Autonomous On-board Planning for Earth-Orbiting Spacecraft

Adam Herrmann
Graduate Research Assistant
Department of Aerospace Engineering Sciences
University of Colorado, Boulder
Boulder, CO 80303
adam.herrmann@colorado.edu

Hanspeter Schaub
Glenn L. Murphy Endowed Chair
Department of Aerospace Engineering Sciences
University of Colorado, Boulder
Boulder, CO 80303
hanspeter.schaub@colorado.edu

***Abstract*—This work explores on-board planning and scheduling for the multi-target, single spacecraft Earth-observing satellite (EOS) scheduling problem. The problem is formulated as a Markov decision process (MDP) where the number of targets included in the state and action space is an adjustable parameter that may account for clusters of targets with varying priorities. As targets are passed or imaged, they are replaced in the state and action space with the next set of upcoming targets. Unlike prior EOS problem formulations, this work explores how the size of the state and action space can be reduced to produce optimal, generalized policies that may be executed on board the spacecraft in a fraction of a second. Performance of the agents is shown to increase with the number of targets in the state and action space. The number of imaged and downlinked targets stays relatively constant, but the reward increases significantly, demonstrating that the agents are prioritizing high priority targets over low priority targets.**

TABLE OF CONTENTS

1. 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 Earth-orbiting spacecraft, but the lessons learned can be 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 a suite of operations tools, sequenced, and uplinked to the spacecraft for open-loop execution. Efforts have been made to automate the ground-based process. The Automated Planning/Scheduling 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 (EO-1) mission as a part of the Autonomous Sciencecraft Experiment (ASE) 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, saving 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 the spacecraft. Combinations of these science objectives may also be considered, depending on the mission architecture. In literature, optimization-based approaches are commonly applied to the EOS scheduling problem. Spangelo et al. apply mixed integer programming to the EOS scheduling problem where operational decisions such as collecting images, downlinking data, and charging batteries are scheduled to maximize the amount of data downlinked to ground stations [7]. Nag et al. apply dynamic programming to solve a similar EOS scheduling problem, demonstrating a large performance improvement when the spacecraft is provided with discrete pointing options as opposed to nadir-pointing [8]. While these approaches find optimal solutions to the EOS 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.

Reinforcement learning has recently been posed as a candidate for the EOS scheduling problem due to its ability to generalize across different initial conditions, rapidly compute

978-1-6654-3760-8/22/$31.00

plans after training, and execute closed-loop on board spacecraft to dynamically respond to changes in the environment. Harris and Schaub 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 [9] [10]. They apply shielded proximal policy optimization (PPO) to demonstrate safe operations while maximizing data collection. Herrmann and Schaub 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 [11]. They show that Monte Carlo tree search (MCTS) and state-action value network regression can achieve near-optimal performance on this 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 towards 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 for the multi-target EOS scheduling problem and apply MCTS to generate optimized task schedules [12]. While the authors formulate a multi-target problem, they do not generalize the solutions to arbitrary initial conditions for on-board execution. This work applies Monte Carlo tree search methods to the multi-target EOS scheduling problem, generating optimal spacecraft plans that may be rapidly executed on board.

This work formulates an Earth-observing satellite scheduling problem in which multiple targets on the surface of the Earth must be 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 a Markov decision process where the number of targets included in the state and action space is explored to determine the effect on performance. The targets in the state and action space are replaced with the next set of upcoming targets as they are passed over or imaged by the agent. Monte Carlo tree search and state-action value neural network regression are applied to compute near-optimal state-action value approximations.

2. Problem Statement

Earth-Observing Satellite Scheduling Problem

In the Earth-observing 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 horizon, which is defined as the total amount of operational time considered for planning and scheduling. In this formulation of the EOS scheduling problem, a single spacecraft in 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 [13]. Over the course of a 3-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, 3 is the lowest). This set of all targets is referred to as $\mathbf{T}$. The goal is to maximize the weighted sum of the targets in $\mathbf{T}$ that are collected and downlinked. The planning horizon is broken into a number of discrete planning intervals where new scheduling decisions are made. The spacecraft schedules resource management and imaging tasks at each planning interval to achieve the goal. Figure 1 depicts this problem.

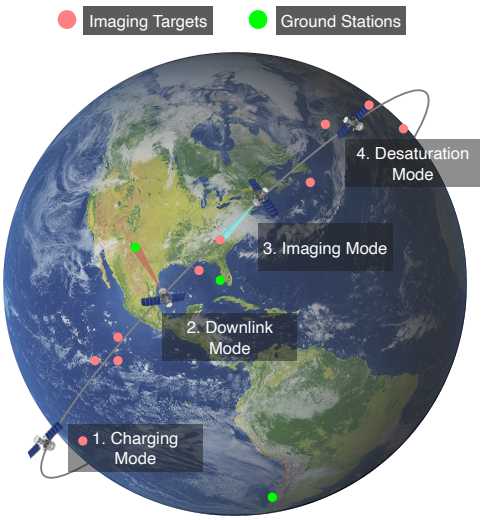


Figure 1: **Multi-target Earth-observing satellite scheduling problem.**

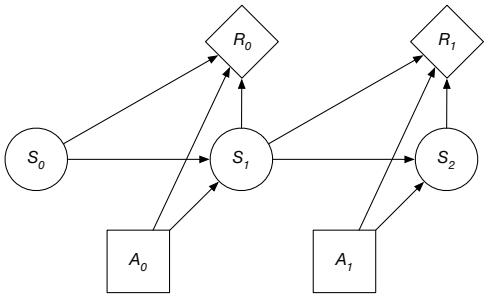


Figure 2: **Markov decision process.**

Markov Decision Process

The EOS scheduling problem is formulated as a Markov decision process, a sequential decision-making process in which an agent observes some state s_i and selects and action a_i following a policy $\pi : \mathcal{S} \times \mathcal{A}$, 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 $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. This process is demonstrated in Figure 2. Markov decision processes follow the Markov assumption, meaning the next state is conditionally dependent only on the current state and action. Mathematically, this may be stated as $T(s_{i+1}|s_i, a_i) = T(s_{i+1}|s_i, a_i, s_{i-1}, a_{i-1}, ..., s_0, a_0)$. All relevant state information for the purposes of maintaining this assumption must be included in the state space.

In this formulation of the EOS scheduling problem, the collection and downlink of individual targets is of particular interest. The relative geometry of the targets, length of the planning interval, length of the planning horizon, and limited spacecraft resources preclude the collection of every target. Therefore, the decision-making agent must make tradeoffs between the targets and resources to maximize the weighted sum of targets collected and downlinked. To achieve the Markov assumption, each target should be included in the state and action space. However, this significantly increases the complexity of the problem by increasing the size of the state and action space because there are 135 targets in set $\mathbf{T}$. Therefore, a subset of $\mathbf{T}$ that contains the next upcoming, unimaged targets should be considered to approximate this

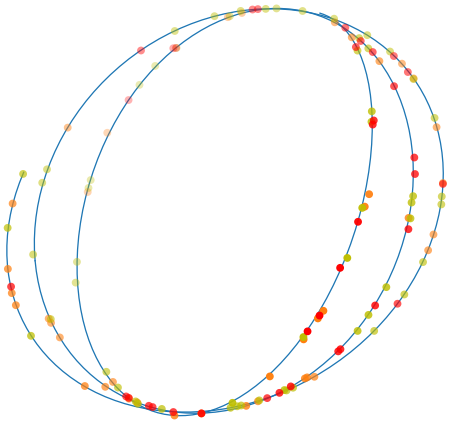


Figure 3: **Sample target distribution. Red is high priority. Orange is medium priority. Yellow is low priority.**

property. This subset is given in Equation 1, where J is the total number of targets in the state and action space and $\mathbf{D}$ is a subset of $\mathbf{T}$ 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 $\mathbf{T}$ are generated by sampling the positions of the spacecraft in the Earth-centered, Earth-fixed frame and projecting them onto the surface of the Earth. Before the projection, a small amount of noise is added to the spacecraft position so the target locations are not just ground-track coordinates. An example of the generated targets may be found in Figure 3. The red dots are targets, and the blue line is the projected orbit of the spacecraft in the Earth-centered, Earth-fixed frame. Due to the nature of the target generation, there are varying concentrations of targets with varying priorities that the agent will encounter.

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, $\mathcal{S}$, for the EOS scheduling problem may be found in the bulleted list below:

- ECEF spacecraft position, ${}^{\mathcal{E}}\mathbf{r}$
- ECEF spacecraft velocity, ${}^{\mathcal{E}}\mathbf{v}$
- Image tuples for targets $c_j \in \mathbf{U}$
  - Target position in the spacecraft Hill frame, ${}^{\mathcal{H}}\mathbf{r}_j$
  - Priority, p_j
  - Imaged indicator, w_j
  - Downlinked indicator, d_j
- L^2 norm of attitude error, $||\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}||$
- L^2 norm of attitude rate, $||{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}||$
- Reaction wheel speeds, $\boldsymbol{\Omega}$
- Battery charge, z
- Eclipse indicator, k
- Stored data in buffer, b
- Data transmitted, h
- Ground station access indicator g_i, $i = 1 : M$

The left-superscript on a vector denotes the coordinate frame relative to which vector components are taken [14]. The body-fixed frame is called $\mathcal{B}$. The position and velocity of the spacecraft expressed in an Earth-centered, Earth-fixed (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 space, a target tuple is also included that contains the target position expressed in the spacecraft Hill frame $\mathcal{H}$, the target priority, whether or not it has been imaged, and whether or not it has been downlinked. 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 $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$, attitude rate ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$, 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, amount of data downlinked over the planning interval, and ground station access indicators 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 is normalized by the radius of the Earth, and the velocity is normalized by the velocity of a circular orbit at the Earth's surface. The target position is normalized by the radius of the Earth. 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.

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. 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 by the bulleted list below:

- Charge
- Desaturate
- Downlink
- 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 i 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.

Transition Function—Due to the continuous dynamics of the EOS 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 given in Equation 2. A generative model simply returns a new state s_{i+1} and reward r_i by sampling an underlying distribution,

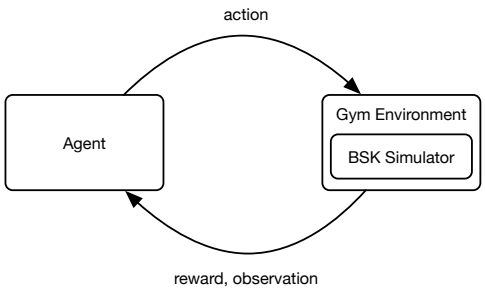


Figure 4: **Gym environment interface** [11].

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 [15] 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 the simulation. This is depicted in Figure 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 six minutes with a time step of 1 second. Afterwards, 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 CU Boulder[2] . The Basilisk simulation described within this work is an iteration of the simulator used in past work [11]. The base simulator for the EOS 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.

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:

$$r_i = \begin{cases} -10 & \text{if failure} \\ \sum_j^{|\mathbf{T}|} H_j(d) & \text{if !failure and } a_i \text{ is downlink} \\ 0.1 H_j(w) & \text{if !failure and } a_i \text{ is image } c_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

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:

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

A function $H_j(x)$ 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.

$$H_j(x) = (1/p_j) \text{ if } !x_{j_i} \text{ and } x_{j_{i+1}} \tag{5}$$

If the downlink mode is initiated and a resource management failure does not occur, the $H_j(d)$ function for all targets is computed and summed using the downlink state. 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 this condition is true for, 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 $H_j(w)$ operator for target c_j is computed, returned, and scaled by 0.1. 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 Monte Carlo tree search.

Target Replacement—After each step i through the environment, the targets in $\mathbf{U}$ are checked to see if they have been imaged or passed. If so, these targets are added to $\mathbf{D}$, and $\mathbf{U}$ is updated and added to the action space. While this does add non-stationarity to the problem, there is a value for $J = |\mathbf{U}|$ that will render this impact negligible because the added information of more targets will only marginally improve stationarity while increasing problem complexity and required training time. In stationary Markov decision processes, the underlying dynamics (i.e transition function) do not change with time. If each target were included in the state and action space, the underlying dynamics would appear stationary to the agent. By only including a subset of targets in the state and action space, the underlying dynamics appear non-stationary because the available targets are randomly changing. An agent operating within this MDP will therefore have difficulty computing the expected reward for each action because it has no state information on targets encountered far in the future of the planning horizon.

3. METHODS

Solving Markov Decision Processes

Solving Markov decision process 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.

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

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 [16]. It may be defined recursively using the Bellman operator, as shown below:

$$V^*(s_i) = \max_{\text{a}} \left(R(s_i, a_i) + \gamma \sum_{s_{i+1} \in \mathcal{S}} T(s_{i+1} | s_i, a_i) V^*(s_{i+1}) \right) \tag{7}$$

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 $Q^*(s_i, a_i)$.

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

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

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 implements. Adding supervised learning allows for the policies found by the online algorithms to generalize across the state space.

Monte Carlo Tree Search

Monte Carlo tree search 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 [17] [18]. At each step through the environment, MCTS computes the optimal action by following the process in Figure 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 $\hat{Q}(s, a)$ with an incremental averaging operator, where q is the return after simulation and rollout.

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

This process is repeated for a specified number of simulations-per-step. Afterwards, the action that maximizes the state-action value estimate is selected and the agent takes a step forward in the environment.

Rollout Policy—A safety MDP and rollout policy are derived as described by Herrmann and Schaub [11]. The safety MDP discretizes the state space to reduce dimensionality to several safety states:

$$\mathcal{S}_{\text{safety}} : \{\text{Tumbling, Low Power, Saturated, Buffer Full}\} \tag{12}$$

The safety states take a value of true or false depending on whether or not the relevant resource state variables in the original MDP are above or below a safety limit. A rollout policy is generated for this 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 nominal. However, if a ground station is accessible, the downlink mode is initiated instead.

Table 1: **Neural network hyperparameters.**

Parameter	Value
Nodes Per Hidden Layer	{50, 100}
Hidden Layers	{3, 4, $\cdots$, 8, 9}
Activation Function	Leaky ReLU
α	0.1
Dropout	0.1
Epochs	10,000
Batch Size	45,000
Loss Function	Mean Squared Error

Supervised Learning

To create a state-action value function that is generalizable over a range of initial conditions, Monte Carlo tree search 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. [11]. Since 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, as shown in Figure 6.

Afterwards, 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, $Q_\theta(s, a)$. This process is demonstrated in Figure 7. 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.

$$\pi(s_i) = \arg\max_{a_i} Q_\theta(s_i, a_i) \tag{13}$$

The benefit of a neural network representation of the state-action value network is that it may be executed in fractions of a second on board the spacecraft.

4. RESULTS

State-Action Value Function Approximation

To determine the artificial neural network size for approximating the state-action value estimates generated by MCTS, a hyperparameter search is conducted over the number of nodes and hidden layers in the network. The number of hidden layers is increased from three to nine. Three network widths are considered - 50, 100, and 200 nodes. MCTS is used to generate 45,000 data points (1,000 unique planning horizons solved). 90% of the data is used for training, and 10% of the data is used for validation. Other network parameters are included in Table 1. For this hyperparameter search, the size of $\mathbf{U}$ is two. In other words, the next two upcoming targets are included in the action space. Experiments are performed for the other sizes of $\mathbf{U}$ but are not shown here for brevity.

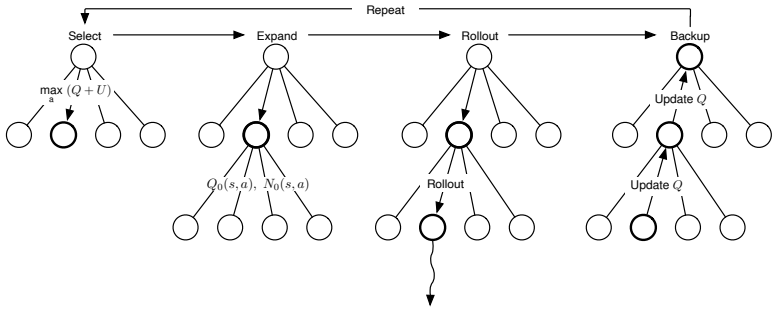


Figure 5: **Monte Carlo tree search** [11].

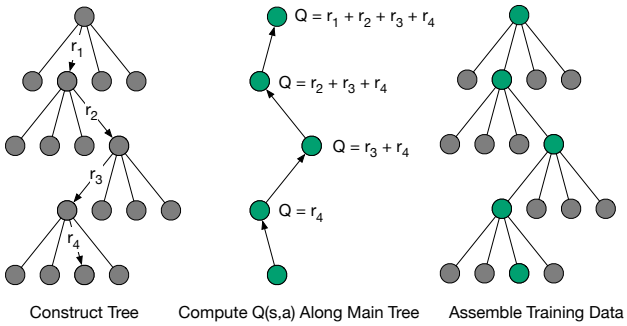


Figure 6: **Updating the state-action value function** [11].

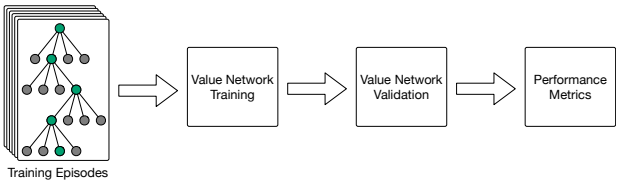


Figure 7: **Supervised learning process** [11].

To validate the performance of each trained neural network, the state-action value networks are used to generate the policy described in Equation 13, which is executed on a standard set of initial conditions. Figure 8 displays the results for 50 nodes per hidden layer, and Figure 9 displays the results for 100 nodes per hidden layer. In each plot, the dotted black line is the mean reward from MCTS ($N = 80$ samples). The distribution of the reward for each trained agent is given as a box and whisker plot with where the triangle is the mean and green line is the median ($N = 100$ samples). The set of initial conditions remains constant between each experiment.

For each number of nodes per hidden layer, the performance

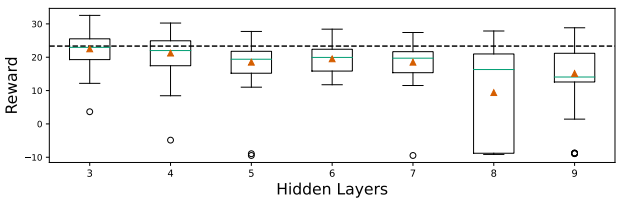


Figure 8: **Reward as hidden layers increase. 50 nodes per layer. Two targets.**

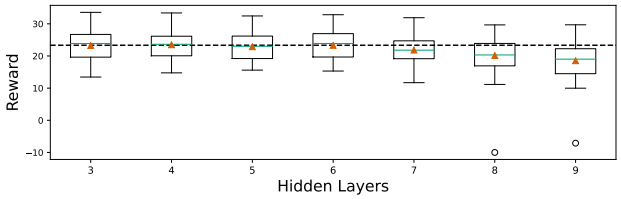


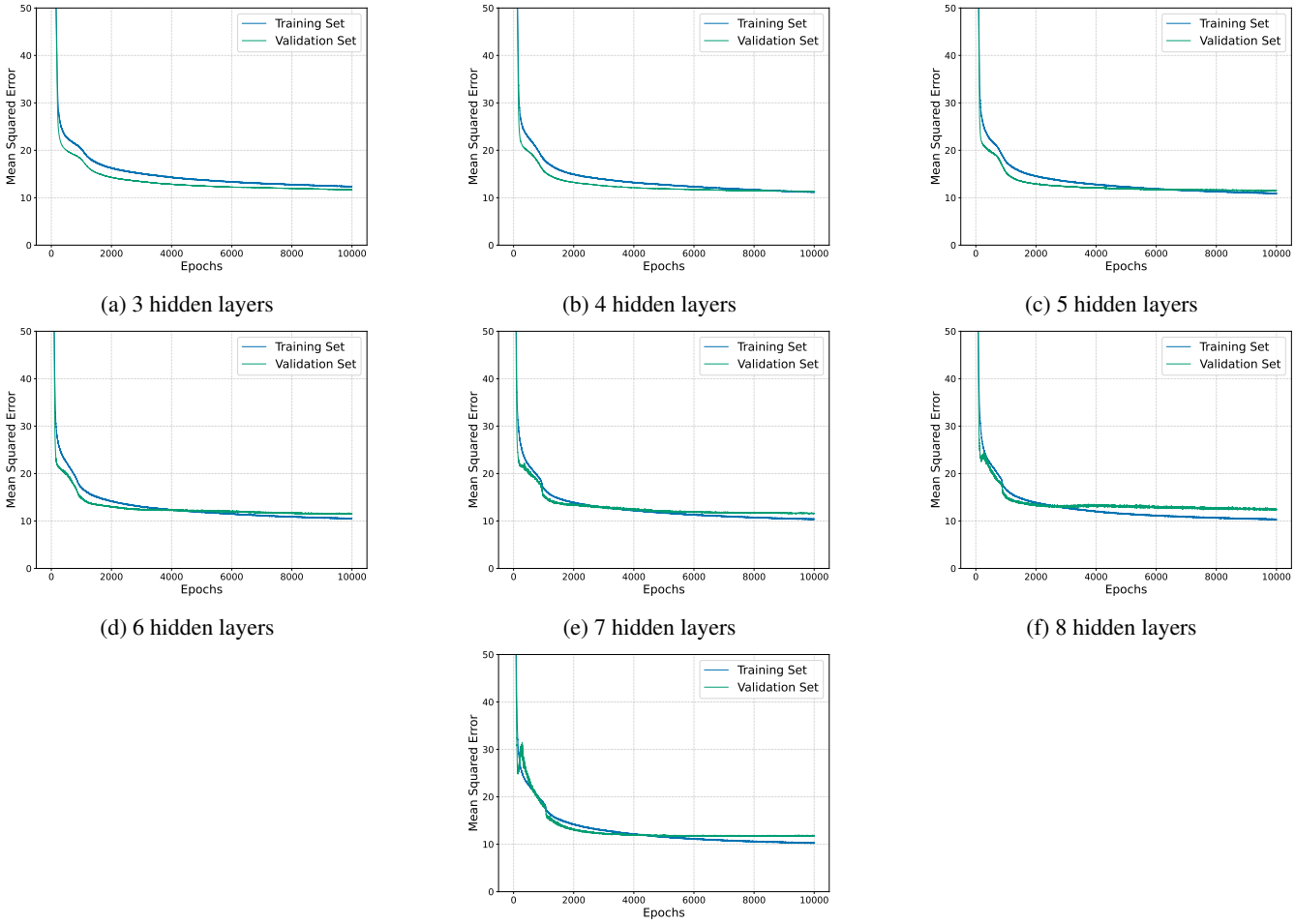
Figure 9: **Reward as hidden layers increase. 100 nodes per layer. Two targets.**

typically matches Monte Carlo tree search at either 3 or 4 hidden layers. However, once the number of hidden layers exceeds this point, the performance begins to degrade until resource management failures begin to occur. This is most evident for 8 or 9 hidden layers. Overfitting is likely the largest contributing factor to the degradation in performance. In Figure 10, the mean squared error loss vs. training epoch is plotted for 100 nodes per hidden layer. As the overfitting on the training data set becomes more pronounced, the state-action value networks begin to produce less reward on average and eventually begin producing resource management failures. This is most notable for 7, 8, and 9 hidden layers. A duplicate experiment is performed for 50 nodes per hidden layer that shows the most overfitting occurring at 8 hidden layers, where performance suffers the most in Figure 8.

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 six-minute planning intervals.

An experiment is performed in which the number of targets in $\mathbf{U}$ is increased from one through four. For each $|\mathbf{U}|$, the MCTS hyperparameters that balance performance and execution time are selected. For all sizes of $\mathbf{U}$, an exploration constant of 10 is selected. The number of simulations-per-step for each $|\mathbf{U}|$ is linearly increased from 15 to 30, with $|\mathbf{U}| = 1$ at 15 simulations-per-step and $|\mathbf{U}| = 4$ at 30 simulations-per-step. The increase in simulations-per-step is due to the increase in problem complexity due to additional actions in the action space. The number of times future state-



(a) 3 hidden layers

(b) 4 hidden layers

(c) 5 hidden layers

(d) 6 hidden layers

(e) 7 hidden layers

(f) 8 hidden layers

(g) 9 hidden layers

Figure 10: Overfitting as the number of hidden layers increase. 100 nodes per layer. Two targets.

action pairs are visited is on the order of:

$$\frac{1}{|\mathcal{A}|^{\text{depth}}} \tag{14}$$

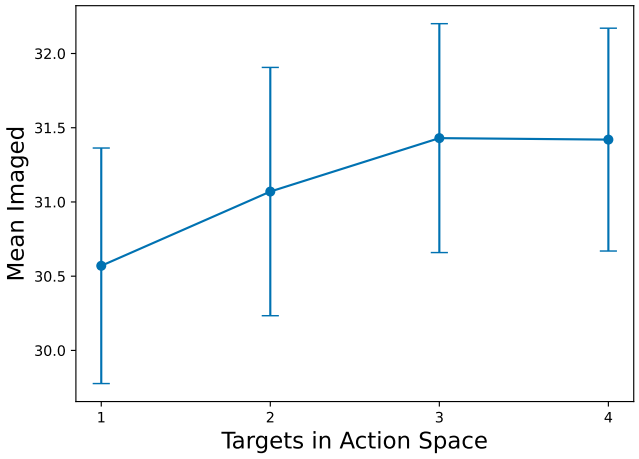
The depth is the depth of the search. To equalize exploration between the different experiments, the number of simulations-per-step is marginally increased to account for this decay. Futhermore, past work has shown that after a certain point, additional simulations-per-step in the EOS scheduling problem do not result in a large difference in the quality of MCTS solutions [11]. Therefore, the smallest number of simulations-per-step is selected for each $|\mathbf{U}|$ such that an MCTS performance plateau is achieved. These hyperparameter combinations are used to generate a data set that is regressed over using various neural network sizes which are described in Section 4.

Once the state-action value network hyperparameterization is performed for each $|\mathbf{U}|$ as described in Section 4, the best state-action value networks for each $|\mathbf{U}|$ are selected for comparison. In Figure 11, the average reward for each size of $\mathbf{U}$ are plotted, along with the 95% confidence intervals for each. The mean number of imaged and downlinked targets are also plotted. As the size of $\mathbf{U}$ increases, so does the reward. However, the reward begins to plateau at $|\mathbf{U}| = 4$, which is expected. At a certain point, adding more targets to the action space does not add more value. As reward increases, the number of targets imaged and downlinked increases, but the increase in targets imaged and downlinked is only partially responsible for the increase in reward, and both of these metrics plateau more quickly than the reward. The majority of the increase in reward is due to the state-action value networks prioritizing high-priority targets over low-priority targets.

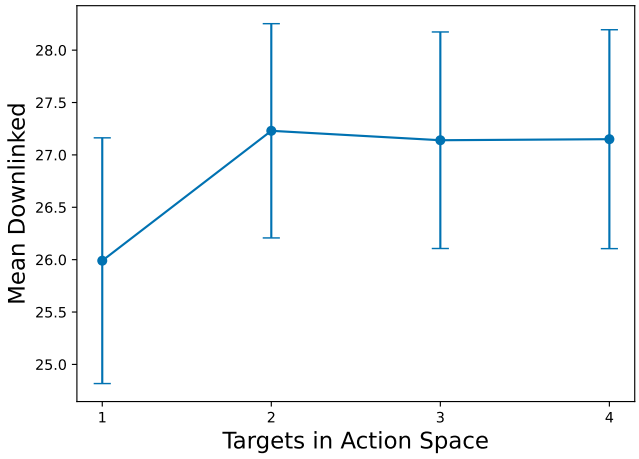
5. CONCLUSIONS

This work demonstrates the use of Monte Carlo tree search (MCTS) and state-action value function regression using artificial neural networks for the multi-target Earth-observing satellite (EOS) scheduling problem. The performance of state-action value function approximators are benchmarked for different combinations of hidden layers and nodes per hidden layer and are shown to match the performance of MCTS. However, overfitting is shown to negatively impact performance of the trained agents and cause resource constraint violations.

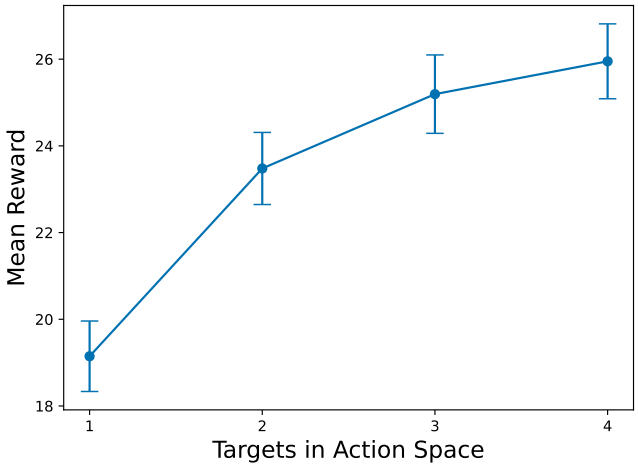
This work also shows that for a constant target density, increasing the number of targets in the action space provides the agent with the ability to make trade-offs between target priorities and target geometry, resulting in higher reward,



(a) Number of imaged targets.



(b) Number of downlinked targets.



(c) Reward.

Figure 11: Performance vs. targets in action space.

imaged targets, and downlinked targets on average. This increase in mean reward, imaged targets, and downlinked targets for $|\mathbf{U}| = \{1, 2, 3, 4\}$ is demonstrated for density of 135 targets over three orbits. Future work should study how the size of $\mathbf{U}$ affects performance when different target densities are considered, and the impact of using target densities not trained for should also be studied.

Acknowledgments

This work is supported by a NASA Space Technology Graduate Research Opportunity (NSTGRO) grant, 80NSSC20K1162.

References

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

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

[3] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davis, D. Mandl, S. Frye, B. Trout, S. Shulman, and D. Boyer, "Using autonomy flight software to improve science return on earth observing one," *Journal of Aerospace Computing, Information, and Communication*, 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 *Twentieth International Conference on Automated Planning and Scheduling*, 2010.

[5] S. A. Chien, A. G. Davies, J. Doubleday, D. Q. Tran, D. Mclaren, W. Chi, and A. Maillard, "Automated volcano monitoring using multiple space and ground sensors," *Journal of Aerospace Information Systems*, 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," *Journal of Aerospace Information Systems*, vol. 16, no. 3, pp. 107–119, 2019.

[7] S. Spangelo, J. Cutler, K. Gilson, and A. Cohn,

"Optimization-based scheduling for the single-satellite, multi-ground station communication problem," *Computers and Operations Research*, vol. 57, May 2015.

[8] S. Nag, A. S. Li, and J. H. Merrick, "Scheduling Algorithms for Rapid Imaging Using Agile Cubesat Constellations," *Advances in Space Research*, vol. 61, no. 3, pp. 891–913, Feb. 2018.

[9] A. Harris and H. Schaub, "Deep on-board scheduling for autonomous attitude guidance operations," in *AAS Guidance, Navigation and Control Conference*, Breckenridge, CO, January 30 – Feb. 5 2020, aAS 02-117.

[10] ——, "Spacecraft command and control with safety guarantees using shielded deep reinforcement learning," in *AIAA SciTech*, Orlando, Florida, January 6–10 2020.

[11] A. P. Herrmann and H. Schaub, "Monte carlo tree search methods for the earth-observing satellite scheduling problem," *Journal of Aerospace Information Systems*, pp. 1–13, 2021. [Online]. Available: https://doi.org/10.2514/1.I010992

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

[13] G. S. Center, "Near earth network users' guide," National Aeronautics and Space Administration, Greenbelt, MD, Tech. Rep., March 2019.

[14] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*, 4th ed. Reston, VA: AIAA Education Series, 2018.

[15] P. W. Kenneally, H. Schaub, and S. Piggott, "Basilisk: A flexible, scalable and modular astrodynamics simulation framework," *AIAA Journal of Aerospace Information Systems*, vol. 17, no. 9, pp. 496–507, 2020.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[17] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved monte-carlo search," *Univ. Tartu, Estonia, Tech. Rep*, 2006.

[18] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. Massachusetts Institute of Technology, 2015, ch. Sequential Problems, pp. 102–103.

BIOGRAPHY

Adam Herrmann *received his B.S. in Aerospace Engineering from the University of Cincinnati and is a current Ph.D. student at the University of Colorado, Boulder, in the Autonomous Vehicle Systems (AVS) Laboratory. His research focuses on the application of reinforcement learning to spacecraft planning and scheduling in Earth-orbiting and small body domains.*

Hanspeter Schaub *is a professor in the Aerospace Engineering Sciences department and the Glenn L. Murphy Chair in Engineering at the University of Colorado. He has over 20 years of research experience, of which 4 years are at Sandia National Laboratories. His research interests are in 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, and has won the AIAA/ASEE Atwood Educator award, as well as the AIAA Mechanics and Control of Flight award. He currently serves as the Editor-In-Chief for the AIAA Journal of Spacecraft and Rockets.*