner would be blind to them. To compensate for this, each node i of the continuous graph is labeled with information about the satellite attitude while performing observation i, namely, the min, max and average pitch and roll of the satellite over the observation VTW. Although this information is not sufficient to recover the exact duration of transitions, it allows the learner inferring them closely enough to perform well, as shown in our simulation results.

Graph rewiring: A Message-Passing Graph Neural Network (MP-GNN) [23] uses a graph structure as a computational lattice. It propagates information, represented as messages, along the oriented graph edges only. In our case, if an MP-GNN uses only the EOSP continuous graph edges, then we explicitly forbid information to flow from future acquisitions to the present choice of the next acquisition. This is definitely not what we want: we want the agent to choose the next observation to schedule based on its effect on the on future conflicts. In other words, we want information to go from future to present tasks. Therefore, we have to edit the input graph before it can used by the MP-GNN. This is known in the MP-GNN literature as "graph rewiring".

For every (precedence) edge in the continuous graph, a link pointing in the other direction is added to the rewired graph (reverse-precedence). Different edge types are defined for precedence and reverse-precedence edges, to enable the learned operators to differentiate between chronological and reverse-chronological links. The systems learns to pass information in a forward and backward way, depending on what is found useful during learning.

Embeddings: A graph embedder builds the rewired graph by adding edges. It embeds node attributes (VTW, attitude stats) using a learnable MLP, and edge attributes (type of edge) using a learnable embedding. The output dimension of embeddings is an open hyper-parameter *hidden_dim*. We found a size of 64 being good in our experiments.

Graph pooling: A node is added and connected to every other node to allow collecting global information about the entire graph, as opposed to the local information associated with the nodes of the original graph. It is used by the critic to estimate the value of the graph as a whole. It is also used by the actor, where the global graph encoding is concatenated to each node embedding. Indeed, messages are passed by the MP-GNN algorithm only between immediate neighbors. Therefore, a network of depth n_layers is able to anticipate only n_layers observations ahead. Having the global node embedding concatenated to each node embedding compensates for this, allowing the current decision to take into account the entire graph.[1]

GNN: As a message passing GNN, we use EGATConv from the DGL library [21], which enriches GATv2 graph convolutions [2] with edges attributes. We used 4 attention heads, leading to an output of size $4\times$*hidden_dim*. This dimension is reduced to *hidden_dim* using learnable MLPs, before being passed to next layer (in the spirit of feed-forward networks used in transformers [20]). The output of a layer can be summed with its input using residual connections [8]. For most of our experiments, we used 10 such layers. The message-passing GNN yields a value for every node, and a global value for the graph (from the graph pooling node).

Action selection: Action selection aims at computing action probabilities given the node values (logits) output by the GNN. We can either use the logits output from the last layer of the GNN, or use a concatenation of the logits output from every layer. We chose to concatenate the global graph logits of every layer, leading to a data size of $((n\_layers + 1)\times$*hidden_dim*$) \times 2$ per node, where *hidden_dim* is the dimension of the embeddings (see next section). This dimension is reduced to 1 using a learnable linear combination, that is, a minimal case of a Multi-Layer Perceptron (MLP). We did not find using a larger MLP to be useful. Finally, a distribution is built upon these logits by normalizing them, and using action masks to remove actions that are not feasible in the current state. As node numbers correspond to action/acquisition numbers, we directly have the action identifier when drawing a value from the distribution.

[1] Adding such kind of node to the graph is equivalent to learning a custom graph pooling operator.

Dealing with different problem sizes: The GNN outputs a logit per node, and there is a one-to-one mapping between nodes and actions whatever the number of nodes/actions. Learning the best action boils down to node regression, with target values being given by the reinforcement learning loop. Internally, the message passing scheme collects messages from all neighbors, making the whole pipeline agnostic to the number of nodes.

Connecting to PPO: In most generic PPO implementation, the actor (policy) consists of a feature extractor whose structure depends on the data type of the observation, followed by a MLP whose output dimension matches the number of actions. The same holds for the critic (value estimator), with the difference that the output dimension is 1. Some layers can be shared (the feature extractor and first layers of the MLPs). In our case, we do not want to use such a generic structure because we have a one-to-one matching from the number of nodes to the actions. We thus always keep the number of nodes as a dimension of the data tensors.

5 Experiments

We use a set of real-life problems provided by an industrial partner who owns Earth observation satellites. As explained in Section 3, problems are given in the form of a discrete-time graph. The simulator uses this graph to compute and maintain the continuous-time graph. To provide intuition on the difficulty of the problem, Table 1 shows some statistics on a few test problems and their representation as graphs.

	# Nodes			# Edges		
# Acquisitions	Discrete graph	Continuous graph	Ratio	Discrete	Continuous	Ratio
106	10297	106	97	835566	9273	90
308	52020	308	169	12598738	81244	155
508	46589	508	92	14842035	225398	66
809	59583	809	74	28015753	447945	63
1074	94071	1074	88	58343397	741634	79

Table 1: Exemple problem sizes

We compare our DRL approach to two solutions currently being used for operating such satellites.

Greedy algorithm: It is the algorithm that is currently used for real-world operations. It greedily selects acquisitions to add to the schedule based on the utility, and inserts them in the plan if possible. Previously selected tasks may be slightly postponed, but never canceled.

RAMP: [1] It is an implementation of a Dijkstra search algorithm in the discrete graph. Although based on an admissible algorithm (Dijkstra), RAMP is not guaranteed to find the absolute optimal schedule. This is due to the exclusion links between nodes of the discrete graph that represents the same task started at different times. Nevertheless, RAMP constantly provides the best known schedules on real problems. Unfortunately, its complexity prevents using it for real-time operations. Therefore, it is used as a reference in these simulation results.

5.1 Unitary Score

First, we compare our approach to baselines on a relaxed problem: we try to maximize the number of acquisitions scheduled, irrespectively of their priority or utility. This measure of performance is insensitive to the large gaps in acquisitions utility discussed in Section 4.1. We run two experiments:

Single problem: First, we want to measure if our models and algorithms can possibly achieve competitive performance on a given problem. We train our learner on a single problem with a total of 106 acquisitions and let it overfit as much as it needs, as long as it achieves great performance. As illustrated in Fig. 3-left, we observe that it is indeed able to beat both the greedy algorithm and RAMP scores. This result shows that our architecture is able to implement very powerful policies. In the next set of experiments, we put it to the challenge of realistic learning environment.

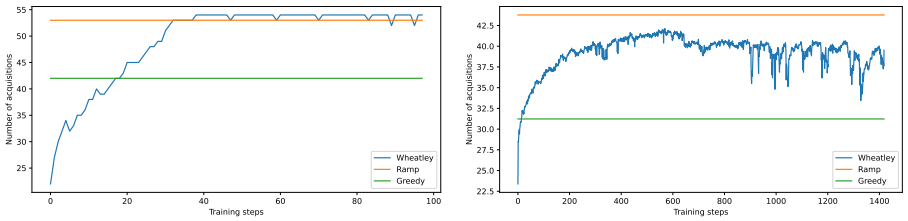

Figure 3: Unitary scores. Left: Training on a single problem of 106 acquisitions. Right: Test performance on 31 unseen problems, after training on 128 different problems.

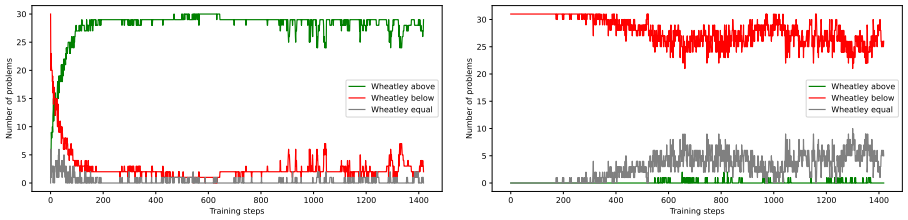

Figure 4: Unitary scores for Wheatley vs. Greedy (left) and vs; Ramp (right) on 31 test problems.

Generalization: To measure the ability to transfer knowledge from one task to another, we train on 128 problems of about 100 acquisitions and test on 31 unseen problems of similar size. The learning curve of Fig. 3-right shows the evolution of the performance on the test set, as learning progresses. We also measure the number of times where Wheatley's performances are above, below or equal to the greedy algorithm and RAMP (Fig. 4) on the test problems. This shows that our system is able to generalize to unseen problems, outperforming the currently deployed solution.

5.2 General Utility

In our second sets of experiments, we take into account the utility of observations and aim at maximizing the cumulative utility of all the observations included in the final schedule, as in the full-fledged MDP framework presented in Section 7. As before, we perform two sets of experiments: one where the learner is free to overfit on a single problem to reach its best performances, and one aiming at measuring its ability to generalize.

Single Problem: Our test on single problem with a total of 88 candidate acquisitions shows that our system is able to outperform the greedy algorithm and reach the score of RAMP (Figure 5-left). This proves the suitability of the architecture for the full MDP set up.

Generalization: We train on 639 problems of about 100 acquisitions and test on 27 unseen problems of similar size. The learning curves are displayed in Figure 5-right and show that the learner is able to generalize. The plot showing the number of times where Wheatley is above, below or equal to the greedy solution and RAMP are presented in Fig. 6. We see that Wheatley outperforms the deployed solution and approaches the best known performances, in a realistic set-up where problems

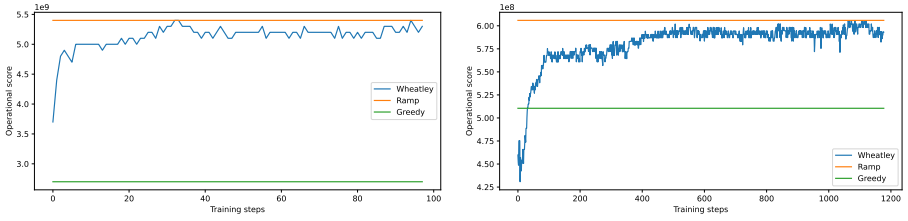

Figure 5: Utilities obtained when training on a single problem with 88 acquisitions (left) and averaged on 27 unseen problems, after a training on 639 different problems (right).

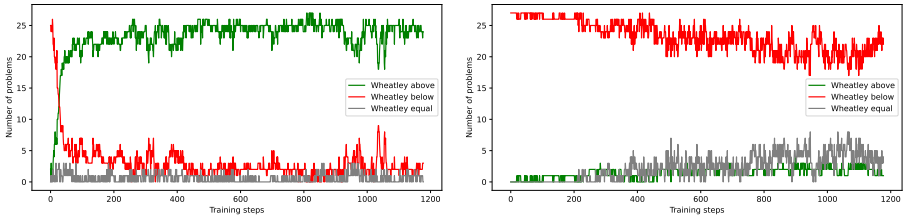

Figure 6: Utilities for Wheatley vs. Greedy (left) and vs. Ramp (right) on 27 test problems.

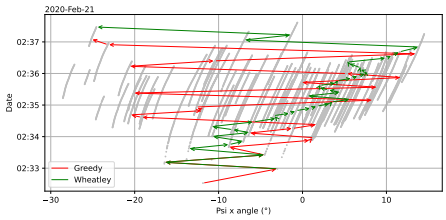

Figure 7: Trajectories found by Wheatley and Greedy approaches on a 87 acquisitions instance.

Instances Set		Average Utility Score			Avg. Scores Ratios	
#Acq.	#Instances	Wheatley	Greedy	Ramp	$\frac{\text{Wheatley}}{\text{Greedy}}$	$\frac{\text{Wheatley}}{\text{Ramp}}$
100	27	605,732,913	510,488,439	**605,894,711**	**1.1788**	0.9901
300	30	239,934,939	202,565,994	**263,514,132**	**1.2560**	0.9420
508	1	221,226	142,487	**308,355**	**1.5526**	0.7174
809	1	52,000,039	49,000,279	**59,000,286**	**1.0612**	0.8814
1074	1	2,910,000,156	2,225,103,224	**4,124,116,197**	**1.3078**	0.7056
1591	1	220,734	307,159	**393,214**	0.7186	0.5614
100	10	**689,578,101**	688,941,262	678,921,586	0.9996	**1.0115**

Table 2: Average scores obtained when generalizing on different instances sets.

are not known in advance. Fig. 7 shows examples of solutions produced by Greedy and Wheatley. We can see that Wheatley finds a smoother and more efficient trajectory for the satellite.

Table 2 shows comprehensive results for the agent trained on problems of size 100, evaluated on different sizes of problems with operational scores. The last line is an evaluation on instances with many conflicts, where RAMP performs worse than the greedy algorithm. Results show that Wheatley performs very well on not too large instances but is quite outperformed by the greedy approach on the largest instance. However, it is quite competitive in the case of highly conflictual instances, which is promising for future works.

6 Conclusion

We showed that DL-based approaches to the EOSP are challenging some of the best known techniques. There are several perspectives we are currently exploring to extend this work. First, as stated before, we are trying to take advantage of the large discrepancy in acquisition utility by using lexicographic RL algorithms such as [17]. Scheduling tasks by decreasing priority would provide stronger guarantees to find the optimal schedule. To achieve this, schedules must be built in a non-chronological order, which is not the case in our current implementation. Currently, we choose the next acquisition to insert just after the last inserted one, using some foresight given by the GNN. This foresight is limited by the number of layers of the GNN. As we said, the discrete-time graph is tailored for state-space search and chronological insertion. Future work will consider developing an alternative continuous-tine graph representation of the EOSP where observations can be added to the schedule in any order, using Simple Temporal Networks [5]. Such work will open promising avenues for using lexicographic preferences.

References

[1] Pierre Blanc-Pâques. (US. PATENT US10392133B2) Method for planning the acquisition of images of areas of the earth by a spacecraft, august 2019.

[2] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *CoRR*, abs/2105.14491, 2021.

[3] Jie Chun, Wenyuan Yang, Xiaolu Liu, Guohua Wu, Lei He, and Lining Xing. Deep reinforcement learning for the agile earth observation satellite scheduling problem. *Mathematics*, 11(19), 2023.

[4] Li Dalin, Wang Haijiao, Yang Zhen, Gu Yanfeng, and Shen Shi. An online distributed satellite cooperative observation scheduling algorithm based on multiagent deep reinforcement learning. *IEEE Geoscience and Remote Sensing Letters*, 18(11):1901–1905, 2021.

[5] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1):61–95, 1991.

[6] T.A. Feo and M.G.C Resende. Greedy randomized adaptive search procedures. *Journal of Globlal Optimization*, 6:109–133, 1995.

[7] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[9] Adam Herrmann and Hanspeter Schaub. Reinforcement learning for the agile earth-observing satellite scheduling problem. *IEEE Transactions on Aerospace and Electronic Systems*, PP:1–13, 01 2023.

[10] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *The International FLAIRS Conference Proceedings*, 35, may 2022.

[11] Guillaume Infantes, Stéphanie Roussel, Pierre Pereira, Antoine Jacquet, and Emmanuel Benazera. Learning to solve job shop scheduling under uncertainty. In *21th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2024.

[12] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[13] Michel Lemaître, Gérard Verfaillie, Frank Jouhaud, Jean-Michel Lachiver, and Nicolas Bataille. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, 6(5):367–381, 2002.

[14] Shuang Peng, Hao Chen, Chun Du, Jun Li, and Ning Jing. Onboard observation task planning for an autonomous earth observation satellite using long short-term memory. *IEEE Access*, 6:65118–65129, 2018.

[15] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[17] Joar Skalse, Lewis Hammond, Charlie Griffin, and Alessandro Abate. Lexicographic multi-objective reinforcement learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, IJCAI-2022. International Joint Conferences on Artificial Intelligence Organization, July 2022.

[18] Samuel Squillaci, Cédric Pralet, and Stéphanie Roussel. Comparison of time-dependent and time-independent scheduling approaches for a constellation of earth observing satellites. In *Proceedings of the Thirteenth International Workshop on Planning and Scheduling for Space*, pages 96–104, 2023.

[19] Samuel Squillaci, Cédric Pralet, and Stéphanie Roussel. Scheduling complex observation requests for a constellation of satellites: Large neighborhood search approaches. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 443–459. Springer, 2023.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[21] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J. Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *CoRR*, abs/1909.01315, 2019.

[22] 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.

[23] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[24] Xinyi Yang, Ziyi Wang, Hengxi Zhang, Nan Ma, Ning Yang, Hualin Liu, Haifeng Zhang, and Lei Yang. A review: Machine learning for combinatorial optimization problems in energy areas. *Algorithms*, 15(6), 2022.