(Preprint) AAS 21-228

AUTONOMOUS SPACECRAFT TASKING USING MONTE CARLO TREE SEARCH METHODS

Adam Herrmann* **and Hanspeter Schaub†**

This work explores the use of Monte Carlo tree search (MCTS) and state-action value network regression to solve the single-spacecraft, multi-ground station Earth-observing satellite scheduling problem. Both algorithms are explored for their potential use as ground-based and on-board planning tools. An in-depth hyperparameter search is conducted for Monte Carlo tree search on the basis of performance, safety, and downlink opportunity utilization. A hyperparameter search is also conducted on the neural network architectures, and the general behavior of each network is explored to determine and validate learned behavior. Furthermore, each algorithm is compared to a genetic algorithm to determine the optimality gap and compare and contrast the use of reinforcement learning algorithms to classical optimization techniques. MCTS is shown to compute near-optimal solutions in comparison to the genetic algorithm. Furthermore, the state-action value networks are shown to match or exceed the performance of MCTS in five orders of magnitude less execution time, showing promise for execution on-board spacecraft.

INTRODUCTION

Autonomous spacecraft planning is becoming an enabling capability for future spacecraft missions. As the number of Earth-orbiting constellations increases, current operations infrastructure will be stretched to meet the operational needs of these missions. Furthermore, some environments and mission architectures will require autonomous exploration as the round-trip light-time communication delay will constrain maneuvers and prevent opportunistic science collection. Several challenges must be addressed to enable on-board planning. On-board planning algorithms must be safe and verifiable in order to guarantee the longevity of spacecraft. Additionally, the computational overhead of on-board planning algorithms must be addressed. While advances in radiation hardened processors can be expected to increase on-board computational resources, current missions rely on limited processing solutions such as the RAD750. Lastly, on-board solutions for operations should utilize the full capability of the spacecraft. Without sacrificing safety or exceeding computational limitations, planning solutions should be near-optimal or optimal. This research focuses on addressing these challenges - safety, computational overhead, and optimality - in the context of a single Earth-orbiting spacecraft downlinking data to multiple ground stations. This is a problem commonly referred to as the single-spacecraft, multi-ground station Earth-observing satellite (EOS) scheduling problem.

State-of-the-art solutions to the EOS scheduling problem for real spacecraft missions are typically ground-based. Spacecraft plans are generated on the ground, sequenced, and uplinked to the spacecraft for execution. A notable example, which automates this process, is the Automated Planning/Scheduling Environment (ASPEN) software architecture.[1] ASPEN is an autonomous, ground-based planning and scheduling architecture that can be applied to a variety of spacecraft missions. Significant work has also been performed to develop on-board solutions that give spacecraft the ability to iteratively repair a ground-based plan if a resource constraint violation or unexpected science opportunity occurs. CASPER addresses this need by continually checking an existing plan for resource constraint violations and modifying the plan on-board the spacecraft

*PhD Student, Ann and H.J. Smead Department of Aerospace Engineering Sciences, University of Colorado, Boulder, Boulder, CO, 80309. AIAA Member.

†Glenn L. Murphy Chair of Engineering, Ann and H.J. Smead Department of Aerospace Engineering Sciences, University of Colorado, Boulder, 431 UCB, Colorado Center for Astrodynamics Research, Boulder, CO, 80309. AAS Fellow, AIAA Fellow.

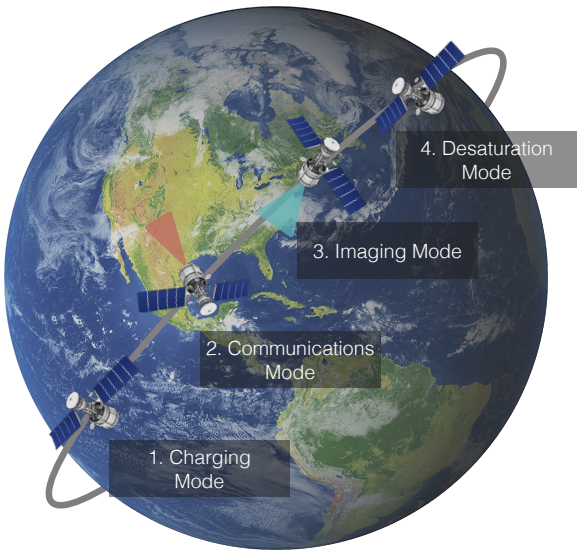


Figure 1: Scheduling of Flight Modes

if necessary.[2] CASPER has been applied to several missions such as EO-1[3] and IPEX,[4] in combination with science detection algorithms to mark future science targets, to demonstrate autonomous science operations. While state-of-the-art solutions to spacecraft planning have decreased the burden on operators, algorithms with more control over operational decisions are required for full autonomy.

In literature, many solutions to the EOS scheduling problem follow an optimization-based approach. Optimization-based approaches find high-quality, optimal solutions, but are brittle to uncertainty and do not generalize or scale well for complex scenarios with many targets, ground stations, and spacecraft. Spangelo et al. formulate the Earth-observing satellite scheduling problem as an optimization problem where operational decisions such as collecting imagery, downlinking data, and charging batteries are considered to maximize downlinked data and keep the spacecraft within its resource constraints.[5] Cho et al. apply a two-step binary linear programming algorithm to solve the EOS scheduling problem for a constellation of agile spacecraft imaging a set of user-defined targets by scheduling imaging and downlink tasks.[6] Optimization-based solutions are executed open-loop on board spacecraft based on modeling done on the ground. Replanning must occur in the event of resource constraint violations due to mismodeling. Future planning solutions should take into account observations from the spacecraft in a closed-loop implementation to minimize the impacts of mismodeling without having to re-run computationally intensive optimization algorithms.

Recent work has proposed Reinforcement Learning (RL) as a viable alternative to state-of-the-art spacecraft operations and guidance algorithms due to its speed and ability to generalize across training data. In the small-body domain, RL is well-suited to solve many challenging problems that require fault detection or rigorous handling of unknown or uncertain dynamics. Gaudet et al. develop an adaptive guidance system using recurrent neural networks to respond real-time to faults or unknown dynamics in landing scenarios.[7] Hockman and Pavone apply policy iteration for hopping rover motion planning on the surface of an asteroid, rigorously handling the uncertainty of the dynamics on the surface of small bodies.[8] Chan and Agha-mohammadi solve the small body mapping problem by formulating it as a partially-observable Markov Decision Process and applying the REINFORCE algorithm to maximize map quality by learning when to execute thrust and imaging commands.[9] In the EOS domain, Harris et al. train Deep Q-Learning or shielded Proximal Policy Optimization agents on the ground and execute them real-time on board simulated spacecraft.[10–12] These techniques allow operators to forgo the arduous ground-based planning process by developing generalized plans through RL policies or value functions that may be rapidly executed on board spacecraft. However, Deep Q-Learning requires many modifications to achieve good performance as it suffers from maximization bias, single-step learning, etc.[13] While policy gradient methods offer several benefits such as stochastic poli-

cies and good performance in continuous state and action spaces, only convergence to local maxima can be guaranteed, which can result in training and initialization sensitivities. Furthermore, stochastic policies can take unsafe actions and necessitate the use of a shield to safely bound the behavior of the agent.

To address these problems, this work focuses on a class of techniques that use Monte Carlo tree search (MCTS) to compute optimal policies that can be regressed over using neural networks to produce generalized, deterministic, and optimal behavior for on-board spacecraft planning. Kocsis et al. propose a variant of MCTS, the Upper Confidence Bound for Trees (UCT), that will converge to the globally optimal action as the number of simulations-per-step approaches infinity.[14] Shah et al. propose a variant of UCT with a polynomial bonus, and pose that this algorithm holds the property that Kocsis et al. claim.[15] Theoretical guarantees aside, Silver et al. demonstrate that MCTS methods, specifically AlphaZero, can achieve super-human performance in the game of Go.[16] MuZero, a model-based variant of AlphaZero, also demonstrates the power of MCTS methods on a myriad of canonical reinforcement learning problems.[17] Fedeler et al. extend these methods to the space situational awareness domain, demonstrating telescope tasking for tracking space objects using a novel MCTS method.[18] While these methods have demonstrated promise on real-world problems, none have considered the use of MCTS for spacecraft operations with resource constraints as this paper does. Building upon previous work,[19] this work explores neural network value regression over optimal or near-optimal policies generated by UCT, which can be rapidly executed on board spacecraft. The generalizability of the neural network architectures is explored to determine how well the neural networks compute planning solutions within the distributions of the training data. The execution times of each algorithm presented in this work are also compared to determine which may be useful for on-board execution and which are better suited as ground-based planning tools.

In this paper, the implementation of EOS scheduling problem is first described in detail. Then, the problem is formulated as a Markov Decision Process. The MCTS algorithm and associated rollout policies are also described, as are the techniques used to regress over the state-action value pairs generated by MCTS. The results from a hyperparameter search for both Monte Carlo tree search and the state-action value networks are shown and discussed. Each algorithm is evaluated on the basis of average reward, downlink utilization, and resource management success. The learned behavior of the state-action value networks is also explored. Finally, a comparison to genetic algorithms is made to determine the optimality gap and compare and contrast to optimization-based planning techniques.

PROBLEM STATEMENT

Earth-Observing Spacecraft Simulation

In this formulation of the Earth-observing satellite scheduling problem, a satellite in a 500-km orbit makes operational decisions to collect and downlink science data to any of seven different ground stations around the Earth. As shown in Figure 1, operational decisions include sun-pointing charging, nadir-pointing imaging, nadir-pointing communications, and reaction wheel desaturation. The Earth-observing satellite scheduling problem is simulated using the Basilisk* Astrodynamics Software Framework.[20] A complete diagram of the associated Basilisk modules may be found in Figure 2. Each module in the diagram represents a separate, modularized block of code that receives inputs from other modules, performs computations, and sends outputs to modules subscribed to its messages. The modularity and speed of Basilisk allows for a high-fidelity simulation with cross-couplings between disparate spacecraft subsystems to be constructed and quickly executed. Furthermore, the framework provides the opportunity to simulate flight software used on board spacecraft, which lends itself to future autonomy work that may one day fly on board a real spacecraft.

The Basilisk simulation includes a full attitude control system to simulate a representative spacecraft mission where many systems are coupled to the attitude dynamics. Hill-pointing and inertial reference frames are switched between based on the specific mode and passed to an attitude error computation module, which then passes the attitude error to an MRP feedback control law. The MRP feedback control law sends torque commands to reaction wheels, which change the dynamics of the spacecraft. The reaction wheels are modeled after the Honeywell HR16 reaction wheels. A momentum dumping module is also implemented, which

*http://hanspeterschaub.info/basilisk

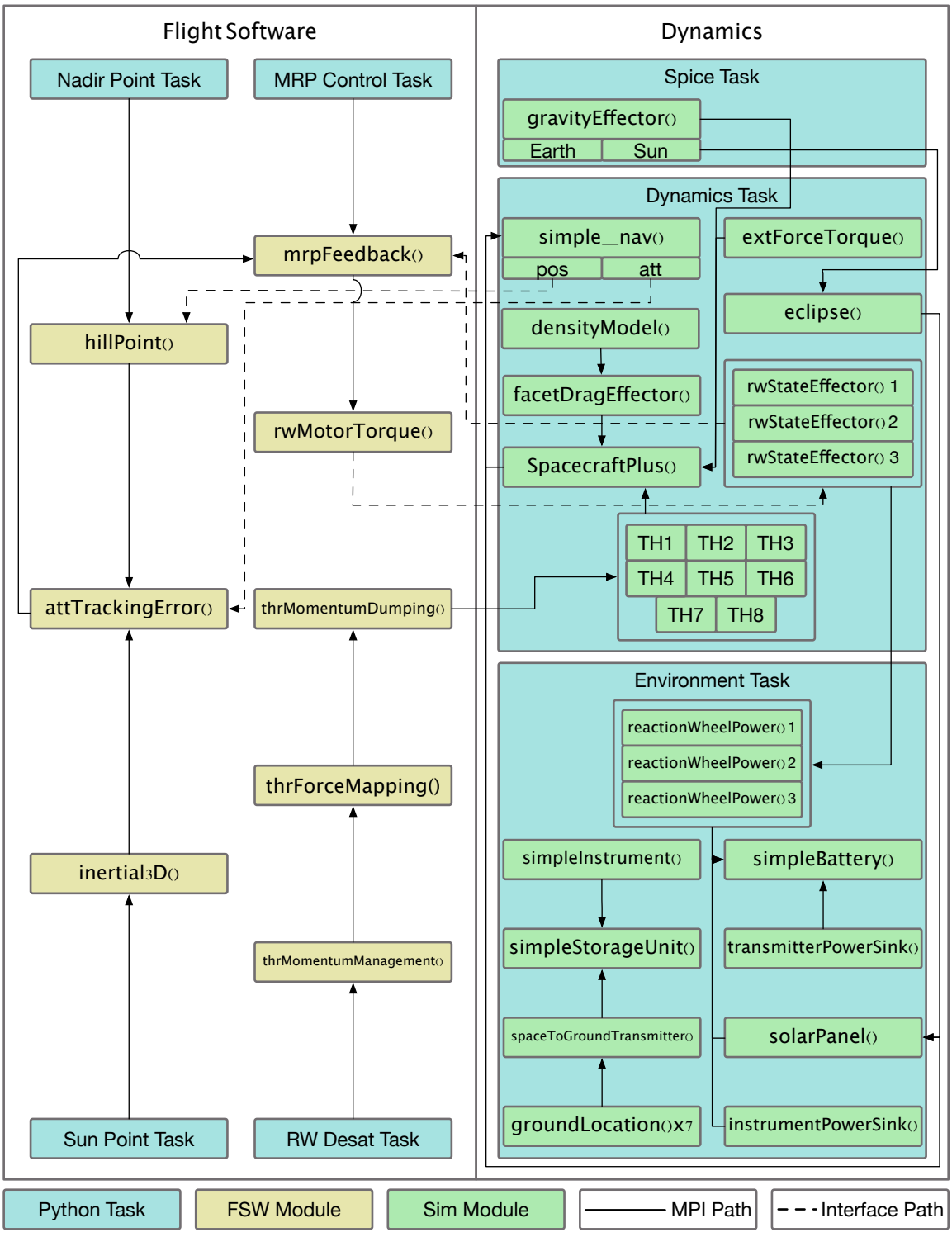


Figure 2: Basilisk Simulation

Table 1: Ground Station Parameters

Location	Latitude	Longitude	Elevation (m)	Min. Elevation Angle
Boulder, CO (USA)	40.0150 N	105.2705 W	1624 m	10 deg
Ka Lae, HI (USA)	19.8968 N	155.5828 W	9.0 m	10 deg
Merritt Island, FL (USA)	28.3181 N	80.6660 W	0.9144 m	10 deg
Singapore, Malaysia	1.3521 N	103.8198 E	15.0 m	10 deg
Weilheim, Germany	47.8407 N	11.1421 E	563 m	10 deg
Santiago, Chile	33.4489 S	70.6693 W	570 m	10 deg
Dongara, Australia	29.2452 S	114.9326 E	34.0 m	10 deg

maps reaction wheel momentum to thrust commands to dump momentum from the system. The thrusters are modeled after the Moog Monarc-1 thrusters. Faceted drag, random external torque, third-body gravity effects, and J2 perturbations are implemented to build up momentum in the reaction wheels over time or change the orbital parameters, albeit almost negligibly.

A power system is simulated in Basilisk, leveraging Basilisk's high-fidelity dynamics capabilities to accurately compute power consumption and generation. Simulated solar panels generate power based on incidence angle, panel area, and efficiency. Eclipse effects are also considered. Generated power is stored in a modeled battery, and the imager, transmitter, and reaction wheels all consume power from the battery. Similarly, an on-board data management system is modeled. An instrument generates data during the imaging mode, which is stored in a data buffer. In the communications mode, a transmitter downlinks data from the buffer to ground stations located on the Earth. The spacecraft antenna is omni-directional, and it is assumed that nadir pointing will suffice to communicate with a ground station that is within line of sight. The location and parameters of each ground station are provided in Table 1. The ground stations are selected from a list of stations utilized by NASA's Near Earth Network.[21] A Boulder, Colorado ground station is also implemented. The ground stations are selected from the list such that they are located within the minimum and maximum boundaries of the randomly generated orbits, which are discussed in more detail in the Gym Environment section.

The spacecraft simulation parameters may be found in Table 2. The subsystem specifications were sized to provide a representative, but challenging mission scenario. Specifically, the small storage capacity in comparison to the instrument baud rate creates a mission scenario in which the spacecraft must continually alternate between collecting and downlinking data at a relatively high frequency.

In Table 3, the details of the four flight modes of the spacecraft are provided. The agent makes operational decisions by selecting the next flight mode to enter. The Basilisk tasks and models are provided in the left-hand column. Enabling or disabling tasks and setting different models on or off dictates the behavior of each flight mode. During the observation mode, the spacecraft points in the nadir direction and the instrument data and power models are turned on. During the downlink mode, the spacecraft points in the nadir direction and the transmitter data and power models are turned on. The transmitter will only downlink data if a ground station is accessible. Otherwise, it remains on and consumes power until a ground station becomes available. In the charging mode, the spacecraft points the solar panels in the direction of the sun. All power and data models are turned off. Lastly, in the desaturation mode, the spacecraft points its solar panels towards the sun while the thrusters are used to dump momentum from the reaction wheels. Like the charging mode, all power and data models are turned off.

Gym Environment

The Basilisk simulation described in the previous section is wrapped in a `Gym` environment, building upon previous open-source work* from the AVS Laboratory. `Gym` environments provide a standardized interface for test environments that decision-making algorithms can act on and learn from†. A simple depiction of this

*http://github.com/atharris/basilisk_env

†https://gym.openai.com/

Table 2: Spacecraft Parameters

General Spacecraft Parameters	
Mass	330 kg
Dimensions	1.38 x 1.04 x 1.58 m
Power System	
Solar Panel Area	1.0 m^2
Solar Panel Efficiency	0.20
Instrument Power Draw	30 W
Transmitter Power Draw	15 W
Battery Capacity	80 Whr
Attitude Control System	
Max Wheel Speeds	6000 RPM
Max Momentum	50 Nms
Max Wheel Torque	0.2 Nm
Max Thrust	0.9 N
Thruster Min On Time	0.02 s
Data & Communications System	
Data Buffer Storage Capacity	1 GB
Instrument Baud Rate	4 Mbps
Transmitter Baud Rate	4 Mbps

Table 3: Flight Modes

	Modes			
Basilisk Tasks & Models	Observation	Downlink	Charge	Desaturation
Nadir Point Task	Enabled	Enabled	Disabled	Disabled
Sun-Point Task	Disabled	Disabled	Enabled	Enabled
MRP Control Task	Enabled	Enabled	Enabled	Enabled
RW Desat Task	Disabled	Disabled	Disabled	Enabled
Instrument Power Model	On	Off	Off	Off
Instrument Data Model	On	Off	Off	Off
Transmitter Power Model	Off	On	Off	Off
Transmitter Data Model	Off	On	Off	Off

interface is provided in Figure 3. The Gym environment provides standardized methods that an agent can use to interact with the environment. At each step through the environment, the agent takes an action and receives a reward and a new observation. The agent uses the new observation to take the next action, and the process continues until the finite time-horizon is reached or the episode terminates due to success or failure. When the agent passes an action to the environment, the environment passes the action through to the BSK simulator, which turns the relevant models on or off and executes the dynamics for a specified amount of time.

Table 4 provides the simulation parameters used in the `Gym` environment. The semi-major axis of the orbit is set to a constant 6871 km. However, the other orbital parameters are sampled from distributions. The eccentricity is sampled from a distribution with a relatively large range for low-Earth orbit, $\mathcal{U}[0, 0.01]$. Therefore, the agent experiences a range of orbit altitudes for training purposes. The inclination, longitude of the ascending node, and argument of the periapsis are selected from uniform distributions with a range of 20 deg. By sampling the initial conditions from many uniform distributions, agents are presented with a diverse set of training data. If the agents can learn to operate in any of the orbits generated from these distributions, a case for generalizability within the range of low-Earth orbits can be made. Generalizability makes agents more robust to mismodeling or an erroneous orbit insertion. Note, the orbital parameter ranges do not include

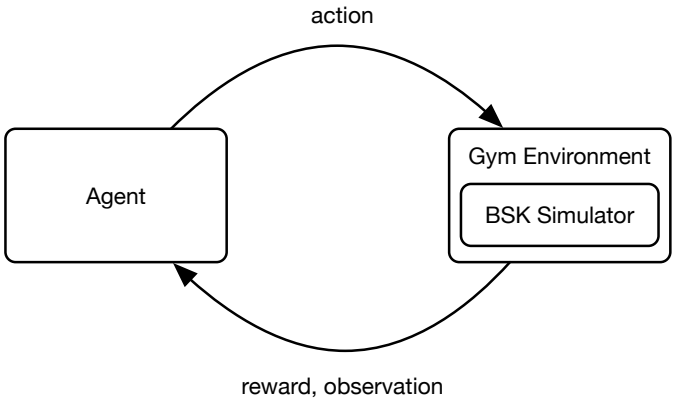


Figure 3: OpenAI Gym Framework

Table 4: Simulation Parameters

Orbit	
Semi-Major Axis, a	6871 km
Eccentricity, e	$\mathcal{U}$[0, 0.01]
Inclination, i	$\mathcal{U}$[40, 60] deg
Long. of Ascend. Node, Ω	$\mathcal{U}$[0, 20] deg
Arg. of Periapsis, ω	$\mathcal{U}$[0, 20] deg
True Anomaly, f	$\mathcal{U}$[0, 360] deg
Spacecraft	
Disturbance Torque, $\boldsymbol{\tau}_{\text{ext}}$	2×10^{-4} Nm
Attitude Initialization, $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$	$\mathcal{U}$[0, 1.0] rad
Rate Initialization, ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$	$\mathcal{U}$[-1e-05, 1e-05] rad/s
Reaction Wheel Speeds	$\mathcal{U}$[-4000, 4000] RPM
Initial Battery Charge	$\mathcal{U}$[30, 50] Whr
Planning Horizon	
Maximum Simulation Time, t_{max}	270 minutes
Mode Length, t_{mode}	6 minutes

any imaginable low-Earth orbit to keep the amount of training data needed to generalize behavior reasonable. Furthermore, a larger inclination range would necessitate ground stations closer to the poles in the event of high inclination orbits. This would increase simulation complexity while making a marginal increase in the case for generalizability. Furthermore, more specific orbit conditions allow for the demonstration of generalization extrapolated to orbits outside of the conditions provided in Table 4. This gives the opportunity to demonstrate robustness outside of training data distributions.

Markov Decision Process

The Basilisk simulation described in the previous section is formulated as a finite-horizon, deterministic Markov Decision Process (MDP). An MDP is a sequential decision-making problem in which an agent selects an action, a_i, in state, s_i, following some policy, $a_i = \pi(s_i)$, which maps states to actions. At the next state, s_{i+1}, the agent receives a reward, r_i, based on a reward function, $R(s_i, a_i, s_{i+1})$. An MDP follows the Markov assumption, which states that the next state is dependent only on the current state and action taken. Mathematically, this is represented with the equality below, where $T(s_{i+1}|s_i, a_i)$ represents the probability of transitioning to state s_{i+1} given state s_i and action a_i.

$$T(s_{i+1}|s_i, a_i) = T(s_{i+1}|s_i, a_i, s_{i-1}, a_{i-1}, ..., s_0, a_0) \tag{1}$$

A Markov Decision Process is represented by the 5-tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$. $\mathcal{S}$ is the state-space for the problem and is shown in Equation (2). Planet-centered, planet-fixed inertial position and velocity unit vectors, expressed in inertial frame $\mathcal{N}$ components as ${}^{\mathcal{N}}\hat{\mathbf{r}}$ and ${}^{\mathcal{N}}\hat{\mathbf{v}}$, are included in the state space of this MDP. The reason for this is two-fold. Specifically, the function approximators can correlate the orbital parameters of the satellite to high-value states when ground station access is approaching. Ground station access, q_j, is defined for each ground station location j as the percentage of the planning interval i that the ground station is visible to the spacecraft. Because access is computed at the end of the interval, the inertial position and velocity provide function approximators with a state that indicates a high-value state tuple. More generally, the position and velocity of the spacecraft may give function approximators additional information on the type of orbit the spacecraft is in so decision-making agents can make more non-intuitive decisions related to resource management or data collection. Theoretically, radius and velocity allow function approximators to generalize its behavior to all orbits within the training parameters. While the agent will not have state information on altitude because radius and velocity are normalized, the small range of altitudes (due to eccentricity and perturbations) render this lack of state information relatively inconsequential. The percentage of the total planning horizon that has already passed, p, is also included in the state space. Because the problem is a finite-horizon problem, value at the start of the simulation (before any downlink windows have passed) is typically higher than the value at the end of a simulation (when the majority of access windows have already passed). By including p, the function approximator can compute more accurate value estimates.

$$\mathcal{S} = \{{}^{\mathcal{N}}\hat{\mathbf{r}}, {}^{\mathcal{N}}\hat{\mathbf{v}}, \boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}, {}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}, \hat{\boldsymbol{\Omega}}, z, k, b, h, q_{1:7}, p\} \tag{2}$$

To keep the satellite within resource constraints, several other states are added to $\mathcal{S}$. The Modified Rodrigures Parameter[22] (MRP) attitude error, $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$, the inertial angular velocity, ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$, and reaction wheel velocities over the maximum allowable velocities, $\hat{\boldsymbol{\Omega}}$, are included to manage the attitude determination and control system. The percent charge of the battery, z, an eclipse indicator, k, and the percent fill of the data buffer, b are also included in the state space so the function approximator can correlate other constraint violations with low-value states so corrective actions may be taken (ie. charging batteries and downlinking data). The percentage of the planning interval spent downlinking data, h, is also included in the state space. This state represents how much of the access time is utilized by the satellite.

$$\mathcal{A} = \{\text{Image}, \text{Downlink}, \text{Charge}, \text{Desaturate}\} \tag{3}$$

The action-space, $\mathcal{A}$, includes the four separate flight modes previously described - image, downlink, charge, and desaturate. The mode-based planning approach lends itself well to Markov Decision Processes. High-level behavior can be abstracted by the use of modes, with the low-level behavior of the modes dictated by the state of the system. Each mode lasts for a total of six minutes, Δt_{mode}. A timespan of six minutes was selected to ensure attitude error is negligible by the end of the planning interval and to give the satellite enough time to dump the momentum in the reaction wheels during desaturation.

The reward function is defined as the amount of data downlinked over each planning interval in megabytes, H_i. Because the problem is finite-horizon, a $\gamma = 1.0$ is used. A $+1$ success bonus is included in the reward if the agent reaches the end of the planning horizon without failing. If the agent does fail at any point during planning, a reward of -1000 is returned and the episode terminates immediately.

$$R(s_i, a_i, s_{i+1}) = \begin{cases} H_i \text{ if !failure} \\ H_i + 1 \text{ if } t \geq t_{\text{max}} \text{ and !failure} \\ -1000 \text{ if failure} \end{cases} \tag{4}$$

In the EOS Markov Decision Process, a failure constitutes a violation of resource constraints. Failures include zero charge in the battery, reaction wheels exceeding their maximum speeds, or an overflow in the data buffer. Failures are evaluated at the end of a planning interval i.

$$\text{failure if } z = 0, \text{ any}(\hat{\boldsymbol{\Omega}} \geq 1), \text{ or } b \geq 1 \tag{5}$$

The transition function, T, is represented by a Basilisk simulation formulated as a `Gym` environment integrated forward six minutes at a time.

METHODS

Monte Carlo Tree Search

Solutions to MDPs involve finding either the optimal policy, $\pi^*(s_i)$, or the optimal value function, $V^*(s_i)$. The optimal policy is the mapping from states to actions that maximizes the value function.

$$\pi^*(s_i) = \text{arg max}_\pi V^\pi(s_i) \tag{6}$$

The value function, $V^\pi(s)$, is defined as the expected sum of all future reward following a policy π.

$$V^\pi(s_i) = R(s_i, \pi(s_i)) + \gamma \sum_{s_{i+1} \in \mathcal{S}} T(s_{i+1}|s_i, \pi(s_i)) V^\pi(s_{i+1}) \tag{7}$$

The state-action value function, $Q^(s_i, a_i)$, is the value of a particular state-action pair. The optimal value function can be found by evaluating $Q^*(s_i, a_i)$ with the action that maximizes $Q^*(s_i, a_i)$.

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

Many algorithms have been posed to solve Markov Decision Processes. Two of the most well-known algorithms, policy iteration[23] and value iteration,[24] compute the optimal policies and value functions offline by repeatedly iterating over the policy or the value function using a form of the Bellman operator. While these techniques provide exact solutions, they require an explicit transition function and discrete state spaces. Conversely, online algorithms solve Markov Decision Processes while interacting with the environment such that only reachable states are considered. In a deterministic environment, this allows for the relatively fast computation of optimal policies in large state spaces. Furthermore, online algorithms may be combined with the generalization power of artificial neural networks. In this work, the authors hypothesize that MCTS can compute optimal state-action value functions and artificial neural networks can be trained to generalize the optimal policies.

Monte Carlo tree search, specifically the Upper Confidence Bound for Trees (UCT), is an online search algorithm that computes optimal solutions to decision-making problems by continually simulating interactions with the environment to compute intermediate state-action values, which are used to incrementally step through the real environment. The specific version of UCT used in this work is described in detail by Kochenderfer.[25] Figure 4 demonstrates this process. The agent executes the demonstrated process from the current state a specified number of times, which is known as the number of "simulations-per-step."

During selection, the agent chooses the action that maximizes the intermediate state-action value, Q, plus an exploration bonus, U. During the expansion step, if the agent reaches a state that it has not visited before, it initializes a state-action value for each possible action in that state, as well as the number of times the state-action combinations have been visited (initialized to zero). After expansion, the agent executes a rollout policy to step through the problem until the episode ends. In this problem, the agent stops executing the rollout if it violates the resource constraints or the end of the planning interval is reached. A rollout policy is a policy that either randomly selects actions or follows some heuristic that will lead the agent to reward. The reward generated during rollout is then backed up through each state, and the state-action value pairs are updated.

This process is repeated until the specified number of simulations-per-step is reached. The agent then decides on which action to select in the real environment by selecting the action associated with the maximum state-action value pair. The process is repeated for the designated number of simulations-per-step at the next state in the real environment. The process continues until the episode ends due to failure or reaching the end of the planning interval.

Rollout Policy

Two different rollout policies are explored - random and heuristic. The random rollout policy selects random actions unless a downlink opportunity is present (ie. any of the states $q_{1:7}$ are non-zero), which initiates

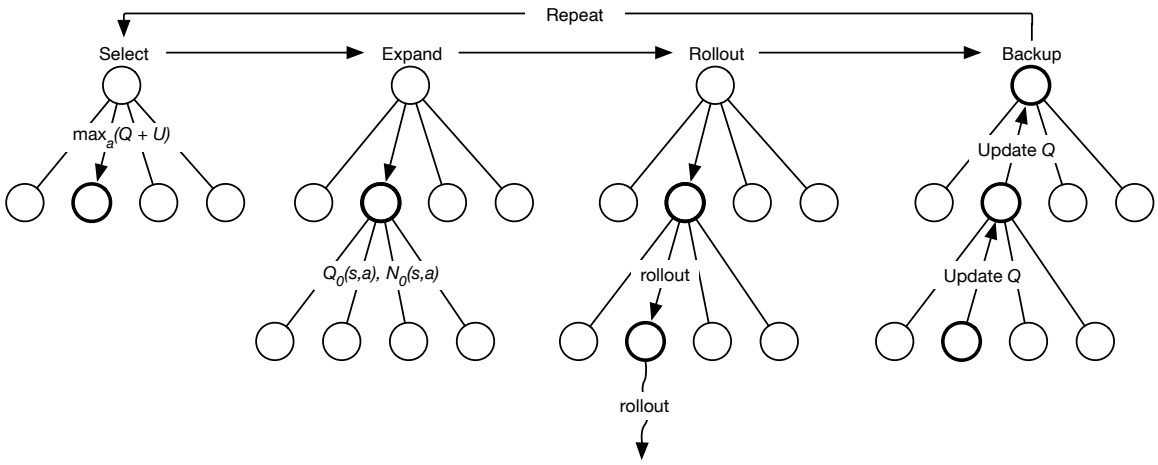


Figure 4: Upper Confidence Bound for Trees

a downlink mode. The heuristic rollout policy, however, examines the states most relevant to constraint violations to determine which action will guarantee a resource constraint violation does not occur. In the event that a nominal resource state is achieved, the heuristic policy either downlinks or images.

The states most relevant to resource constraint violations are the body rate of the spacecraft (${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$), the rate of the reaction wheels ($\boldsymbol{\Omega}$), the power in the battery (Z), and the amount of data in the data buffer (B). The limits on each state variable are provided in Equations 9 - 12. A low-fidelity safety MDP is constructed using these variables. The state of this safety MDP is given by the tuple (Tumbling, Saturated, Low Power, Buffer Limit), where each state in the tuple can take the value true or false. In total, there are 16 possible states. This technique is similar to what Harris et al. employ for shielded deep reinforcement learning in the EOS scheduling problem.[11] However, Harris et al. use the shield to conservatively bound a learning agent's actions based on the safety MDP during training. In this work, the heuristic policy derived from the safety MDP guides MCTS to high-value states that avoid constraint violations during rollout. The heuristic policy does not interfere with the action selected by MCTS when the agent takes a step in the real environment or selects intermediate actions during simulation.

$$|{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}| \geq \text{1e-2 rad/s} \rightarrow \text{Tumbling = True} \tag{9}$$

$$|\boldsymbol{\Omega}| \geq 400 \text{ rad/s} \rightarrow \text{Saturated = True} \tag{10}$$

$$Z \leq 40 \text{ Whr} \rightarrow \text{Low Power = True} \tag{11}$$

$$B \geq 0.8 \text{ GB} \rightarrow \text{Buffer Limit = True} \tag{12}$$

Based on the state of the safety MDP, the heuristic policy selects the action that guarantees a resource violation will not occur. The value of the state tuple and associated action may be found in Table 5. Like the random rollout policy, if the image action is selected (which means the system state is nominal in terms of the safety variables), but any of the ground station access variables $q_{1:7}$ are non-zero, downlink is initiated instead.

State-Action Value Function Neural Networks

Monte Carlo tree search is used to generate search trees that are regressed over using different neural network architectures. In order for the trees generated by MCTS to be used, the intermediate state-action value pairs found using MCTS must be modified. Only the intermediate state-action value pairs associated with states that are visited in the real environment are used for neural network regression. At the end of each planning interval solved using MCTS, the reward received in the real environment is used to compute new

Table 5: Heuristic Policy Conditional States

Tumbling	Saturated	Low Power	Buffer Limit	Action
1	1	1	1	Charge
1	1	1	0	Charge
1	1	0	1	Desaturate
1	1	0	0	Desaturate
1	0	1	1	Charge
1	0	1	0	Charge
1	0	0	1	Downlink
1	0	0	0	Image
0	1	1	1	Desaturate
0	1	1	0	Desaturate
0	1	0	1	Desaturate
0	1	0	0	Desaturate
0	0	1	1	Charge
0	0	1	0	Charge
0	0	0	1	Downlink
0	0	0	0	Image

Q-values in the main tree for the actual actions selected. The intermediate state-action value pairs for each other action are left as they are. This process is demonstrated in Figure 5.

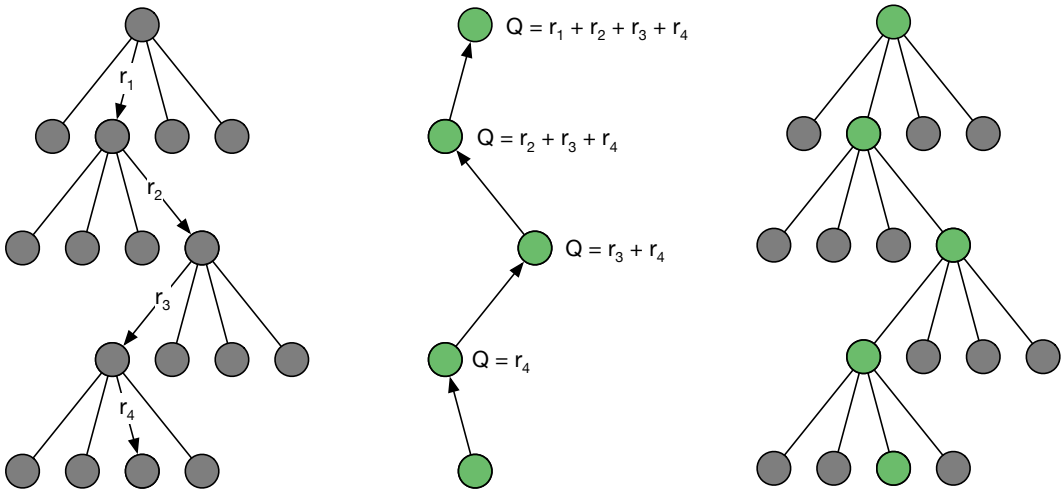


Figure 5: UCT Tree Value Computation

Figure 6 demonstrates the training pipeline that is used to train and validate the neural networks. A large number of episodes starting at random initial conditions are solved using MCTS, and the state-action value pairs are assembled for each set of initial conditions. A total of 1,200 unique initial conditions are solved by MCTS. Each state and its four associated state-action value pairs are separated from the tree and added to the training set. The training set is randomized and split into a training and test set where 90% of the data is used for training and 10% is used for testing. After generation and assembly of the data, neural networks are trained to produce state-action value function approximators, $Q_\theta(s, a)$. The neural networks are validated on the environment by executing them on a test set of 100 initial conditions. At each step through each

environment, the state is input into the value network and the action that returns the highest state-action value pair is selected. The neural network policy is the selection of the action associated associated with the highest state-action value pair at each step through the environment. This is represented in Equation (13).

$$\pi_\theta(s) = \arg\max_a Q_\theta(s, a) \tag{13}$$

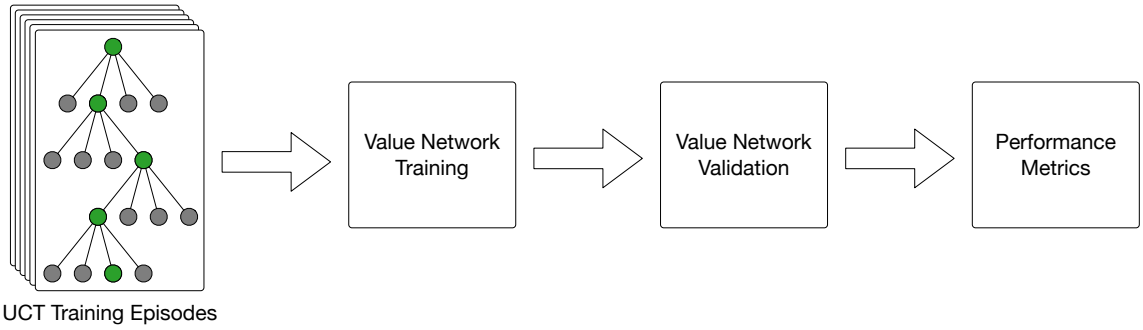


Figure 6: Value Network Training

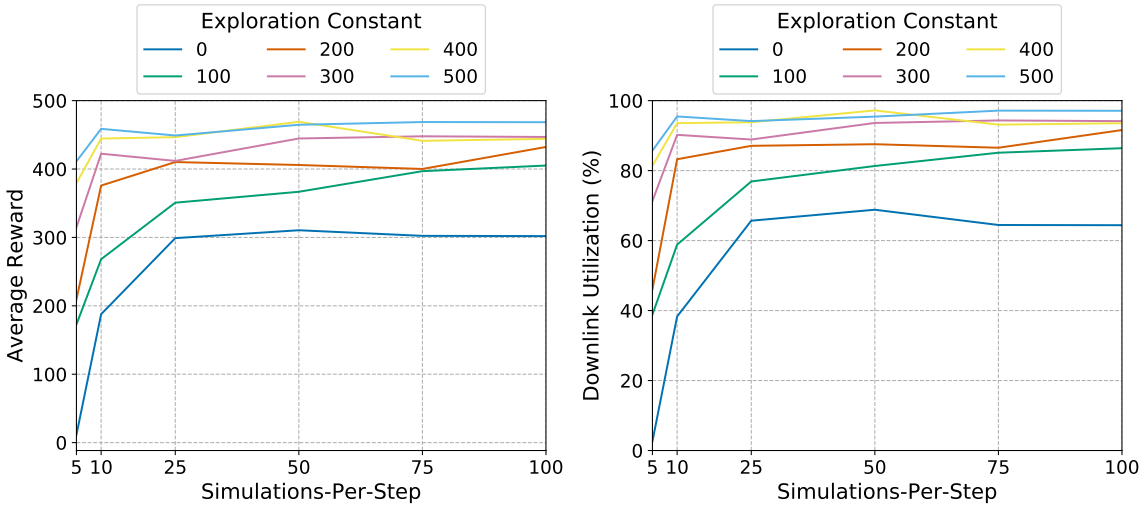
RESULTS

Monte Carlo Tree Search Hyperparameterization

A hyperparameter search is conducted to determine the best Monte Carlo tree search hyperparameterization for generating neural network training data. In the following search, different combinations of two key parameters for MCTS are tested: the exploration constant, c, and the number of simulations-per-step. Furthermore, this search is conducted for both types of rollout policies - random and heuristic. Each hyperparameter combination is evaluated based on average episodic reward and downlink utilization. The downlink utilization is a measure of how effectively the agent utilizes downlink opportunities and is defined as the percent time the agent downlinks data over all available downlink windows.

Figure 7 displays both the average reward and average downlink utilization for MCTS with a heuristic rollout policy. To generate these plots, MCTS is executed on the same set of 10 different initial conditions for each combination of c and number of simulations-per-step. The results show that the average reward and downlink utilization are much more dependent on the exploration constant than the number of simulations-per-step. Adequate exploration ensures that the high-reward states discovered by the rollout policy are found again during the simulation step. Furthermore, adequate exploration allows MCTS to find higher value states than those discovered during rollout. While the exploration constant appears to be the most important hyperparameter, the number of simulations per step is important in terms of optimality. At 10 simulations-per-step, MCTS achieves a maximum average reward of 459 and downlink utilization of 95.5%. At 100 simulations-per-step, MCTS achieves a maximum average reward of 469 and downlink utilization of 97.1%. In the literature, MCTS is shown to converge as the number of simulations-per-step approaches infinity. While MCTS achieves acceptable performance for this problem at 10 simulations-per-step, it takes many more simulations-per-step to converge to the optimal solution.

In Figure 8, the same hyperparameter search is conducted for a random rollout policy. Both average reward and downlink utilization are much lower than results from the heuristic rollout policy. In Figure 9, the resource management success rate is shown. 50 simulates-per-step is the minimum required number for most exploration constants to achieve a 100% success rate. In the case of the heuristic rollout policy, the success rate is 100% regardless of the combination of hyperparameters. As hypothesized, the heuristic rollout policy does a far better job at avoiding resource constraint violations.



(a) Reward

(b) Downlink Utilization

Figure 7: UCT Hyperparameter Search - Heuristic Rollout Policy

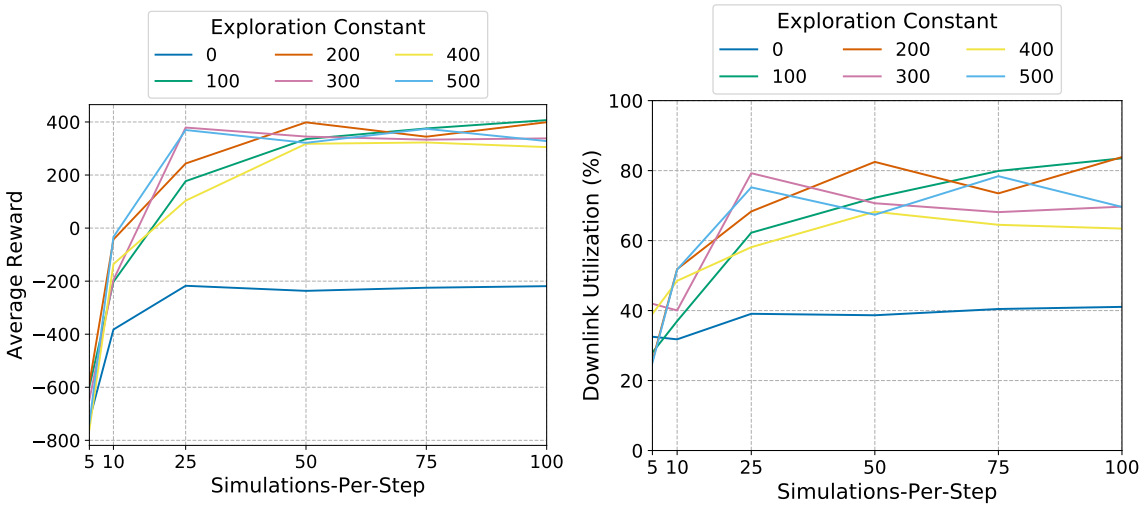
State-Action Value Network Hyperparameterization

As described in the State-Action Value Function Neural Networks section, state-action value data is generated by using MCTS to solve the EOS scheduling problem for 1,200 randomized initial conditions. An exploration constant of $c = 500$ and 10 simulations-per-step are the selected hyperparameters for MCTS. The selected hyperparameters balance the quality of the solutions with the time it takes to generate a reasonable amount of training data. To determine the best neural network architecture, a hyperparameter search is conducted over many different hyperparameter values.

The first hyperparameter search examines the performance of the state-action value networks by varying the number of hidden layers, number of nodes per hidden layer, and activation functions. The dropout rate, which is the probability that a node will be randomly dropped during a training epoch to avoid overfitting,[26] is held constant at 0.25. Furthermore, the α parameter for Leaky ReLU is kept at the default of 0.3. α controls the slope of the Leaky ReLU activation function for $x < 0$. Each network is trained for a total of 3000 epochs, which is the number of times the training data is run through the network. The performance is benchmarked using total reward, downlink utilization, and total time to execute. The percent time each action is taken is also collected to determine the general behavior of each neural network.

In Tables 6 and 7, the performance of each network architecture is provided. For each architecture, the top number is the average reward, and the bottom number is the average downlink utilization. Architectures that achieve more than 95% downlink utilization are highlighted in green. Architectures that achieve between 90-95% downlink utilization are highlighted in yellow. The Leaky ReLU activation function is superior to Tanh, as only network architectures with a Leaky ReLU activation function achieve more than 95% downlink utilization. Furthermore, larger networks perform better on average. Between four and six hidden layers with 250 or 500 nodes each achieves the best performance, totaling between 2.0E5 and 1.3E6 trainable parameters. A smaller number of trainable parameters is preferred to increase the speed of training the networks.

A second, more detailed hyperparameter search is also performed to determine the best combination of parameters when the number of nodes per hidden layer and activation function are held constant. 250 nodes per layer and a Leaky ReLU activation function are selected. The dropout rate, α, and the number of hidden layers are all varied during the hyperparameter search. As demonstrated in Table 8, the general architecture (nodes-per-layer and activation function) is relatively robust to the other hyperparameters. Each dropout rate produces networks that achieve greater than 95% downlink utilization. Furthermore, each number of hidden



(a) Reward

(b) Downlink Utilization

Figure 8: UCT Hyperparameter Search - Random Rollout Policy

Table 6: Leaky ReLU General Architecture Search

	Hidden Layers					
Nodes	1	2	3	4	5	6
100	309 64.7%	356 75.6%	383 79.9%	409 84.4%	365 75.6%	416 87.4%
250	348 72.5%	379 78.3%	433 89.3%	455 94.2%	461 95.9%	453 94.0%
500	344 70.9%	402 82.4%	450 93.0%	460 95.5%	459 94.9%	462 95.8%

layers produces networks that achieve greater than 95% downlink utilization. α is the one parameter that does not produce more than 95% downlink utilization for all values. In most cases, $\alpha = 0.50$ struggles to produce networks that can achieve greater than 90% downlink utilization. Note, the percentage sign is dropped from the downlink utilization for each entry in the table for a compact representation of the results.

In addition to performance, other metrics may give insight into the learned behavior of each neural network architecture. One such metric, the average amount of time each network architecture spends in each mode, can give insight into how well the network has learned which planet-centered, planet-fixed position and velocity vectors are correlated with ground station access. In Figure 10, the average time (expressed as the percent time over each planning interval) each agent spends in each mode is plotted. Only the agents that achieve greater than 95% downlink utilization from Table 8 are plotted. Each agent spends about the same amount of time in the imaging mode. However, the time split between the charging, desaturation, and downlink modes varies widely for different architectures. Several architectures spend between 30-40% of the time in the downlink mode but achieve 95% downlink utilization. This suggests that they have learned where the ground stations are located in terms of the planet-centered, planet-fixed position and velocity vectors. Other architectures spend 60-70% of the time attempting to downlink data. While this does not make a strong case for learning ground station locations, it is more likely than not that they have learned the locations to some degree because of the high downlink utilization. This is encouraging for future work, especially for multi-target scenarios in which the radius and velocity of multiple targets are input states and each target is its own action.

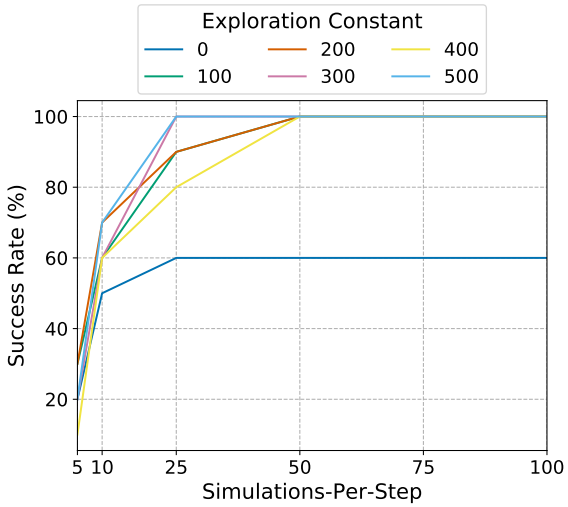


Figure 9: Resource Management Success Rate

Table 7: Tanh General Architecture Search

Nodes	Hidden Layers 1	2	3	4	5	6
100	295 62.5%	1.00 0.00%	230 52.5%	262 59.5%	206 46.9%	204 46.9%
250	293 60.1%	1.00 0.00%	1.00 0.00%	423 88.7%	438 91.1%	382 82.3%
500	315 65.6%	292 60.3%	1.00 0.00%	446 93.0%	422 88.9%	425 88.9%

Another insight into the learned behavior of each network is demonstrated by the small variance in the percent of time each agent spends in the imaging mode. In Figure 10, each agent spends around 10-15% of the time in the imaging mode. This is due to the spacecraft filling up the data buffer and only having a limited number of downlink opportunities available. The high-performing agents were limited by the size of the data buffer. Another learned behavior demonstrated by a few networks highlights the dependence on the activation function. In Table 7, where a Tanh activation function is used, several architectures achieve an average reward of 1.00 and average downlink utilization of 0.00%. This is because the state-action value approximation converged to a local minima where spacecraft charging was always the highest-value action in $Q_\theta(s, a)$.

Table 8: Leaky ReLU Architecture Tuning - 250 Nodes

Dropout		0.05				0.10				0.25			
α		0.01	0.10	0.25	0.50	0.01	0.10	0.25	0.5	0.01	0.10	0.25	0.50
Hidden Layers	4	459 94.9	454 94.5	449 93.0	403 84.3	462 96.0	459 95.0	419 87.0	373 77.0	445 92.7	459 95.1	446 92.6	436 90.9
	5	452 94.3	461 95.4	461 95.5	384 79.4	459 95.3	466 96.7	456 94.6	401 84.4	455 94.1	459 95.4	462 95.8	418 86.6
	6	455 94.7	462 95.9	463 96.1	403 84.4	455 94.5	462 95.8	453 94.4	362 76.3	451 93.7	463 95.8	460 95.3	453 94.4

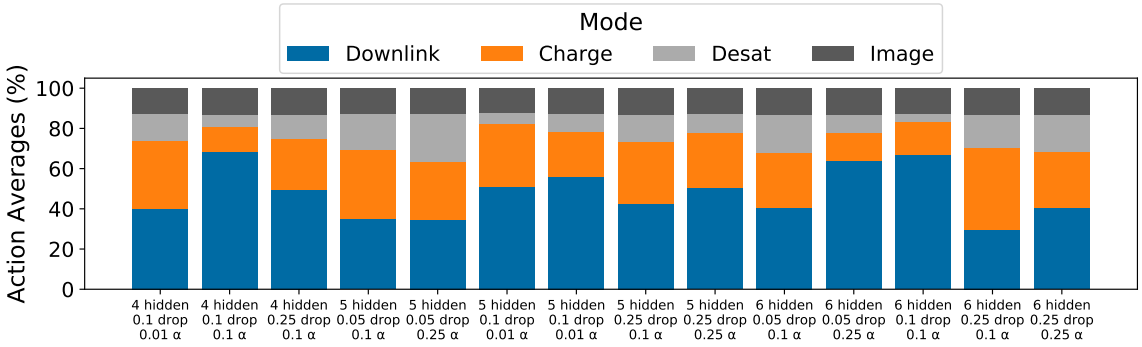


Figure 10: Average Action Percentages - Specific Search

Table 9: Genetic Algorithm Performance

Population Size	Generations		
	45	100	200
10	445 94.0%	463 98.3%	447 96.2%
20	467 98.9%	472 99.8%	472 99.8%

Genetic Algorithm Comparison

To determine the optimality gap of Monte Carlo tree search and the resulting value networks for the given spacecraft configuration, a genetic algorithm is also tested on the same set of initial conditions. The genetic algorithm yields open-loop tasking solutions based on the expected environment, while MCTS and the neural networks yield closed-loop tasking solutions specific to observations generated by stepping through a real environment. Regardless, the genetic algorithm solution provides insight into how optimal the particular MCTS and neural network solutions are. In the interest of time, only the first 10 out of 100 initial conditions are used. The DEAP evolutionary computation framework* is used to implement a simple genetic algorithm to solve for a mode schedule for each initial condition evaluated using the same reward and environment specification described in the Markov Decision Process section. The crossover and mutation probabilities are each set to 0.25. The number of generations and population size are varied between 45-200 and 10-20, respectively. In Table 9, the genetic algorithm seemingly achieves the optimal solution of 472 reward and 99.8% downlink utilization.

Table 10 displays the reward, downlink utilization, and time-to-compute for each algorithm implemented in this paper. The hyperparameter combination that achieves the highest reward is selected for each method. The genetic algorithm produces the best performance on average. Furthermore, the genetic algorithm computes the optimal solution much faster than both random and heuristic MCTS theoretically could. Note, however, that the genetic algorithm is implemented with multi-processing for a single initial condition's solution, whereas the MCTS multi-processing implementation solves the planning problem for multiple initial conditions in parallel. Regardless of execution-time implementation dependencies, the genetic algorithm does not generalize to initial conditions that it has not solved for. Furthermore, the genetic algorithm only generates a single trajectory as opposed to a search tree. The search tree generated by MCTS can be used to train a neural network, $Q_\theta(s, a)$, where the value of each action is an output of the network. The genetic algorithm could be used to train a neural network where the action is an input via one-hot encoding. However, the search state-action value pairs generated by MCTS, which are computed by combining the rollout and simulation reward, produce state-action value estimates that indicate resource constraint violations or missed

*https://deap.readthedocs.io/en/master/

Table 10: Comparison of Algorithms

Random MCTS	Heuristic MCTS	Value Network	Genetic Algorithm
407	469	466	472
83.9%	97.1%	96.7%	99.8%
19,100 s	11,400 s	0.0672 s	1,910 s

downlink opportunities in future states. The value of the optimal action is elevated over the noise-floor of the sub-optimal actions during data generation, which would not happen if a genetic algorithm were used for training.

After training, the neural networks achieve near-optimal performance several orders of magnitude faster than any other algorithm implemented in this work. The value network is the best candidate for on-board execution where execution speed is paramount on resource-constrained flight processors. Furthermore, the value network should respond well to noisy radius and velocity measurements output from orbit determination algorithms or the attitude states and rates output from Kalman filters. This would likely result in slightly sub-optimal performance. However, the authors hypothesize that generalizability of the value network and the continual evaluation of it every six minutes would result in a negligible degradation in performance due to noisy measurements.

Finally, the state-action value networks achieve near-optimal performance by interacting with a given environment only one time, selecting an action after each observation. MCTS and the genetic algorithm require many environment interactions to solve for the optimal solution. Due to the power of neural networks to generalize across training data, the state-action value networks are able to interpolate and compute solutions to planning horizons with initial conditions they have never experienced before. Future work should study how the neural networks perform on initial conditions outside of the ranges of the distributions of the training data to test their ability to extrapolate to orbital parameters outside of the training data ranges.

CONCLUSION

This work successfully demonstrates the use of Monte Carlo tree search (MCTS) and state-action value regression with neural networks for the given Earth-observing satellite (EOS) scheduling problem. The performance of Monte Carlo tree search is investigated by varying rollout policies, exploration constants, and the number of simulations-per-step. It is shown that MCTS achieves near-optimal performance with a heuristic rollout policy and relatively small number of simulations-per-step, Furthermore, the state-action value trees generated by MCTS are regressed over using a variety of neural network architectures. Networks with 2.0E5 to 1.3E6 trainable parameters perform the best, with the Leaky ReLU activation function proving to be very robust to the dropout rate, number of hidden layers, and α. Finally, MCTS and state-action value network regression are compared to a genetic algorithm, which provides an upper bound on performance. The state-action value networks achieve comparable performance to both the genetic algorithm and MCTS, but with a fraction of the execution time after training, making a state-action value network a candidate for on-board execution.

ACKNOWLEDGEMENT

This work is supported by a NASA Space Technology Graduate Research Opportunity. The author would like to thank the AVS Laboratory for developing the Basilisk Astrodynamics Software Framework. The author would also like to thank Andrew Harris, who developed a Basilisk Gym Environment the author could modify for this version of the EOS scheduling problem.

REFERENCES

[1] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, "Towards an Application Framework for Automated Planning and Scheduling," *1997 IEEE Aerospace Conference*, April 1997, pp. 375–386, 10.1109/AERO.1997.574426.

[2] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood, "CASPER: Space Exploration Through Continuous Planning," *IEEE Intelligent Systems*, Vol. 16, September 2001, pp. 70–75, 10.1109/MIS.2001.956084.

[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, 2005, pp. 196–216, 10.2514/1.12923.

[4] S. Chien, J. Doubleday, D. R. Thompson, K. Wagstaff, J. Bellardo, C. Francis, E. Baumgarten, A. Williams, E. Yee, E. Stanton, and J. Piug-Suari, "Onboard Autonomy on the Intelligent Payload EXperiment (IPEX) CubeSat Mission," *Journal of Aerospace Information Systems (JAIS)*, April 2016, 10.2514/1.I010386.

[5] 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, 10.1016/j.cor.2014.11.004.

[6] D.-H. Cho, J.-H. Kim, H.-L. Choi, and J. Ahn, "Optimization-Based Scheduling Method for Agile Earth-Observing Satellite Constellation," *Journal of Aerospace Information Systems*, Vol. 15, No. 11, 2018, pp. 611–626, 10.2514/1.I010620.

[7] B. Gaudet, R. Linares, and R. Furfaro, "Adaptive Guidance and Integrated Navigation with Reinforcement Meta-learning," *Acta Astronautica*, Vol. 169, April 2020, pp. 180–190, 10.1016/j.actaastro.2020.01.007.

[8] B. Hockman and M. Pavone, "Stochastic Motion Planning for Hopping Rovers on Small Solar System Bodies," *Robotics Research*, Springer, 2020, pp. 877–893.

[9] D. M. Chan and A. Agha-mohammadi, "Autonomous Imaging and Mapping of Small Bodies Using Deep Reinforcement Learning," *2019 IEEE Aerospace Conference*, 2019, pp. 1–12, 10.1109/AERO.2019.8742147.

[10] A. Harris, T. Teil, and H. Schaub, "Spacecraft Decision-Making Autonomy Using Deep Reinforcement Learning," *AAS Spaceflight Mechanics Meeting*, Maui, Hawaii, January 13–17 2019. Paper No. AAS-19-447.

[11] A. Harris and H. Schaub, "Deep On-Board Scheduling For Autonomous Attitude Guidance Operations," *AAS Guidance, Navigation and Control Conference*, Breckenridge, CO, January 30 – Feb. 5 2020. AAS 02-117.

[12] A. Harris and H. Schaub, "Spacecraft Command and Control with Safety Guarantees using Shielded Deep Reinforcement Learning," *AIAA SciTech*, Orlando, Florida, January 6–10 2020.

[13] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," *arXiv preprint arXiv:1710.02298*, 2017.

[14] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo Search," *Univ. Tartu, Estonia, Tech. Rep*, 2006.

[15] D. Shah, Q. Xie, and Z. Xu, "Non-Asymptotic Analysis of Monte Carlo Tree Search," *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '20, New York, NY, USA, Association for Computing Machinery, 2020, pp. 31–32, 10.1145/3393691.3394202.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis, "Mastering the Game of Go Without Human Knowledge," *Nature*, Vol. 550, 10 2017, pp. 354–359, 10.1038/nature24270.

[17] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model," *arXiv preprint arXiv:1911.08265*, 2019.

[18] S. Fedeler and M. Holzinger, "Monte Carlo Tree Search Methods for Telescope Tasking," *AIAA Scitech 2020 Forum*, 2020, p. 0659.

[19] A. Herrmann and H. Schaub, "Monte Carlo Tree Search with Value Networks for Autonomous Spacecraft Operations," *AAS/AIAA Astrodynamics Specialist Conference*, Lake Tahoe, CA, Aug. 9-13 2020. AAS 20-473.

[20] P. W. Kenneally *et al.*, "Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework," *7th International Conference on Astrodynamics Tools and Techniques (ICATT)*, DLR Oberpfaffenhofen, Germany, Nov. 6–9 2018.

[21] G. S. Center, "Near Earth Network Users' Guide," tech. rep., National Aeronautics and Space Administration, Greenbelt, MD, March 2019.

[22] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*. Reston, VA: AIAA Education Series, 4th ed., 2018, 10.2514/4.105210.

[23] R. A. Howard, "Dynamic Programming and Markov Processes," 1960.

[24] R. Bellman, "A Markovian Decision Process," *Indiana Univ. Math. J.*, Vol. 6, 1957, pp. 679–684.

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

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, Vol. 15, No. 56, 2014, pp. 1929–1958.