Towards Ensemble AI for Satellite Plan Execution (Abstract)

Gonzalo Montesino Valle[1] **and Michael Cashmore**[2]
University of Strathclyde, Glasgow, Scotland
[1]*gonzalo.montesino-valle@strath.ac.uk*, [2]*michael.cashmore@strath.ac.uk*

Abstract

The execution of plans onboard satellites is a challenging process due to the inherently non-deterministic and stochastic environment. We define an execution agent as the system responsible of dispatching and monitoring the actions of a plan while guaranteeing real-time response. In this abstract we propose an execution agent that is able to execute the a number of plans in parallel, managing a dynamic priority between them. This method utilises the idea of *ensemble AI*, which takes input from multiple voices, such that each voice has its own goal specific interest, and with the help of a human control variable or function to manage priority. Finally an arbiter agent chooses an optimal action to take at each moment. Our results show that the ensemble execution agent is able to scale to much larger sets of mission objectives when compared to a system that attempts to merge these objectives before planning and execution.

1 Introduction

Satellite mission planning involves the generation of plans or training of agents to pursue objectives in an uncertain environment. During execution planned actions may fail and the execution agent must react to unexpected outcomes. Plan execution algorithms deal with this challenge in a variety of ways (Wang et al. 2020; Lima et al. 2020; Coruhlu, Erdem, and Patoglu 2021). Policies, for example trained through reinforcement learning can also naturally adapt to changing circumstances (Baker et al. 2019, 2022).

Satellite mission planning may also involve multiple missions to be carried out in parallel over a rolling horizon of decision making, often with *over-subscription*: more objectives that can be reasonably carried out within the time constraints. In addition, changing situations can alter priorities between objectives during the mission's execution. For example, after some data is dumped to a ground station a mission is re-prioritised as highly desirable, or an unexpected phenomenon occurs such as a micro-asteroid collision or solar storm that necessitates switching to a safety mode. These ideas bring additional challenges in prioritisation between objectives, and fairness between mission providers. While existing plan execution approaches can be used to respond to uncertainty, they do not deliberate over this balancing of multiple objective sets under changing prioritisation.

In this paper we investigate a new approach to allow multiple agents control a single asset in a non-deterministic environment without intervention. To validate the approach we have implemented an execution agent based on *ensemble decision making* and experimented in a satellite simulation with the following aims.

- We aim to show that this approach is able to balance priorities between multiple mission providers during execution, while still achieving an acceptable proportion of the overall mission goals.
- We also aim to show that in some cases, allowing multiple agents to plan or train separately, and then executing those plans in parallel can aid scalability.
- Finally that allowing those agents to plan or train independently on the ground, and only combining their results during execution allows more flexibility in their approach - i.e. learned policies can be followed alongside human-scripted plans without the need for off-board reasoning. This allows for the simple combination of human prescribed missions alongside automatic safety and recovery procedures.

This paper consequently presents two contributions: an architecture that implements our proposed approach and a light-weight satellite mission simulation that can be used to test and benchmark mission execution in a simplified satellite environment.

The execution agent architecture is based on the idea of *ensemble decision making* (Rodgers, Levine, and Anderson 2018) and evaluated using the light-weight simulation. This agent uses ensemble reasoning to simultaneously deconflict and execute the commands from a set of agents with associated priority. This priority is set dynamically - meaning that it can change based on the state of the mission, current time, or other condition.

In our architecture the agents responsible for individual missions are abstracted - they may be scripted plans written by hand, plan execution algorithms capable of replanning online, or policies learned via reinforcement learning, or something else. We refer to these agents as *voices* throughout this paper. Our architecture describes how these voices are configured and fed into an *arbiter* agent that is responsible for deciding the commands to be carried out.

The paper is structured as follows: in section 2 we discuss the related work on which our architecture is based. The architecture is then described in detail in section 3. The

This is a peer-reviewed, accepted author manuscript of the following paper: Valle, G. M., & Cashmore, M. (Accepted/In press). Towards ensemble AI behaviours for satellite plan execution. Paper presented at 73rd International Astronautical Congress 2022, Paris, France.

light-weight simulation is presented in section 4, followed by a description of our experimental setup in section 5. We discuss the results and conclude in sections 6 and 7.

2 Related Work

In this section we briefly discuss other executives that have been used for planning and executing missions in space. Then we discuss related work in ensemble decision making that motivates our approach.

The Multi-mission Executive (MEXEC) algorithm (Hughes et al. 2020) uses a task network to create and execute a conflict-free schedule to achieve a set of goals, to do so it is divided into 3 different modules: one in charge of planning, one for executing and a timeline library used by the planning module to slot tasks in free slots. The LILA algorithm (Saint-guillain, Vaquero, and Chien 2021) is used to solve the problem of plan execution for Probabilistic Simple Temporal Network (PSTN) based problems with Monte Carlo Tree search. LILA uses the idea of "playing a game against nature" in which the MCTS nodes are divided into decision nodes and contingency nodes. The contingency nodes simulate the randomness of the PSTN times, whereas the decision node set the starting time of each action. LILA is effective at executing plans that have been made robust to uncertain temporal durations, but requires another system to generate the input PSTN. Finally, Gillette and George (2022) use the Schedule Management framework, which generates and executes schedules of highly constricted tasks, and the the cFS apps to create the CSM framework for satellite plan generation and execution.

In Rodgers, Levine and Anderson (2018) present a novel system using ensemble AI create a decision-making algorithm able to play *Ms. Pac-Man*. Their architecture consists of four voices each with a clear objective. The output of these voices is then compared and an action is executed depending upon a set of state-based rules. Anderson et al. (2019) devised an updated algorithm able to play general video games, with voices based on heuristics that were previously designed for general game search.

Similar systems to ensemble AI have been also used for Reinforcement Learning (RL). Yang et al. (2020) merge three different deep RL algorithms to achieve a robust trading agent. The choice of action to execute is made using a custom performance metric. This approach was shown to out-preform each of the three voices independently. Unlike Yang et al. our motivation is not to create a more robust execution towards a single goal, but instead to merge multiple objectives in a balanced fashion. However, it is feasible that this kind of "adaptable robustness" could also be achieved using our architecture.

3 Architecture

In this section we describe the architecture of our execution agent, used to execute a satellite mission with different scientific teams. The system is divided into *voice components* and *arbiter*. The voices are first described relatively abstractly. The arbiter is then described in more detail, including the way in which priorities are associated with each voice.

Voice Component

As mentioned previously, we take an abstract definition of a voice in this architecture. A voice is considered to be an agent with its own goal, which takes as input the current state of the world and sends as output an action that is desirable to take. It could be noted that the architecture we describe in this section also fits the definition of a voice. While not essential, it is desirable voices should be reactive to the state and able to modify their course of action according to unforeseen circumstances. The reasoning for this is that as a sub-component and not the ultimate controller of the system, their action might not be selected for execution - as a result the environment from the perspective of a voice becomes non-deterministic.

In our implementation each voice is wrapped by a *voice wrapper* to allow it to speak with the rest of the system. Figure 1 illustrates the design of a voice wrapper, which is composed of two parts:

1. the agent in charge of choosing the action, and
2. an observation function that fetches the current state of the world and parses this state into the formalism required by the agent.

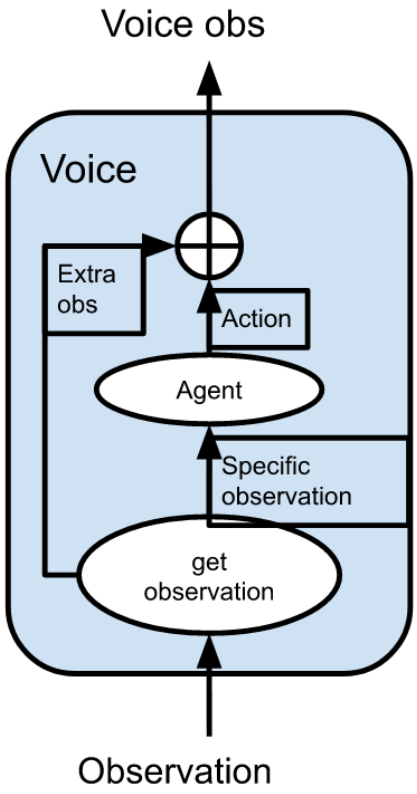

Figure 1: Voice Component

The parsing from general to specific observation allows different voices to operate upon different models of the world. For example, using different RL observation spaces, differing domain models for planning, and so on.

Arbiter Component

The arbiter is responsible for choosing the actions to be executed. As input the arbiter is given the action choices from each voice, the state observation, and an additional parameter vector called the *weight vector*.

The weight vector $\alpha \in \mathcal{R}^n$ is the parameter used to associate priority with each of n voices, i.e. such that voice v_i has priority α_i.

In figure 2, we show the design of the whole system consisting of arbiter and voices. The output of all voices are merged with the complete observation of the environment to obtain the full observation used by the arbiter. Finally, the full observation and the weight vector values are given to the arbiter. The arbiter then outputs an action to the environment. This may mean simply selecting the action according to the voice with highest priority, but could also be a more complex reasoning system that selects a compromising action that was not suggested by any voice. In this paper we propose two different algorithms that instantiate the arbiter component: the *priority arbiter* and the *weighted arbiter*.

Priority Arbiter This arbiter takes action of the voice with the highest possible α value that does not interfere with the future actions of higher-priority voices. The intuition is that this voice will greedily maximise achieving the goals of the high-priority voices, while still achieving the goals of lower priority voices as opportunistic objectives. This idea is explored in experimentation in Section 5.

Algorithm 1 Priority Arbiter

```
function PRIORITYACTIONSELECTION(A, α)
    Input: A set of actions
    Input: α set of priorities
    Sort: ⟨A, α⟩                          ▷ From highest α to lowest
    for all a_i, α_i in A, α do
        if ActionIsPossible(a_i) then
            if CheckInterference(a_i, A_hp) then
                return IdleAction
            else
                return a_i
            end if
        end if
    end for
end function
```

Weighted Arbiter this approach instead sums the priority associated with each *selected action*, allowing groups of lower-priority voices to override the decision of a high-priority voice. The intuition is that this kind of arbiter would maximise the priority-weighted sum of achieved objectives overall.

Algorithm 2 Weighted Arbiter

```
function PRIORITYACTIONSELECTION(A, α)
    Input: A set of actions
    Input: α set of priorities
    Init: w as array [0, ..., 0] of length |α|
    for all a_i, α_i in A, α do
        for all a_j, w_j in A, α do
            if a_i == a_j then
                w[i] ← w[i] + α_j
            end if
        end for
    end for
    return A[argmax(w)]
end function
```

4 Simulation

To carry out the experiments we implemented a lightweight simulation setup of a 2D earth observation environment. The main purpose of the simulation is to recreate the complexity of mission planning in real life while being simple to connect with any type of decision making algorithm. To achieve this purpose the environment is implemented as an *OpenAI Gym Environment* with normalized input and output.

The simulation achieves the complexity of mission planning by having characteristic earth observation constraints such as limited memory, action windows, and a tunable uncertainty level.

Figure 3 shows an example of the rendered simulation. On the top the set of boxes are the different memory slots which are in white if they are free, orange if there is a non-analyzed picture and purple if the picture is analyzed. Just below is a time bar that shows how much time is left in the execution of the current action. To the left is the type of action currently being executed where "DN" is "Do Nothing", "TP" is "Take a Picture", "AN" is "Analyze" and, "DP" is "Dump a Picture". Finally on the center of the image we see the 2D Earth and the satellite, as well as the targets which are shown in in orange and the ground stations which are shown in purple.

The simulation environment provides two different action spaces:

- **Simple or target blind action space**: this action space consists of four actions "Take a Picture", "Analyze", "Dump a Picture" and "Idle". These actions do not specify the target, which is instead chosen depending on the following rules:
  1. When "Take a Picture" is triggered a picture of the target currently below the satellite will be taken and if there is free space in the memory it will be stored as not analyzed.
  2. When "Analyze" action is triggered the latest non-analyzed picture will be the one analyzed. This action can be taken at any time and has an uncertain duration. There is a nonzero probability that the picture is not analyzed correctly. Both temporal and failure uncertainties can be adjusted as required.
  3. When "Dump a Picture" action is triggered the last pic-

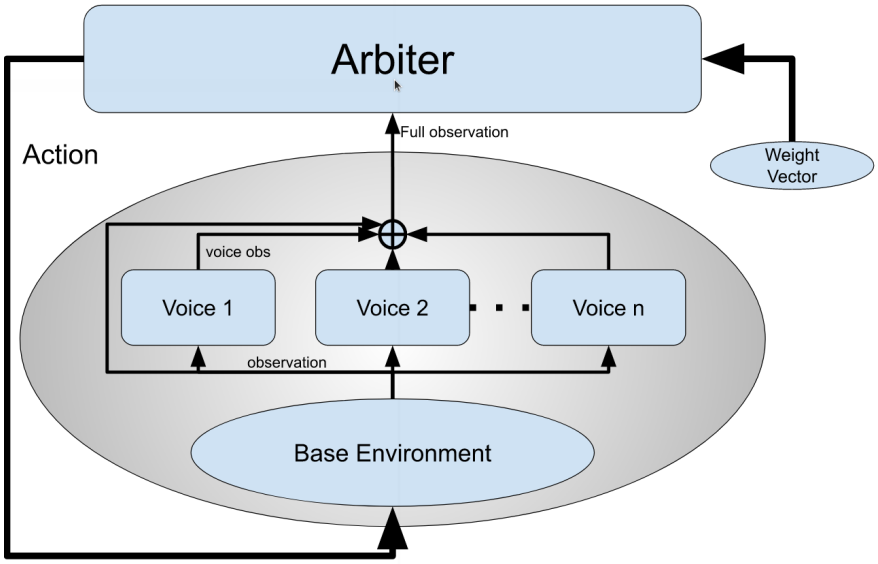

Figure 2: Arbiter architecture schematic

ture in memory to be analyzed is dumped. As done with the "Analyze" action, the probability of success can be adjusted.

- **Advanced or Selective action space**: In this case the action space is expanded from the previous one by specifying which target to take the picture or which picture to be analyze or dump. This action gives the agent flexibility on what to do but also increases the complexity of the planning problem.

The observation space consists of:

- the current angular position of the satellite,
- the current orbit count,
- all information about the current memory state (empty, unanalyzed picture, analyzed picture)
- angular location of the targets and ground stations,
- whether the satellite is busy or not.

The simulation allows different scenarios to be created by changing the number and size of the ground station and targets as well as the duration of each action. The simulation was used in our experiments described in the next section.

5 Experimental setup

Simulations were run twice for each random seed, one with the ensemble architecture controlling execution and once with a single planner and executor. The single plan and execution was a deterministic temporal planner (also used in the voices). To compare approaches the environment was increased in complexity in terms of number of targets, the number of objectives per scientific team, and maximum number of objectives.

The priority arbiter is used in all experiments. To see the influence of the weight vector, two setups are tested: a fixed setup for the whole mission, and changing the weight vector every five orbit to prioritise the voice that has achieved the fewest objectives so far. The latter setup is used to evaluate the ability of the approach to balance between voices.

The selection of the number of voices is a key aspect of the arbiter. For these experiments we will assume three different scientific teams have to use the same satellite. We also assume that in case the memory becomes full, pictures must be analyzed and dumped until half of the memory is empty. This results in a total of four voices, three voices representing scientific teams whose priority are set by α and one emergency voice that will meet the memory requirement.

The memory-emptying voice is included to account for the case that the other voices fill the memory and for some reason do not wish any images to be dumped. Ideally this kind of deadlock could be accounted for within a more sophisticated arbiter. Using the weighted and priority arbiters the deadlcok is possible. Including a deadlock-resolving voice is a simple solution that serves to illustrate the flexibility of the ensemble approach. Hypothetically, reasoning over the different plans of the voices should allow deadlocks to be avoided without losing as much efficiency.

Each of the voices is instantiated as a temporal planner. The planner generates an initial plan and only re-plans when the last action is dispatched. The planner is not reactive - meaning that the plan must be dispatched in full before new objectives can be considered. As previously mentioned, using deterministic executors such as these is not idea, and will harm the performance of the arbiter. However, they fulfill the minimum requirements for a voice and so serve to make a fair comparison against non-ensemble executor.

The single plan and execution agent used as a comparison merges the objectives of all teams into a single goal, and then attempts to plan and achieve the maximum number of objectives across all teams. Note that if two voices have the

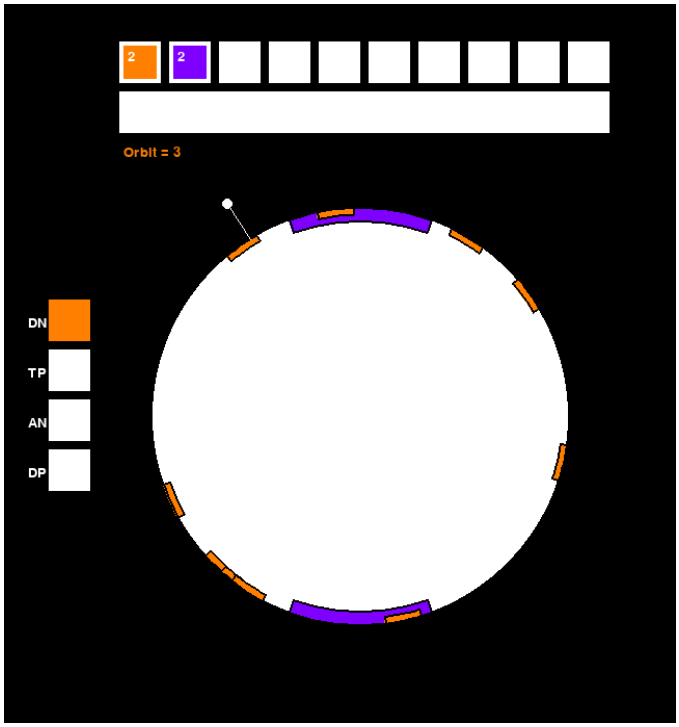

Figure 3: Rendered Simple Satellite Simulation example

same target as an objective, this will create only a single objective in the merged problem.

Table 1 shows simulation parameters.

nº of orbits (n_{orbit})	20
nº Ground Stations (n_{gs})	2
Size Ground Stations (s_{gs})	40
Angular Velocity (v)	0.6
Take Picture Duration (t_p)	20 s
Analyze Duration (t_a)	50 s
Dump Picture Duration	20 s

Table 1: Simulation parameters

6 Results

In figure 4, we compare the scalability of the ensemble architecture to the one of a single planner. The comparison is done by increasing the number of total targets and the number of objectives per voice. To better show the limits of the environment a dotted red lines illustrates the ideal performance if the environment was perfectly distributed so that no time could be wasted.

The results show that the planner does not scale to larger problem as it is not able to create plans for the higher complexity problems. The merged problem provides better results at lower complexities as can be seen by the performance of the planner matching the theoretical optimal. However, it provides no solutions to problems with more than 100 objectives. The ensemble architecture solves the scalability problem by dividing the problem into lower complexity ones. We can also see a limit performance of the arbiter being of a logarithmic type with high variance depending on the environment.

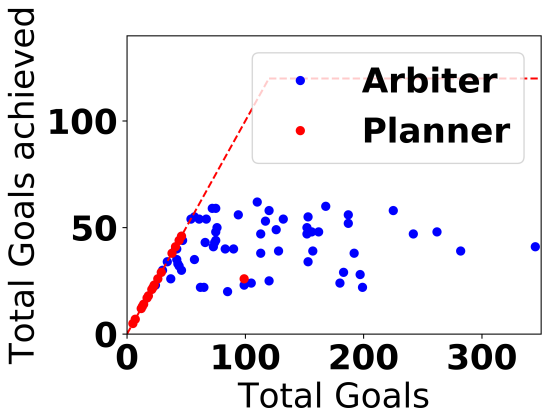

Figure 4: Comparison between goals achieved and total goals

Figures 5 and 6 show how changing the weight vector influences the specific actions of each voice. The two scenarios are a voice in which the priority is set at the beginning of the mission (Figure 5) and the second scenario the priority changes every 5 orbits to prioritise the voice with fewest goals achieved (Figure 6).

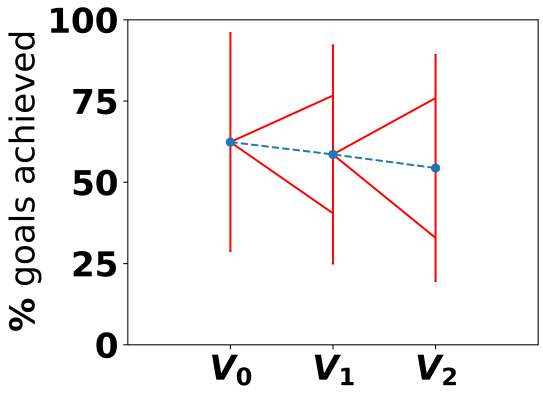

Figure 5: Percentage of goals achieved per voice, the red line between voices represent the standard deviation of the difference between percentage between the 2 adjacent voices. Mean while the blue line is the mean

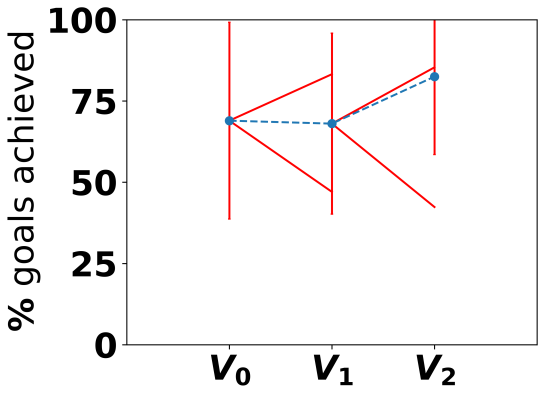

Figure 6: Percentage of goals achieved per voice, the red line between voices represent the standard deviation of the difference between percentage between the 2 adjacent voices. Mean while the blue line is the mean

In 5, the results of the fixed priority can be seen, in it there is a clear trend that complies with the priority levels. From figure 5 we can clearly see the priority level set at the beginning of the simulation are follow as a mean, although the error is high due to the lack purely reactive voices.

Meanwhile, figure 6 shows a better balance between the two first voices. However, the third voice receives a much higher overall reward than the other two on average. This is likely due to the limited and fixed number of orbits favouring a specific voice in the sequence, which could be verified in future experiments.

7 Conclusion

In the paper we proposed an online architecture for autonomous satellite decision making, by using ensemble AI. The main goal of the architecture is to be able to modify the behaviour if the satellite by changing a single parameter. We showed that the arbiter was able to match and out-preform a single planner and executor as complexity grew.

We showed that the behaviour of execution can be affected by adjusting the priority of voices online. Coupled with the ability to add additional simple voices to the system, this provides a low-effort way to manipulate the desired behaviour of the agent. Dynamic changing of priority can also be used to switch between different ”off-the-shelf” approaches depending upon their observed performance (Yang et al. 2020).

the arbiter architecture shows the advantage of been capable of using of the shelf algorithms for the different voices. However, ideally these algorithms should be reactive to actions having unexpected outcomes, as their action might not be the one selected by the arbiter. This type of constraint greatly match the benefits of reinforcement learning (RL) algorithms, which could highly increase the performance of the architecture. Future work will investigate using a mixture of rule-based and RL voices for robust execution.

References

Anderson, D.; Rodgers, P.; Levine, J.; Guerrero-Romero, C.; and Perez-Liebana, D. 2019. Ensemble decision systems for general video game playing. *IEEE Conference on Computatonal Intelligence and Games, CIG* 2019-Augus. ISSN 23254289. doi:10.1109/CIG.2019.8848089.

Baker, B.; Akkaya, I.; Zhokhov, P.; Huizinga, J.; Tang, J.; Ecoffet, A.; Houghton, B.; Sampedro, R.; and Clune, J. 2022. Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos URL http://arxiv.org/abs/2206.11795.

Baker, B.; Kanitscheider, I.; Markov, T.; Wu, Y.; Powell, G.; McGrew, B.; and Mordatch, I. 2019. Emergent Tool Use From Multi-Agent Autocurricula URL http://arxiv.org/abs/1909.07528.

Coruhlu, G.; Erdem, E.; and Patoglu, V. 2021. Explainable Robotic Plan Execution Monitoring Under Partial Observability. *IEEE Transactions on Robotics* 1–21. ISSN 1552-3098. doi:10.1109/tro.2021.3123840.

Gillette, A.; and George, A. 2022. Reusable Schedule Design and Execution Framework for Autonomous Mission Management in Space. *Journal of Aerospace Information Systems* 19(2): 154–165. ISSN 23273097. doi:10.2514/1.I010990.

Hughes, K.; Rothstein-Dowden, A.; Bocchino, R.; Donner, A.; Feather, M.; Smith, B.; Fesq, L.; Barker, B.; and Campuzano, B. 2020. Mexec : an Onboard Integrated Planning and Execution Approach (October).

Lima, O.; Cashmore, M.; Magazzeni, D.; Micheli, A.; and Ventura, R. 2020. Robust Plan Execution with Unexpected Observations URL http://arxiv.org/abs/2003.09401.

Rodgers, P.; Levine, J.; and Anderson, D. 2018. Ensemble Decision Making in Real-Time Games. *IEEE Conference on*

Computatonal Intelligence and Games, CIG 2018-August. ISSN 23254289. doi:10.1109/CIG.2018.8490401.

Saint-guillain, M.; Vaquero, T. S.; and Chien, S. A. 2021. Lila : Optimal Dispatching in Probabilistic Temporal Networks using Monte Carlo Tree Search .

Wang, D.; Russino, J. A.; Basich, C.; and Chien, S. A. 2020. Using Flexible Execution , Replanning , and Model Parameter Up- Dates To Address Environmental Uncertainty for a Planetary .

Yang, H.; Liu, X. Y.; Zhong, S.; and Walid, A. 2020. Deep reinforcement learning for automated stock trading: An ensemble strategy. *ICAIF 2020 - 1st ACM International Conference on AI in Finance* doi:10.1145/3383455.3422540.