Reinforcement Learning for the Agile Earth-Observing Satellite Scheduling Problem

ADAM HERRMANN
HANSPETER SCHAUB
University of Colorado Boulder, Boulder, CO USA

This work explores reinforcement learning (RL) for on-board planning and scheduling of an agile Earth-observing satellite (AEOS). In this formulation of the AEOS scheduling problem, a spacecraft in low Earth orbit attempts to maximize the weighted sum of targets collected and downlinked. The AEOS scheduling problem is formulated as a Markov decision process (MDP) where the number of upcoming imaging targets included in the action space is an adjustable parameter to account for clusters of imaging targets with varying priorities. Monte Carlo tree search (MCTS) and supervised learning are used to train a set of agents with varying numbers of targets in the action space. Two backup strategies are explored for MCTS—an incremental averaging operator and a maximization operator. For all backup operators, performance asymptotically increases as the number of targets in the action space approaches the maximum number of available targets. A benchmark is computed with MCTS to determine an upper bound on performance. Furthermore, MCTS is compared with solutions generated by a genetic algorithm. MCTS demonstrates a 2%–5% increase in average reward at 10%–20% of the single-core wall clock time of the genetic algorithm. A search of various neural network hyperparameters is presented, and the trained neural networks are shown to approximate the MCTS policy with three orders of magnitude less execution time. Finally, the trained agents and the genetic algorithm are deployed on varying target densities for comparison purposes and to demonstrate robustness to mission profiles outside of the training distribution.

Manuscript received 2 June 2022; revised 24 September 2022 and 26 December 2022; accepted 27 February 2023. Date of publication 2 March 2023; date of current version 11 October 2023.

DOI. No. 10.1109/TAES.2023.3251307

Refereeing of this contribution was handled by G. Inalhan.

This work was supported by NASA Space Technology Research Graduate Opportunity (NSTGRO) under Grant 80NSSC20K1162.

Authors' addresses: Adam Herrmann is with the Aerospace Engineering Sciences Department, University of Colorado Boulder, CO 80309 USA, E-mail: (adam.herrmann@colorado.edu); Hanspeter Schaub is with the Aerospace Engineering Sciences Department and is the Schaden Leadership Chair, University of Colorado, Boulder, CO 80309 USA, E-mail: (hanspeter.schaub@colorado.edu). *(Corresponding author: Adam Herrmann.)*

I. INTRODUCTION

On-board planning and scheduling for spacecraft operations will become a requirement for future mission architectures in multiple domains. In the Earth-observing domain, on-board planning and scheduling is useful for replanning in the event of a ground station outage or unexpected science collection opportunity. For large constellations, on-board planning and scheduling can reduce the burden on operators, saving both time and cost. For deep space missions, on-board planning and scheduling will reduce constraints imposed by the round-trip light-time delay and facilitate operations in the uncertain dynamics environments of asteroids and comets. This work explores on-board planning and scheduling for a single Earth-orbiting spacecraft, but the lessons learned can also be applied to deep space mission architectures.

Typically, planning and scheduling is a ground-based activity in which a plan is generated on the ground using planning software. The plan is then sequenced and uplinked to the spacecraft for open-loop execution. Efforts have been made to automate the ground-based process. The Automated Scheduling/Planning Environment (ASPEN) software architecture is one such architecture used for a variety of spacecraft missions [1]. Significant work has been performed to develop on-board systems that modify the plans generated by ASPEN. The Continuous Activity Scheduling Planning Execution and Replanning (CASPER) tool uses iterative repair to modify plans generated by ASPEN in the event of resource constraint violations or unexpected science opportunities [2]. ASPEN and CASPER have been applied to several missions to demonstrate this process and improve operations. ASPEN and CASPER were deployed on the Earth-Observing 1 mission as a part of the Autonomous Sciencecraft Experiment to demonstrate on-board planning and scheduling modification to detect and respond to opportunistic science events [3], [4]. These tools were also deployed as a part of a larger web of space- and ground-based sensors to detect and capture data on volcanoes and floods [5], [6]. While ASPEN and CASPER have increased the autonomous capability of spacecraft and saved millions of dollars in operations costs, methods with more control over operational decisions are required for fully autonomous spacecraft.

In the Earth-observing satellite (EOS) scheduling problem, one or more spacecraft must collect and downlink science data to one or more ground stations on the Earth. Several science objectives may be considered. One such science objective is nadir-pointing data collection to maximize the surface area of the Earth imaged and downlinked. Another is a target-pointing science objective in which hundreds or thousands of individual targets must be imaged and downlinked by a spacecraft with three-axis attitude control capabilities. This latter problem is commonly referred to as the agile Earth-observing satellite (AEOS) scheduling problem. Real world examples of such missions include the two-spacecraft Pleiades constellation developed by the National Center of Space Research (French Space

Agency) or the WorldView constellation developed by DigitalGlobe [7]. Combinations of these science objectives may also be considered, depending on the mission architecture. Nag et al. [8] formulate an AEOS scheduling problem for a constellation of CubeSats and apply dynamic programming to solve the problem, demonstrating a large performance improvement over traditional nadir-pointing satellite architectures. In literature, mixed-integer programming is the most common problem formulation for the agile EOS scheduling problem [9], [10], [11]. While these approaches find optimal solutions to the AEOS scheduling problem, the solutions are brittle to a change in initial conditions. Furthermore, because they are executed open-loop on board the spacecraft, a significant deviation in resource consumption or the addition of new science opportunities cannot be easily accounted for on board and require that a new plan is generated on the ground.

Reinforcement learning (RL) has recently been posed as a candidate for the EOS scheduling problem due to its ability to generalize across initial condition distributions, rapidly compute plans after training, and execute closed-loop on board the spacecraft to dynamically respond to changes in the environment. Haijiao et al. [12] formulate a satellite scheduling problem with data storage constraints where the next imaging task is evaluated for acceptance or rejection as it becomes available and apply the asynchronous advantage actor–critic (A3C) algorithm to solve the problem. Zhao et al. [13] apply RL to solve the scheduling and timing problem in a two-phase manner for an EOS, demonstrating that the RL approach can match the performance of a genetic algorithm. Wei et al. [14] formulate a similar scheduling and timing problem and apply an actor–critic RL algorithm that outperforms a genetic algorithm in terms of cost and execution time. He et al. [15] apply deep Q-learning to an Earth-observing task scheduling problem and utilize a heuristic algorithm to solve the timing problem. Hadj-Salah et al. [16], [17] apply actor–critic RL (A2C) to a large area coverage scheduling problem and demonstrate that the proposed method meets or exceeds state-of-the-art heuristic algorithms under various weather conditions. While these methods have demonstrated success in generating solutions for EOS scheduling problems, especially when compared with heuristic or genetic algorithms, they typically do not consider data dynamics, data downlink, and/or energy dynamics. Harris et al. [18] formulate an EOS scheduling problem in which the goal is to maximize the amount of time the spacecraft spends in the nadir-pointing science mode while also managing resources, such as power and reaction wheel momentum. They apply shielded proximal policy optimization to demonstrate safe operations while maximizing data collection. Herrmann and Schaub [19] formulate an EOS scheduling problem where the goal is to maximize the amount of science data downlinked while managing power, reaction wheel momentum, and on-board data storage. They show that Monte Carlo tree search (MCTS) and state–action value network regression can compute near-optimal solutions to the problem. The state–action value neural networks may be executed in tenths of a second to compute the next mode the spacecraft should enter. While these problem formulations are an important step toward more complicated planning problems, they do not account for the presence of multiple targets that require precise pointing for imaging. Eddy and Kochenderfer formulate a semi-Markov decision process (MDP) for the agile EOS scheduling problem and apply MCTS to generate optimized task schedules [20]. While the authors formulate a multi-target problem, they do not generalize the solutions to arbitrary initial conditions for on-board execution, requiring that a new solution is computed using MCTS for each unique initial condition. This work applies MCTS methods to the agile EOS scheduling problem, generating optimal spacecraft plans that may be rapidly executed on board for any initial condition contained within the training distributions.

This work formulates an AEOS scheduling problem in which multiple targets on the surface of the Earth must be imaged and downlinked by scheduling a sequence of actions to image the targets, downlink them, and manage spacecraft resources such as power, reaction wheel speeds, and on-board storage. The problem is formulated as an MDP where the number of targets included in the state and action spaces is explored to determine the effect on performance. The targets in the state and action spaces are replaced with the next set of upcoming targets as they are passed over or imaged by the spacecraft. MCTS is utilized to benchmark the performance of the MDP with different numbers of targets in the action space. Two different MCTS backup operators are compared—an incremental averaging operator and a maximization operator. Furthermore, MCTS is compared with a genetic algorithm on the basis of reward and single-core wall clock time. State–action value neural network regression is then used to compute state–action value approximations using training data generated by MCTS. A search of the neural network hyperparameters is performed and presented, and the learned policies are compared with MCTS. Finally, a study is performed to determine how robust the learned policies and the genetic algorithm are to target densities outside of the training densities. This work is a substantial iteration of past work that formulates the MDP and presents preliminary results on the neural network regression of the state–action value estimates [21]. This work includes several new innovations, such as the use of the maximization backup operator, comparisons to a genetic algorithm, a more complete hyperparameter search and presentation of the results, and a study on the robustness of the learned policies to target densities outside of the target density used in training.

II. PROBLEM STATEMENT

A. AEOS Scheduling Problem

In the AEOS satellite scheduling problem, one or more spacecraft collect and downlink data to one or more ground stations on the surface of the Earth over some planning

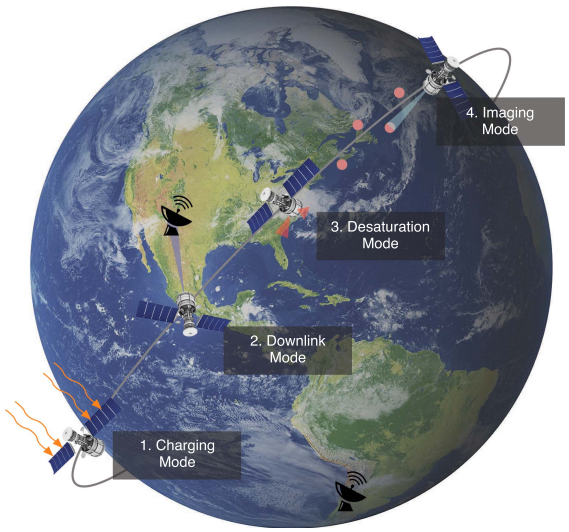


Fig. 1. AEOS scheduling problem. The decision-making agent selects the schedule of flight modes to manage resources and maximize the weighted sum of imaging targets imaged and downlinked, which are shown in red.

horizon, which is defined as the total amount of operational time considered for planning and scheduling. In this formulation of the AEOS scheduling problem, a single spacecraft in a 500 km altitude low Earth orbit collects images of targets located on the Earth's surface, which are downlinked to ground stations in NASA's Near Earth Network [22]. Over the course of a three-orbit planning horizon, the spacecraft has a set of 135 targets available along its flight-path for imaging, each with its own priority (1 is the highest and 3 is the lowest). This set of all targets is referred to as **T**, which is ordered by when the spacecraft has access to the targets. The goal is to maximize the weighted sum of the targets in **T** that are collected and downlinked. The planning horizon is broken into 45 discrete planning intervals where new scheduling decisions are made. The decision-making intervals do not overlap, and each interval is 6 min long. The length of the planning intervals is primarily driven by the time required for the attitude of the spacecraft to converge to the reference. If the planning intervals are too small, the flight modes may become unstable. Because the planning horizon is broken into 45 decision-making intervals, the maximum number of targets that can be imaged is 45 (if operational resources are ignored). The spacecraft schedules resource management and imaging tasks at each planning interval to achieve the goal. Fig. 1 depicts this problem.

B. Markov Decision Process

The AEOS scheduling problem is formulated as an MDP, a sequential decision-making problem in which an agent observes some state s_i and selects and action a_i following a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which maps states to actions. The agent observes a new state s_{i+1} and receives a reward r_i based on the reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. This process is demonstrated in Fig. 2. MDPs follow

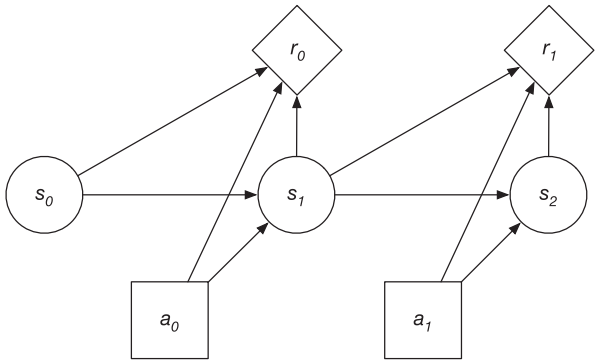


Fig. 2. MDP. A decision-making agent takes an action a_i while in state s_i. The decision-making agent transitions to a new state s_{i+1} and receives a reward r_i.

the Markov assumption, meaning the next state is conditionally dependent only on the current state and action. Mathematically, this may be stated as $T(s_{i+1}|s_i, a_i) = T(s_{i+1}|s_i, a_i, s_{i-1}, a_{i-1}, \ldots, s_0, a_0)$. All relevant state information for the purposes of maintaining this assumption must be included in the state space.

1) *Target Notation:* In this formulation of the AEOS scheduling problem, the collection and downlink of individual targets is of particular interest. The relative geometry of the targets, length of the planning intervals, and limited spacecraft resources preclude the collection of every target. Therefore, the decision-making agent must make tradeoffs between the different targets and its resources to maximize the weighted sum of targets collected and downlinked. Ideally, each target is included in the state space so the agent has perfect information. However, this significantly increases the complexity of the problem by increasing the size of the state and action spaces because there are 135 targets in set **T**. Therefore, only a subset of **T** that contains the next upcoming, unimaged targets should be considered. This subset is given in (1), where J is the number of targets in the state and action spaces and **D** is a subset of **T** containing the imaged or passed targets.

$$\mathbf{U} = \{c_j \in (\mathbf{T} - \mathbf{D}) \mid \forall\ j \in [1, J]\} \tag{1}$$

The targets in set **T** are generated by sampling the positions of the spacecraft in the Earth-centered, Earth-fixed (ECEF) frame and projecting them onto the surface of the Earth. Before the projection, a small amount of noise is added to the spacecraft position to generate off-nadir targets. An example of the generated targets in terms of latitude and longitude may be found in Fig. 3. The red, orange, and yellow dots are targets, and the blue line is the ground track of the spacecraft. The targets are at most one to two degrees off of the ground track of the spacecraft. Due to the nature of the target generation, there are varying concentrations of targets with varying priorities that the agent will encounter.

2) *State Space:* Real-world problems are difficult to cast as MDPs with strict adherence to the Markov assumption. The designer of the MDP must decide which information is most relevant to the problem. The state space

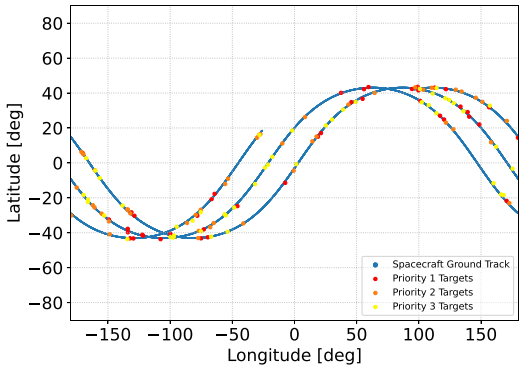


Fig. 3. Example of spacecraft ground track and target locations. Red is high priority. Orange is medium priority. Yellow is low priority.

$\mathcal{S}$, for the AEOS scheduling problem may be found in the following:

1) ECEF spacecraft position, ${}^{\mathcal{E}}\mathbf{r}$;
2) ECEF spacecraft velocity, ${}^{\mathcal{E}}\mathbf{v}$;
3) image tuples for targets $c_j \in \mathbf{U}$;
   a) target position in the spacecraft Hill frame, ${}^{\mathcal{H}}\mathbf{r}_j$;
   b) priority, p_j;
4) L^2 norm of Modified Rodrigues Parameter (MRP) attitude error, $||\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}||$;
5) L^2 norm of angular attitude rate vector, $||{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}||$;
6) reaction wheel speeds, $\boldsymbol{\Omega}$;
7) battery charge, z;
8) eclipse indicator, k;
9) stored data in buffer, b;
10) data transmitted, h.

The left-superscript on a vector denotes the coordinate frame relative to which vector components are taken [23]. The body-fixed frame is referred to as $\mathcal{B}$, and the desired attitude reference frame is referred to as $\mathcal{R}$. The inertial frame is denoted with $\mathcal{N}$. The position and velocity of the spacecraft expressed in an ECEF frame $\mathcal{E}$ are included in the problem to retain information on upcoming downlink windows. For each target included in the state and action spaces, a target tuple is also included that contains the target position expressed in the spacecraft Hill frame $\mathcal{H}$ and the target priority. The position of each target expressed in the spacecraft Hill frame allows the agent to understand the geometric relationship between each target to make tradeoffs between access and priority.

The attitude error $\boldsymbol{\sigma}_{\mathcal{B}/R}$, attitude rate ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/N}$, and reaction wheel speeds $\boldsymbol{\Omega}$ provide state information for the attitude control system. The battery charge z and eclipse indicator k provide state information on the spacecraft's power system. Likewise, the amount of data stored in the buffer and amount of data downlinked over the planning interval provide state information on the spacecraft's data management system and ability to downlink data to the ground.

Each state is approximately normalized to a range of $[-1, 1]$ or $[0, 1]$ to aid convergence in function approximation. The ECEF position vector and Hill-frame relative target positions are normalized by the radius of the Earth. The ECEF velocity is normalized by the velocity of a circular orbit at the Earth's surface. Reaction wheel speeds are normalized using the maximum reaction wheel speed. Likewise, the battery charge and data buffer storage level are normalized by their maximum capacity.

3) *Action Space:* A mode-based planning approach is taken where each mode represents a high-level spacecraft behavior. The low-level behavior of each mode is dictated by the attitude reference and ON/OFF states of each spacecraft subsystem. Each mode is entered for a total of 6 min, which is primarily constrained by the rate at which the attitude control system can converge to the attitude reference. By utilizing a mode-based planning approach, the continuous behavior of the spacecraft is decomposed into discrete actions, making the planning problem tractable. The action space, $\mathcal{A}$, is given in the following:

1) charge;
2) desaturate;
3) downlink;
4) image target $c_1 \in \mathbf{U}$
   $\vdots$
6) image target $c_j \in \mathbf{U}$.

In the charging mode, the spacecraft turns OFF the imager and transmitter and points its solar panels at the sun. The desaturation mode is the same as the charging mode, but the spacecraft thrusters are used to remove momentum from the reaction wheels. In the downlink mode, the spacecraft points in the nadir direction and turns ON the transmitter to downlink data to available ground stations. A ground station is accessible if the spacecraft is within the elevation and range requirements of the station. In the imaging mode, the spacecraft points at target j and takes an image once the spacecraft is within the elevation and range requirements of the target. The same access model is shared between ground stations and imaging targets for simplicity. For imaging, an additional attitude requirement is added such that the L^2 norm of the attitude error is below a threshold of 0.1 rad.

4) *Transition Function:* Due to the continuous dynamics of the AEOS scheduling problem, it is difficult to construct an explicit transition function using conditional probabilities that accurately captures state transitions. Therefore, the transition function is represented with a generative model $G(s_i, a_i)$ given in (2). A generative model simply returns a new state s_{i+1} and reward r_i by sampling an underlying distribution, integrating equations of motion, or some combination of both.

$$s_{i+1},\ r_i = G(s_i, a_i). \tag{2}$$

The Basilisk astrodynamics software architecture [24] is used to construct a simulation to model the complex behavior of the spacecraft and environment. The Basilisk simulation is wrapped within a gym environment, which provides a standard interface for the agent to interact with

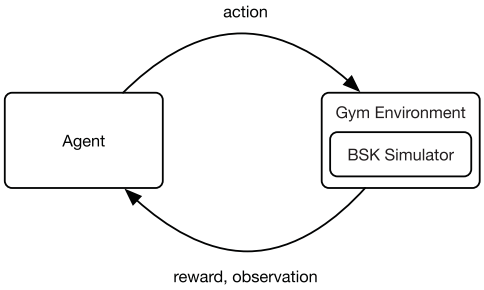


Fig. 4. Gym environment interface [19]. A decision-making agent passes an action to the environment, which returns a reward and observation of the state.

the simulation.[1] This is depicted in Fig. 4. The agent passes an action to the environment, which turns Basilisk models ON or OFF based on the mode. The simulation is integrated forwards in time for 6 min with a time step of 1 s. Afterward, the agent receives a reward and the new state.

The Basilisk astrodynamics software architecture is a high-fidelity astrodynamics simulation tool developed by the Autonomous Vehicle Systems Laboratory at the University of Colorado, Boulder.[2] The Basilisk simulation described within this work is an iteration of the simulator used in past work [19]. The base simulator for the AEOS scheduling problem includes high-fidelity orbit and attitude dynamics, an on-board data system, and an on-board power system. In this work, an instrument controller is included that turns the imager ON to take an image if access and attitude requirements are met. The simulated image is then passed to the data buffer and downlinked if and when a downlink is initiated by the spacecraft.

5) *Reward Function:* The reward function $R(s_i, a_i, s_{i+1})$ is formulated as a piecewise function of the current state, action, and next state. The return at step i is computed as follows:

$$r_i = \begin{cases} -10 & \text{if failure} \\ \sum_j^{|\mathbf{T}|} H(d_j) & \text{if } \neg\text{failure} \wedge a_i \text{ is downlink} \\ 0.1H(w_j) & \text{if } \neg\text{failure} \wedge a_i \text{ is image } c_j \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

If the agent fails, a failure penalty of -10 is returned and the episode terminates. The failure condition is true if the spacecraft exceeds the maximum reaction wheel speeds, expends all charge in the battery, or overfills the data buffer. Mathematically, this is represented as

$$\text{failure} = \left(z = 0 \ \vee \ \text{any}\left(\hat{\mathbf{\Omega}} \geq 1\right) \ \vee \ b \geq 1\right). \tag{4}$$

A function $H(x_j)$ is formulated for each image tuple that checks if the state variable x is false at step i and true at step $i+1$, returning 1 divided by the image priority p_j if these conditions are met.

$$H(x_j) = (1/p_j) \text{ if } \neg x_{j_i} \ \wedge \ x_{j_{i+1}}. \tag{5}$$

[1][Online]. Available: https://bitbucket.org/avslab/basilisk-gym-interface/src/develop/

[2][Online]. Available: http://hanspeterschaub.info/basilisk

If the downlink mode is initiated and a resource management failure does not occur, the $H(d_j)$ function for all targets is computed and summed using a downlink state d_j that represents whether or not target j has been downlinked. In other words, each target is checked to see whether or not it has been downlinked for the first time. Reward is returned for each target for which this condition is true, and the total reward for all targets is summed together.

If the image target c_j mode is initiated and a resource management failure does not occur, the $H(w_j)$ operator for target c_j is computed, returned, and scaled by 0.1. The variable w_j represents if the target has been imaged or not. In other words, target c_j is checked to see if an image was captured for the first time, and a small reward is returned if this is true. The addition of this small positive reward helps to make reward less sparse, which facilitates exploration in MCTS.

6) *Target Replacement:* After each step, the targets in **U** are checked to see if they have been imaged or passed. If so, these targets are added to **D**, and **U** is updated and added to the action space. While this does add unobservability to the problem, there is a value for $|\mathbf{U}|$ that will render this impact negligible because the added information of more targets will only marginally improve observability while increasing problem complexity and required training time. If the agent had observability over every target and their priorities, it would have perfect information and the ability to compute the value exactly. In this work, the size of **U** is explored to determine when the agent has enough information to extract the maximum possible reward from the environment.

III. METHODS

A. Solving MDPs

Solving MDPs is done by solving for the optimal policy $\pi^*(s_i)$, which is the mapping from states to actions that results in the maximum expected reward. The optimal policy can also be represented as the policy that maximizes the value function, as shown in the following:

$$\pi^*(s_i) = \arg\max_{\pi} V^{\pi}(s_i). \tag{6}$$

The optimal value function $V^*(s_i)$ is the expected value of future reward when starting in state s_i and following the optimal policy until the episode terminates [25]. For a deterministic and finite-horizon MDP, it is defined recursively using the Bellman operator

$$V^*(s_i) = \max_{\text{a}} \left(R(s_i, a_i) + V^*(s_{i+1})\right). \tag{7}$$

The state–action value function $Q(s_i, a_i)$ is the expected value of future reward given a state s_i and action a_i. The optimal value function can be found by maximizing $Q^*(s_i, a_i)$.

$$V^*(s_i) = \max_{\text{a}} Q^*(s_i, a_i). \tag{8}$$

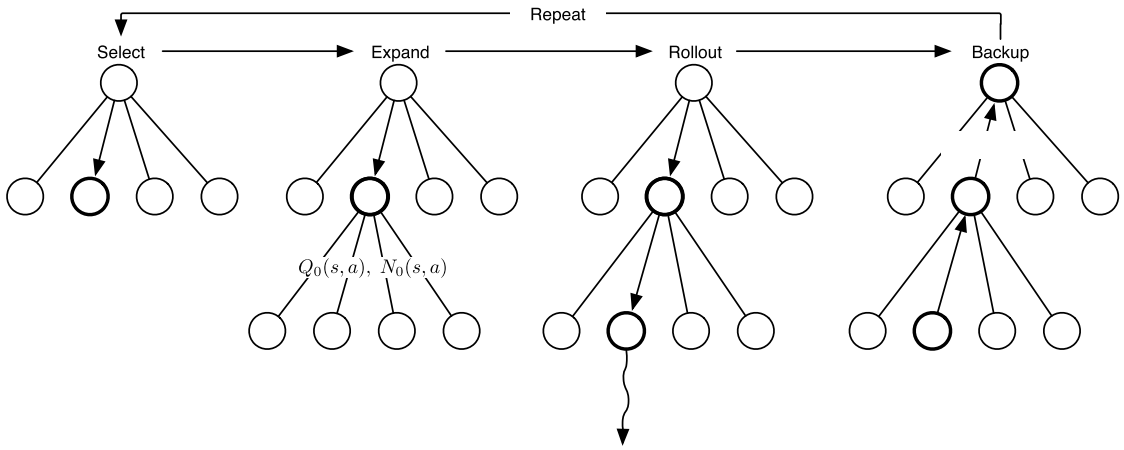


Fig. 5. MCTS algorithm [19]. In the *selection* step, MCTS selects the action that maximizes the state–action value estimate and an exploration term. In the *expansion* step, MCTS initializes the state–action value function and the number of times each state–action pair has been selected. During *rollout*, MCTS uses a heuristic rollout policy to select actions until a specified depth. Finally, during *backup*, MCTS updates the state–action value estimates with the return from rollout.

If the optimal state–action value function is known, the optimal policy is

$$\pi^*(s_i) = \arg\max_{\mathrm{a}} Q^*(s_i, a_i). \tag{9}$$

This work solves for the state–action value function $Q(s, a)$ using online algorithms and supervised learning. The benefit of using online algorithms is that only states that are reachable from state s_i are explored. Therefore, online algorithms are well suited for generative transition functions, which this work utilizes. Adding supervised learning allows for the policies found by the online algorithms to generalize across the state space.

B. Monte Carlo Tree Search

MCTS is an online search algorithm that uses a combination of simulation and rollout to determine the next optimal action the agent should take in the planning problem [26], [27]. The exact implementation of MCTS may be found in [27, Ch. 4.6.4]. At each step through the environment, MCTS computes the optimal action by following the process in Fig. 5. During the selection step, MCTS selects the action that maximizes the state–action value estimate, $\hat{Q}(s, a)$, and the exploration bonus, U. The exploration bonus is a function of an exploration constant ϵ, the number of times the state has been visited $N(s)$, and the number of times a particular state–action tuple has been selected $N(s, a)$.

$$U = \epsilon \sqrt{\frac{\log N(s)}{N(s, a)}}. \tag{10}$$

If MCTS reaches a state in the search tree it has not seen before, it initializes $\hat{Q}(s, a)$ and $N(s, a)$. Then, MCTS executes a rollout policy to a specified depth. A rollout policy is a random or heuristic policy MCTS uses to find areas of high reward, which are promising to search. After rollout, MCTS backs up the reward through each state–action tuple to update $\hat{Q}(s, a)$. In this work, two separate backup operators are compared: an incremental averaging operator and a maximization operator. The incremental averaging operator is given inthe following where q is the return after simulation and rollout:

$$\hat{Q}(s, a) = \hat{Q}(s, a) + \frac{q - \hat{Q}(s, a)}{N(s, a)}. \tag{11}$$

When the maximization operator is utilized, each $\hat{Q}(s, a)$ is initialized to -10. During backup, $\hat{Q}(s, a)$ is set to q if the return is greater than $\hat{Q}(s, a)$; otherwise, no change is made. The equation for the maximization backup operator is provided in the following:

$$\hat{Q}(s, a) = \max\left\{\hat{Q}(s, a), q\right\}. \tag{12}$$

The entire selection, expansion, rollout, and backup process is repeated for a specified number of simulations-per-step. After the maximum number of simulations-per-step is reached, the action that maximizes the state–action value estimate is selected and the agent takes a step forward in the environment.

1) *Rollout Policy:* A safety MDP and rollout policy are derived as described by Herrmann and Schaub [19]. The safety MDP discretizes the state space to reduce dimensionality to several safety states

$$\mathcal{S}_{\text{safety}} : (\text{Tumbling, Low Power, Saturated, Buffer Full}). \tag{13}$$

The safety states take a value of true or false depending on whether the relevant resource state variables are above or below a safety limit. A rollout policy is generated for the safety MDP that guarantees a resource constraint failure does not occur, which allows MCTS to only explore areas in the state space that are promising. At times, the safety MDP achieves a nominal state ($s_i = (0, 0, 0, 0)$), meaning that any action can be safely taken. For this problem, the target in the state space with the minimum Hill-frame position is selected for imaging if the state of the safety MDP is

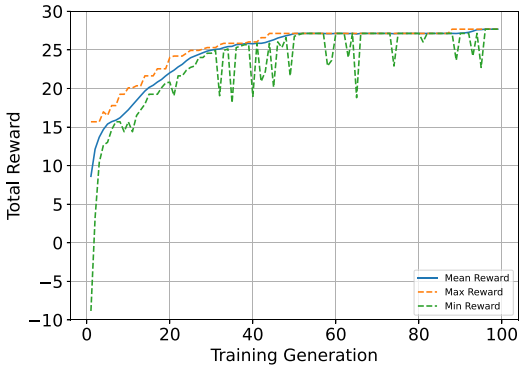


Fig. 6. Genetic algorithm convergence example.

nominal. However, if a ground station is accessible, the downlink mode is initiated instead.

C. Genetic Algorithm

In order to provide a comparison to state-of-the-art scheduling algorithms, a genetic algorithm is implemented using the DEAP evolutionary computation framework.[3] A genetic algorithm is a metaheuristic optimization algorithm that utilizes mutation, crossover, and selection operators to search for good solutions to a given problem. The genetic algorithm begins with a population of random sequences of actions. Each member of the population is evaluated using a fitness function. In the agile EOS scheduling problem, the fitness function is simply the reward function of the MDP. After evaluation, selected members of the population either mutate (i.e., a given action changes to another action following some probability) or crossover (action sequences combine with one another). The new population is evaluated once more, and the entire process repeats for a given number of generations. DEAP's two-point crossover is utilized, and both the mutation and crossover probabilities are set to 0.25. An example of the convergence of the genetic algorithm is provided in Fig. 6.

D. Supervised Learning

A training process is employed to create a state–action value function that is generalizable over a range of initial conditions. The entire process is demonstrated in Fig. 7. MCTS is used to solve the planning problem for hundreds of initial conditions sampled from distributions of the true anomaly, argument of periapsis, longitude of the ascending node, battery charge level, etc. [19]. Because MCTS estimates the state–action value function for each state–action pair, the estimates from each solved planning horizon are first updated using the realized reward from stepping through the planning problem. Afterward, each state–action value estimate is placed in a dataset that is split between a training set and validation set. These state–action value estimates are regressed over using artificial neural networks to produce a state–action value function approximation, $Q_\theta(s, a)$. After $Q_\theta(s, a)$ is computed, a new policy may be derived in which the action that maximizes $Q_\theta(s, a)$ is returned. This is the policy used by the on-board agent to determine the next optimal action the spacecraft should take.

$$\pi(s_i) = \arg\max_{a_i} Q_\theta(s_i, a_i). \tag{14}$$

The learned policies are then validated and benchmarked in the environment. The benefit of a neural network representation of the state–action value network is that it may be executed in milliseconds on board the spacecraft to compute the optimal action based on the current state of the environment.

IV. RESULTS

A. Action Space Parameterization

The effect of the size of $\mathbf{U}$ on performance is an important question this work explores. A constant target density of 45 possible targets per orbit (135 total targets in $\mathbf{T}$ over three orbits) is assumed. It is worth mentioning that in addition to target density, the required number of targets in $\mathbf{U}$ is also a function of the length of the planning interval. A spacecraft that makes a decision every three minutes will be able to collect more targets than a spacecraft that makes a decision every six minutes, assuming the spacecraft can slew from target to target fast enough. This work only considers 6-min planning intervals.

To determine how the size of $\mathbf{U}$ impacts performance, an experiment is conducted in which the number of targets in the action space is increased from one through five. A range of $|\mathbf{U}| = \{1, \ldots, 5\}$ is selected because the spacecraft rarely has access to more than five targets. MCTS with an incremental average operator, MCTS with a maximization backup operator, and the genetic algorithm are applied to solve the problem for each size of $\mathbf{U}$. For each $|\mathbf{U}|$, the MCTS hyperparameters that balance performance and execution time are selected. For the incremental average backup, an exploration constant of $\epsilon = 10$ is selected. For the maximization backup, an exploration constant of $\epsilon = 20$ is used because of the higher state–action value estimates. The number of simulations-per-step for each $|\mathbf{U}|$ is linearly increased from 15 to 35, with $|\mathbf{U}| = 1$ at 15 simulations-per-step and $|\mathbf{U}| = 5$ at 35 simulations-per-step. The increase in simulations-per-step is selected due to the increase in problem complexity due to additional actions in the action space. The number of times future state–action pairs are visited is on the order of

$$\frac{1}{|\mathcal{A}|^{\text{depth}}}. \tag{15}$$

The depth is the depth of the search. To equalize exploration between the different experiments, the number of simulations-per-step is marginally increased to account for this decay. Furthermore, past work has shown that after a certain point, additional simulations-per-step in the EOS

[3][Online]. Available: https://deap.readthedocs.io/en/master/

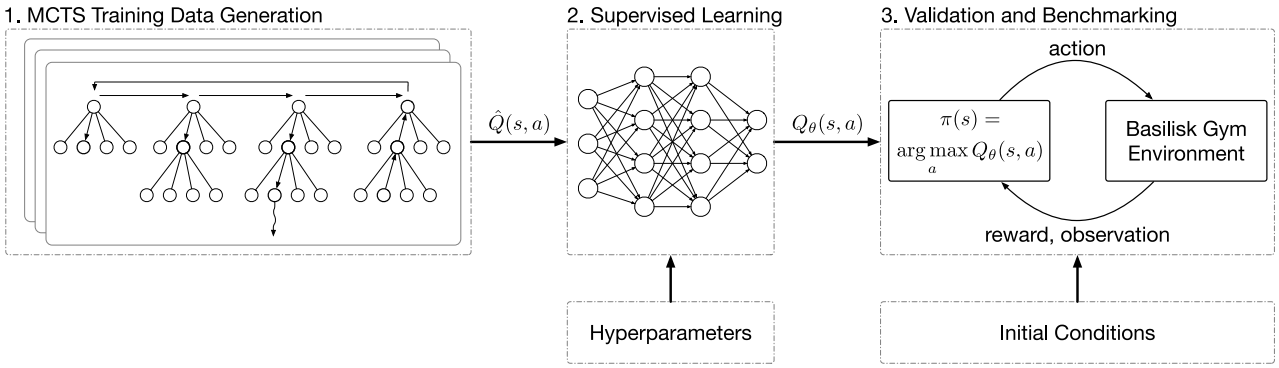


Fig. 7. Training pipeline. MCTS is used to generate thousands of estimates of the state–action value function. Various artificial neural networks are used to regress over the training data, and each neural network is then validated and benchmarked in the environment.

scheduling problem do not result in a large difference in the quality of MCTS solutions [19]. Therefore, the smallest number of simulations-per-step is selected for each $|\mathbf{U}|$ such that an MCTS performance plateau is achieved.

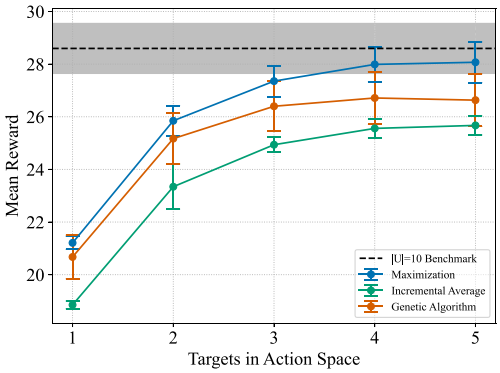
The genetic algorithm is initialized with a population of 80 for $|\mathbf{U}| = 1$ and is linearly increased to 160 for $|\mathbf{U}| = 5$ to account for the increase in problem complexity. For each size of $\mathbf{U}$, the number of generations is set to 100.

In Fig. 8, the average reward, number of imaged targets, and number of downlinked targets for each size of $\mathbf{U}$ are plotted, along with the 95% confidence intervals for each. The maximum possible number of imaged targets is 45. However, this is impossible to obtain because of the data buffer constraint. Therefore, MCTS with the maximization backup operator is used to generate a value for each metric's maximum expected value (75 simulations-per-step and ten targets in the action space). This is provided using the dotted black line, with the 95% confidence intervals provided in gray. The solid blue line is generated from MCTS using the maximization backup operator. The solid green line is generated from MCTS using the incremental averaging backup strategy. Finally, the orange line is generated using the genetic algorithm. The reward asymptotically approaches the maximum possible reward for each backup strategy as $|\mathbf{U}|$ increases. However, the maximization backup strategy consistently achieves more reward. The maximization backup operator also images and downlinks more targets on average. This is because the incremental averaging operator averages in the return from future low-value states. The maximization backup operator does not average in this return and instead sets the action-value estimate to the maximum return found during search. Consider a scenario in which the agent is close to filling up the spacecraft's data buffer. The state–action value estimates along the optimal trajectory of actions will be lower than the true optimal state–action value, the closer the agent is to filling up the data buffer. In contrast, the maximization operator will not consider the low return from a data buffer overflow until it reaches a state in which it will overfill the data buffer at the next time step if it takes an image.

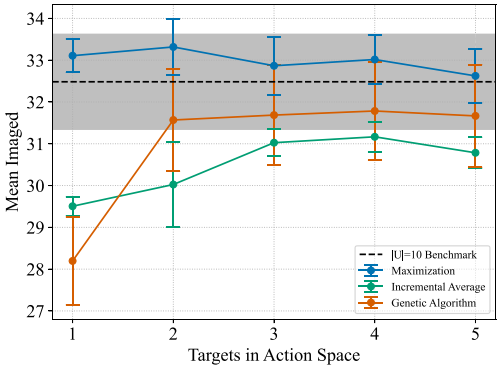
An interesting result in Fig. 8 is that the mean number of imaged and downlinked targets are relatively constant for each backup operator as the size of $\mathbf{U}$ increases. The average reward, however, increases as $|\mathbf{U}|$ increases. The reason for this disparity is that as more targets are included in the action space, MCTS has more of an opportunity to select high priority targets over low-priority targets. The target priorities are uniformly sampled from a range of one to three. With one target in the action space, there is a probability of 1/3 that the target will be priority one. However, this probability increases with the size of $\mathbf{U}$ using the following equation: $1 - (2/3)^{|\mathbf{U}|}$. With five targets in the action space, the probability that MCTS will have at minimum one priority one target is approximately 0.868.

In terms of reward, the genetic algorithm performs better than MCTS with the incremental averaging operator, but not as good as MCTS with the maximization operator. Generally speaking, this is true for the number of imaged targets as well. However, the genetic algorithm usually downlinks more targets than the maximization operator and incremental averaging operator. The GA seems to take better advantage of downlink opportunities, but does not leverage the imaging target priorities quite as well. The mean single-core wall clock time of each algorithm is plotted in Fig. 8(d). The MCTS computation time is typically between 10% and 20% of that of the genetic algorithm. The reason for this is that MCTS leverages expert knowledge in the rollout policy to find high-value states, so the MCTS algorithm is more sample efficient. This implementation of the genetic algorithm does not have an analogous mechanism. It should be noted that the computation times are on the order of hours long for each algorithm. If a faster simulator was used, especially one that makes linear assumptions about the dynamics of the problem, these simulation times could be greatly reduced.

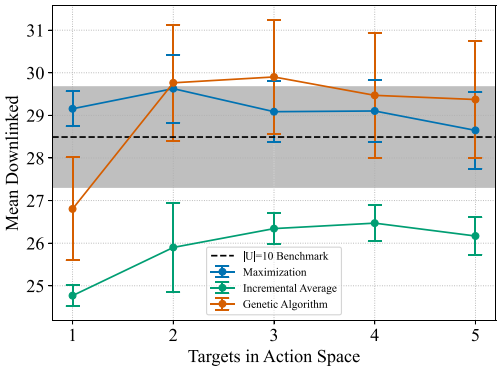
In Fig. 9, the normalized frequency of the number of accessible targets at any given step is provided. The 1σ standard deviation of the frequencies between different initial conditions is plotted in orange. This is computed by generating the normalized frequencies for each unique initial condition and computing the associated standard



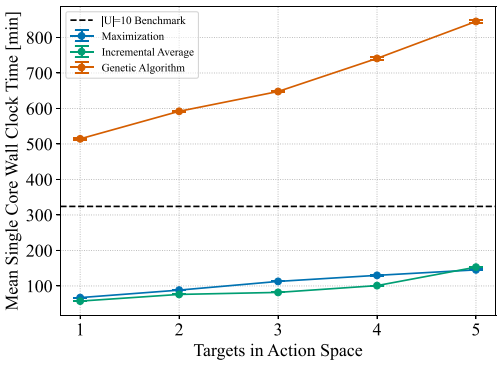
(a) Mean reward.



(b) Mean number of imaged targets.



(c) Mean number of downlinked targets.



(d) Mean single core wall clock time.

Fig. 8. MCTS performance metrics for different sizes of **U** with associated 95% confidence intervals. (a) Mean reward. (b) Mean number of imaged targets. (c) Mean number of downlinked targets. (d) Mean single-core wall clock time.

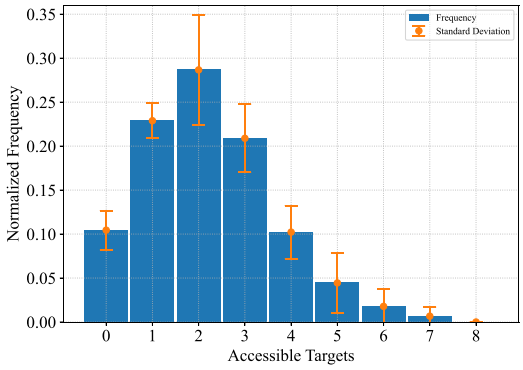


Fig. 9. Normalized frequency of the number of accessible targets at each planning interval. $|\mathbf{T}| = 135$.

TABLE I
Neural Network Hyperparameters

Parameter	Value
Nodes Per Hidden Layer	{50, 100, 200, 300}
Hidden Layers	$\{1, \cdots, 5\}$
Activation Function	{Leaky ReLU, tanh}
α	{0.1, 0.2}
Dropout	{0.01, 0.1}
Epochs	10,000
Batch Size	45,000
Loss Function	Mean Squared Error

deviations between each initial condition. The agent has no targets available to image in the next decision-making interval approximately 10% of the time. The agent has two targets available to image in the next decision-making interval about 28% of the time. Finally, the agent rarely has eight targets available for imaging and never has nine targets available for imaging. Therefore, ten targets in the action space provide MCTS with the maximum amount of targets required to get the maximum reward at any step, with some margin added. The frequencies of available targets also shed light on the curves in Fig. 8. The spacecraft has five or more targets available for imaging less than 10% of the time. The magnitudes of the frequencies between one through four targets are much larger and, therefore, have a much higher impact on the performance.

B. State–Action Value Function Approximation

To determine the appropriate artificial neural network hyperparameters for approximating the state–action value estimates generated by MCTS, a hyperparameter search is conducted over the number of nodes, number of hidden layers, activation function, dropout rate, and α parameter (specific to the Leaky ReLU activation function). The complete network parameters are included in Table I. The training data generation and neural network hyperparameter search were conducted on a computer with a Windows operating system, a 24-core AMD Threadripper 3960x processor, NVIDIA RTX 3090 graphics card, and 128 GB of RAM.

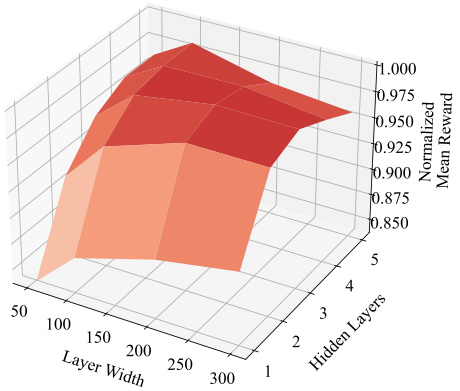
The neural networks are created and trained using the Keras[4] deep learning API in the Python programming language. In general, training data generation takes between 1 and 2 days with multiprocessing being utilized. Each network requires approximately 5–10 min of training using the GPU. If the CPU is utilized instead, this training time increases dramatically.

To approximate the state–action value estimates, fully connected feedforward neural networks with linear output layers are implemented with various hyperparameters. To prevent overfitting, dropout is added to each hidden layer [28]. The dropout rate is the probability that a node will be dropped during training. Two dropout rates are explored: 0.01 and 0.1. If the dropout rate is too low, it will not have the desired effect. If the dropout rate is too large, the network may not sufficiently learn. In order to determine the appropriate number of hidden layers and width of these layers, a hyperparameter search is conducted over a range of these parameters. The number of hidden layers is increased from one to five. Four network widths are considered −50, 100, 200, and 300 nodes. These ranges of values were iteratively increased until a depreciation in performance was present. In general, it is desirable to keep the network as small as possible to increase the speed of training and decrease the memory footprint and execution time of the network, especially when on-board execution is desired.

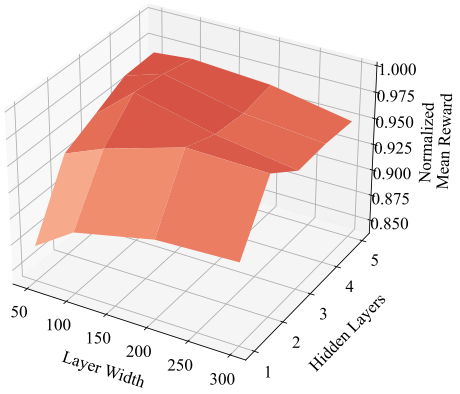
Both the Leaky ReLU and hyperbolic tangent activation functions are considered. The hyperbolic tangent activation function is a popular activation function that may be used for regression, but suffers from the vanishing gradient problem for deep networks [29]. Therefore, Leaky ReLU is also considered, which has become a popular choice for regression because it does not suffer from the vanishing gradient problem (for sufficiently large values of α) or the dying ReLU problem associated with the ReLU activation function. The α-parameter in the Leaky ReLU activation function is the slope of the activation function for $x < 0$. Two α-parameters are explored: 0.1 and 0.2. If the α-parameter is too small, Leaky ReLU may also suffer from the vanishing gradient problem.

MCTS is used to generate 45 000 data points (1000 unique planning horizons solved). In total, 90% of the data are used for training, and 10% of the data are used for validation. The data generated by MCTS is not pre- or post-processed (outside of MCTS updating the state–action value estimates after it steps through the environment). The networks are trained for 10 000 epochs using the mean squared error (MSE) loss function, a popular choice of loss function for regression problems. In general, the MSE for the training and validation set is between 10 and 20. The mean absolute error is also logged and is found to be in between 2 and 4. Overfitting is observed as the networks became larger, but in general this is not an issue for smaller networks with 1–3 hidden layers and 50–200 nodes.

[4][Online]. Available: https://keras.io/



(a) Leaky ReLU activation function.



(b) Hyperbolic tangent activation function.

Fig. 10. Hyperparameter surface plots for mean reward normalized by the mean reward of best performing network. $|\mathbf{U}| = 3$. (a) Leaky ReLU activation function. (b) Hyperbolic tangent activation function.

To validate the performance of each trained neural network, the state–action value networks are used to generate the policy described in (14), which is executed on a standard set of initial conditions. An example of this hyperparameter search is shown with the number of targets in the action space set to three, $|\mathbf{U}| = 3$. The same hyperparameter search is performed for the other sizes of $\mathbf{U}$, but these are not shown here for brevity. The general trends shown here are consistent among the other numbers of targets in the action space. In Fig. 10, a surface plot is shown that plots normalized reward against the number of nodes per hidden layers and number of hidden layers. The normalized reward is the average reward divided by the reward of the best hyperparameter combination. A surface plot is generated for each activation function. However, for each data point in the surface plot, the reward is averaged among the remaining hyperparameters. In the case of Leaky ReLU, the α parameter and dropout rate rewards are averaged. In the case of hyperbolic tangent, only the dropout rate reward is averaged. In general, performance is not as sensitive to the dropout rates and α parameters as it is to the width of the hidden layers and number of hidden layers.

A general trend can be extracted from the plots in Fig. 10. One to two hidden layers are typically insufficient to produce high-performing policies, particularly when small

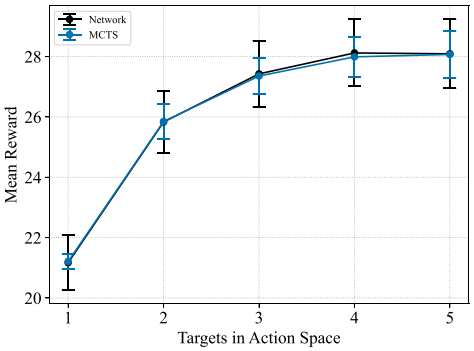
layer widths are used. Furthermore, performance is relatively poor for large layer widths and large numbers of hidden layers, albeit not as poor as the former case. In the former case, the network does not have enough parameters to sufficiently approximate the state–action value estimates. In the latter case, the network has too many parameters and becomes overfit. In the region that excludes these two conditions, performance is relatively high. It is also worth mentioning that no combination of layer widths or hidden layers produces less than 85% normalized reward. Therefore, the performance is relatively robust to the network architecture, which is an encouraging result.

C. Comparison Between MCTS and Learned Policies

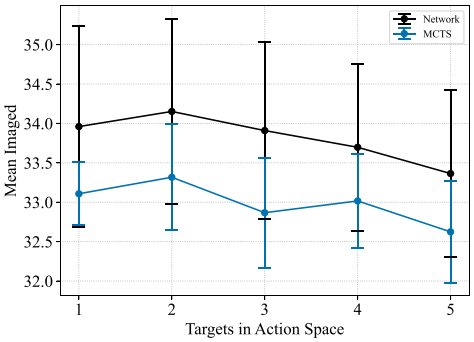
Once the state–action value network hyperparameterization is performed for each $|\mathbf{U}|$, as described in the previous section, the best state–action value networks for each $|\mathbf{U}|$ are selected to compare against MCTS. Each of these networks are trained using data generated from MCTS with the maximization operator solving approximately 1000 unique initial conditions. The average reward, number of imaged targets, number of downlinked targets, and the wall clock time, along with the 95% confidence intervals, for both MCTS and the best-performing neural networks are plotted in Fig. 11. For each $|\mathbf{U}|$, the neural network policies image and downlink anywhere between 0.5 and 1.0 more targets on average. There is a significant overlap in the confidence intervals, but the network policies image and downlink more targets for every number of targets in the action space. Because of this and the fact that the reward achieved by MCTS and the neural network policies is almost identical, it is likely that the neural network policies are slightly worse at imaging the high-priority targets. However, this difference is quite small, so it can be concluded that the neural network policies approximate the MCTS policy very well. Finally, the learned policies find a solution three orders of magnitude faster than MCTS.

D. Robustness to New Target Densities

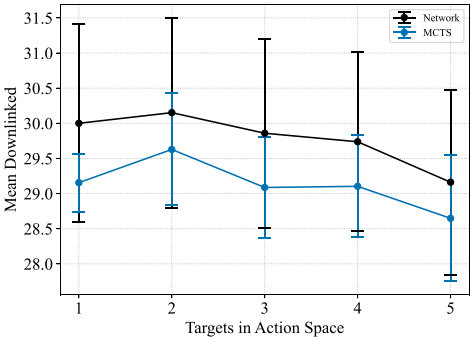
In the last experiment, the robustness of the trained state–action value function approximators is explored for various sizes of the total target set, $\mathbf{T}$. During training, $|\mathbf{T}|$ is set to 135. However, in a real operational scenario, this number will not be constant. The trained networks are deployed in environments with $|\mathbf{T}| = \{45, \ldots, 270\}$ in increments of 45. The results of this experiment are presented in Fig. 12(a) where the average reward and associated 95% confidence intervals are plotted. At $|\mathbf{U}| = 1$, there is little difference between the performance of the trained network for the different numbers of targets in the total target set. This is because the agent is only ever considering the next upcoming target. Adding or subtracting targets typically does not impact the performance, except when $|\mathbf{T}| = 45$. In this case, the targets are very sparse and performance is limited by the number of targets available for imaging and downlink, not the agent's ability to discern between



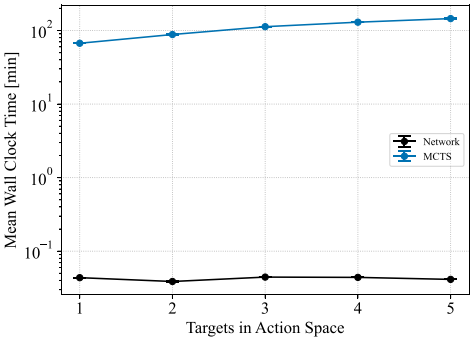
(a) Mean reward.



(b) Mean number of imaged targets.



(c) Mean number of downlinked targets.

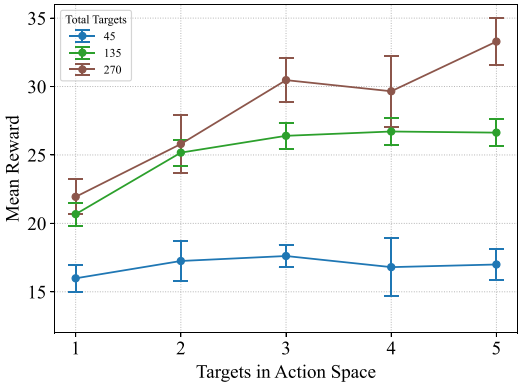


(d) Mean wall clock time.

Fig. 11. MCTS and learned policy performance for different sizes of $|\mathbf{U}|$. (a) Mean reward. (b) Mean number of imaged targets. (c) Mean number of downlinked targets. (d) Mean wall clock time.

different priorities. However, as the number of targets in the action space increases, a separation in performance emerges for $|\mathbf{T}| = \{90, 135, 180\}$. These three target densities are fairly distinct from one another, although there is some overlap in confidence intervals. For $|\mathbf{T}| = \{225, 270\}$, there

(a) Network trained on nominal target density.



(b) Genetic algorithm.

Fig. 12. Average reward for different sizes of **T**. (a) Network trained on nominal target density. (b) Genetic algorithm.

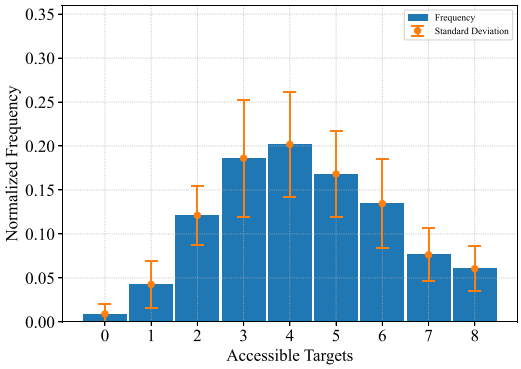


Fig. 13. Normalized frequency of the number of accessible targets at each planning interval. $|\mathbf{T}| = 270$.

is a lot of overlap in the means and confidence intervals for all numbers of targets in the action space. This is expected because priority one targets become more available as the target density increases. In Fig. 13, the normalized frequencies of available targets at each planning interval are plotted for $|\mathbf{T}| = 270$. In contrast to Fig. 9, the most commonly available number of targets is four as opposed to two. The agent has higher priority targets available to it more frequently, which increases the average reward.

A comparison to the genetic algorithm is performed once more, this time on different target densities. In Fig. 12(b), the average reward of the genetic algorithm is plotted for $|\mathbf{T}| = \{45, 135, 270\}$, which provides a lower, middle, and upper bounds on the performance of the GA. At $|\mathbf{T}| = 45$, the performance of the trained networks and the GA immediately plateau because the MDP limits the ability of either decision-making agent to prioritize high-priority targets over low-priority targets. However, the GA performs slightly better at this target density because it makes no assumptions about the underlying density. At the nominal size of **T**, $|\mathbf{T}| = 135$, the GA fails to match the reward of the networks trained using MCTS, which is discussed in the original comparison. However, at $|\mathbf{T}| = 270$, the GA begins to slightly outperform the trained networks again. The networks are trained for a nominal density of 135 total targets. Therefore, the state–action value outputs assume a given target density, which may result in suboptimal results. The GA makes no such assumptions and simply searches for the sequence of actions that will maximize the reward signal, albeit at a high computational cost. Hypothetically, the networks could be retrained using MCTS assuming the lower or higher target density, and the discrepancy in performance would disappear while maintaining the low-computational cost of the trained networks.

V. CONCLUSION

This work demonstrates the use of MCTS and state–action value function approximation using artificial neural networks for the AEOS scheduling problem. The dependence of average reward on the number of targets in the action space is explored for two separate backup operators in MCTS. For a constant target density, increasing the number of targets in the action space provides the agent with the ability to make tradeoffs between target priorities and target geometry, resulting in higher reward. This increase in mean reward for $|\mathbf{U}| = \{1, \ldots, 5\}$ is demonstrated for a density of 135 targets over three orbits. The maximization operator is shown to generate higher reward than the incremental averaging operator. This is due to the maximization operator's ability to ignore low-reward states close to the optimal trajectory. Furthermore, MCTS is compared with a genetic algorithm. MCTS with the maximization operator consistently outperforms the genetic algorithm at a fraction of the computation required by the genetic algorithm. MCTS demonstrates a 2%–5% increase in average reward at 10%–20% of the single-core wall clock time of the genetic algorithm.

The MCTS policies are then approximated using artificial neural networks. A hyperparameter search of the neural networks is presented, showing the dependence of the average reward on the number of hidden layers and number of nodes per hidden layer. The best performing neural networks for each number of targets in the action space are compared with MCTS using the maximization backup operator. The networks are shown to match the performance of MCTS. After training, the networks execute in three orders of magnitude less wall clock time than MCTS, which makes them a candidate for on-board planning and scheduling. Finally, the networks are deployed in environments with different numbers of total targets, demonstrating

robustness to changes in the size of the target request set. The performance of the networks exceeds that of the GA in the nominal target density. However, in the cases that the networks were not trained for, there is a slight reduction in performance when compared with the GA, which could be eliminated if the networks were retrained.

REFERENCES

[1] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, "Towards an application framework for automated planning and scheduling," in *Proc. IEEE Aerosp. Conf.*, 1997, pp. 375–386.

[2] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood, "Casper: Space exploration through continuous planning," *IEEE Intell. Syst.*, vol. 16, no. 5, pp. 70–75, Sep./Oct. 2001.

[3] S. Chien et al., "Using autonomy flight software to improve science return on earth observing one," *J. Aerosp. Comput., Inf., Commun.*, vol. 2, no. 4, pp. 196–216, 2005.

[4] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye, "Timeline-based space operations scheduling with external constraints," in *Proc. 28th Int. Conf. Automated Plan. Scheduling*, 2021, pp. 34–41.

[5] S. A. Chien et al., "Automated volcano monitoring using multiple space and ground sensors," *J. Aerosp. Inf. Syst.*, vol. 17, no. 4, pp. 214–228, 2020.

[6] S. Chien, D. Mclaren, J. Doubleday, D. Tran, V. Tanpipat, and R. Chitradon, "Using taskable remote sensing in a sensor web for Thailand flood monitoring," *J. Aerosp. Inf. Syst.*, vol. 16, no. 3, pp. 107–119, 2019.

[7] X. Wang, G. Wu, L. Xing, and W. Pedrycz, "Agile earth observation satellite scheduling over 20 years: Formulations, methods, and future directions," *IEEE Syst. J.*, vol. 15, no. 3, pp. 3881–3892, Sep. 2020.

[8] S. Nag, A. S. Li, and J. H. Merrick, "Scheduling algorithms for rapid imaging using agile Cubesat constellations," *Adv. Space Res.*, vol. 61, no. 3, pp. 891–913, Feb. 2018.

[9] D.-H. Cho, J.-H. Kim, H.-L. Choi, and J. Ahn, "Optimization-based scheduling method for agile Earth-observing satellite constellation," *J. Aerosp. Inf. Syst.*, vol. 15, no. 11, pp. 611–626, 2018.

[10] X. Chen, G. Reinelt, G. Dai, and A. Spitz, "A mixed integer linear programming model for multi-satellite scheduling," *Eur. J. Oper. Res.*, vol. 275, no. 2, pp. 694–707, 2019.

[11] G. Peng, R. Dewil, C. Verbeeck, A. Gunawan, L. Xing, and P. Vansteenwegen, "Agile earth observation satellite scheduling: An orienteering problem with time-dependent profits and travel times," *Comput. Oper. Res.*, vol. 111, pp. 84–98, 2019.

[12] W. Haijiao, Y. Zhen, Z. Wugen, and L. Dalin, "Online scheduling of image satellites based on neural networks and deep reinforcement learning," *Chin. J. Aeronaut.*, vol. 32, no. 4, pp. 1011–1019, 2019.

[13] X. Zhao, Z. Wang, and G. Zheng, "Two-phase neural combinatorial optimization with reinforcement learning for agile satellite scheduling," *J. Aerosp. Inf. Syst.*, vol. 17, no. 7, pp. 346–357, 2020.

[14] L. Wei, Y. Chen, M. Chen, and Y. Chen, "Deep reinforcement learning and parameter transfer based approach for the multi-objective agile earth observation satellite scheduling problem," *Appl. Soft Comput.*, vol. 110, 2021, Art. no. 107607.

[15] Y. He, L. Xing, Y. Chen, W. Pedrycz, L. Wang, and G. Wu, "A generic Markov decision process model and reinforcement learning method for scheduling agile earth observation satellites," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 3, pp. 1463–1474, Mar. 2022.

[16] A. Hadj-Salah, R. Verdier, C. Caron, M. Picard, and M. Capelle, "Schedule earth observation satellites with deep reinforcement learning," in *Proc. Int. Workshop Planning Scheduling Space (IWPSS)*, Jul. 2019, pp. 61–64.

[17] A. Hadj-Salah, J. Guerra, M. Picard, and M. Capelle, "Towards operational application of deep reinforcement learning to earth observation satellite scheduling," 2020. [Online]. Available: https://hal.science/hal-02925740

[18] A. Harris, T. Valade, T. Teil, and H. Schaub, "Generation of spacecraft operations procedures using deep reinforcement learning," *J. Spacecraft Rockets*, vol. 59, no. 2, pp. 611–626, Mar./Apr. 2022.

[19] A. P. Herrmann and H. Schaub, "Monte Carlo tree search methods for the Earth-observing satellite scheduling problem," *J. Aerosp. Inf. Syst.*, vol. 19, no. 1, pp. 70–82, 2021. [Online]. Available: https://doi.org/10.2514/1.I010992

[20] D. Eddy and M. Kochenderfer, "Markov decision processes for multi-objective satellite task planning," in *Proc. IEEE Aerosp. Conf.*, 2020, pp. 1–12.

[21] A. P. Herrmann and H. Schaub, "Autonomous on-board planning for earth-orbiting spacecraft," in *Proc. IEEE Aerosp. Conf.*, 2022, pp. 1–9.

[22] G. S. Center, "Near earth network users' guide," NASA, Greenbelt, MD, USA, Tech. Rep. 453-UG-002905, Mar. 2019.

[23] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*, 4th ed. Reston, VA, USA: AIAA Educ. Ser., 2018.

[24] P. W. Kenneally, H. Schaub, and S. Piggott, "Basilisk: A flexible, scalable and modular astrodynamics simulation framework," *AIAA J. Aerosp. Inf. Syst.*, vol. 17, no. 9, pp. 496–507, 2020.

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA:MIT Press, 2018.

[26] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo search," Univ. of Tartu, Tartu, Estonia, 2006. [Online]. Available: https://www.researchgate.net/profile/Csaba-Szepesvari/publication/228341626_Improved_montecarlo_search/links/09e415087fee8c7350000000/Improved-monte-carlosearch.pdf

[27] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. Cambridge, MA, USA:MIT Press, 2015, pp. 102–103.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[29] A. Rasamoelina, F. Adjailia, and M. Sinčák, "A review of activation function for artificial neural network," in *Proc. IEEE 18th World Symp. Appl. Mach. Intell. Informa. (SAMI)*, Jan. 2020, pp. 281–286.

Adam Herrmann received the B.S. degree in aerospace engineering from the University of Cincinnati, Cincinnati, OH, USA, in 2019 and the M.S. degree in aerospace engineering sciences from the University of Colorado, Boulder, CO, USA, in 2022. He is currently working towards a Ph.D. degree with the Autonomous Vehicle Systems (AVS) Laboratory, University of Colorado, Boulder, CO, USA.

His research focuses on the application of reinforcement learning to spacecraft planning and scheduling in Earth-orbiting and small body domains.

Hanspeter Schaub received the B.S., M.S., and the Ph.D degrees in aerospace engineering from Texas A&M University, College Station, TX, USA, in 1992, 1994, and 1998, respectively.

He is currently a Professor with the Aerospace Engineering Sciences Department and is the Schaden Leadership Chair with the University of Colorado, Boulder, CO, USA. He has more than 25 years of research experience, of which four years were spent with Sandia National Laboratories. His research interests include nonlinear dynamics and control, astrodynamics, relative motion dynamics, relative motion sensing, and spacecraft autonomy. In the last decade, he has developed the emerging field of charged astrodynamics.

Dr. Schaub has been the ADCS lead in the CICERO mission and the ADCS algorithm lead on a Mars Mission. He is an AAS and AIAA Fellow. He was the recipient of the AIAA/ASEE Atwood Educator Award and AIAA Mechanics and Control of Flight Award. He is the Editor-In-Chief for the *AIAA Journal of Spacecraft and Rockets*.