Journal of Aerospace Information Systems
Vol. 19, No. 1, January 2022

Monte Carlo Tree Search Methods for the Earth-Observing Satellite Scheduling Problem

Adam P. Herrmann* and Hanspeter Schaub†
University of Colorado, Boulder, Colorado 80303

https://doi.org/10.2514/1.I010992

This work explores on-board planning for the single spacecraft, multiple ground station Earth-observing satellite scheduling problem through artificial neural network function approximation of state–action value estimates generated by Monte Carlo tree search (MCTS). An extensive hyperparameter search is conducted for MCTS on the basis of performance, safety, and downlink opportunity utilization to determine the best hyperparameter combination for data generation. A hyperparameter search is also conducted on neural network architectures. The learned behavior of each network is explored, and each network architecture's robustness to orbits and epochs outside of the training distributions is investigated. Furthermore, each algorithm is compared with a genetic algorithm, which serves to provide a baseline for optimality. MCTS is shown to compute near-optimal solutions in comparison to the genetic algorithm. The state–action value networks are shown to match or exceed the performance of MCTS in six orders of magnitude less execution time, showing promise for execution on board spacecraft.

Nomenclature

$\mathcal{A}$	=	action space
a	=	semimajor axis, km
$\mathcal{B}$	=	spacecraft body-fixed coordinate frame
b	=	percent fill of the data buffer
e	=	eccentricity
f	=	true anomaly, deg
$G(\cdot)$	=	generative transition function
H_i	=	data downlinked over planning interval, MB
h	=	percent of planning interval spent downlinking data
i	=	inclination, deg
j	=	ground station number
k	=	eclipse indicator
$\mathcal{N}$	=	planet-centered inertial coordinate frame
$\mathcal{P}$	=	planet-centered, planet-fixed coordinate frame
$Q(\cdot)$	=	state–action value function
q_j	=	access indicator for ground station j
p	=	percent of the planning horizon passed
$\mathcal{R}$	=	reference frame of a given operations mode
$R(\cdot)$	=	reward function
${}^{\mathcal{P}}\hat{\boldsymbol{r}}$	=	spacecraft position unit vector expressed in the planet-centered, planet-fixed coordinate frame
$\mathcal{S}$	=	state space
$T(\cdot)$	=	explicit transition function
$t_{\max}$	=	maximum planning horizon, minutes
t_{mode}	=	planning interval length, minutes
$V(\cdot)$	=	value function
${}^{\mathcal{P}}\hat{\boldsymbol{v}}$	=	spacecraft velocity unit vector expressed in the planet-centered, planet-fixed coordinate frame
z	=	percent charge of the battery
γ	=	discount factor
$\pi(\cdot)$	=	policy
$\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$	=	modified Rodrigues parameter attitude error, rad
τ_{ext}	=	external disturbance torque, N · m
$\hat{\Omega}$	=	normalized reaction wheel speeds
${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$	=	angular velocity of the spacecraft expressed in the body-fixed coordinate frame, rad/s
Ω	=	longitude of the ascending node, deg
ω	=	argument of periapsis, deg

Subscript

θ	=	neural network approximation

I. Introduction

Autonomous spacecraft operations is an enabling capability for future spacecraft missions [1]. In deep space, challenging environments and mission architectures will require autonomous exploration as the round-trip light-time communication delay will constrain maneuvers and prevent opportunistic science collection. Uncertain gravity fields about small bodies limit the amount of planning that may be done before arrival to the body, and even proximity operations about the target body are constrained by the uncertainty in the gravity field or navigation solution. Low-cost spacecraft such as CubeSats may necessitate on-board replanning due to uncertain spacecraft performance or mismodeling. Furthermore, automated planning solutions will become a requirement for certain missions due to the complexity of the planning problem for large constellations [2]. Several challenges must be addressed to enable automated, 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. Although 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 [3]. On-board solutions for operations should also use the full capability of the spacecraft. Planning solutions should be optimal or near-optimal in terms of maximizing science return while meeting science and resource constraint requirements. Finally, on-board task schedulers should have the capability to dynamically respond to last-minute target requests, ground station outages, or mismodeled dynamics without rerunning computationally intensive scheduling algorithms. To address these challenges, this work proposes a method for on-board, closed-loop planning in the context of the single spacecraft, multiple 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 tasking

Received 23 March 2021; revision received 27 July 2021; accepted for publication 31 July 2021; published online 31 August 2021. Copyright © 2021 by Adam Herrmann. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. All requests for copying and permission to reprint should be submitted to CCC at www.copyright.com; employ the eISSN 2327-3097 to initiate your request. See also AIAA Rights and Permissions www.aiaa.org/randp.

*Graduate Research Assistant, Ann & H. J. Smead Department of Aerospace Engineering Sciences, 3775 Discover Drive, AERO 446; adam.herrmann@colorado.edu. Member AIAA.

†Glenn L. Murphy Chair of Engineering, Ann & H. J. Smead Department of Aerospace Engineering Sciences, 3775 Discover Drive, AERO 446; hanspeter.schaub@colorado.edu. Associate Fellow AIAA.

plans are generated on the ground, sequenced, and uplinked to the spacecraft for execution. A notable example for autonomous planning is the Automated Planning/Scheduling Environment (ASPEN) software architecture [4]. ASPEN is an autonomous, ground-based planning and scheduling architecture used for a variety of spacecraft missions. Significant work has also been performed to develop on-board systems that give spacecraft the ability to iteratively repair a ground-based plan if a resource constraint violation or unexpected science opportunity occurs. Continuous Activity Scheduling, Planning, Execution, and Replanning (CASPER) addresses this need by continually checking an existing plan for resource constraint violations and modifying the plan on-board the spacecraft if necessary [5]. CASPER and ASPEN have been applied to several missions, such as Earth Observing 1 (EO-1) [6,7] and Intelligent Payload Experiment (IPEX) [8], to demonstrate on-board schedule modification of ground-based plans. These systems operate in combination with science detection algorithms and sensor webs to mark future science targets [9,10]. Although state-of-the-art solutions to spacecraft planning have decreased the burden on operators and enabled the collection of opportunistic science data, algorithms with more control over operational decisions are required for full autonomy.

In literature, multiple targets variants of the EOS scheduling problem are often formulated as optimization problems [11]. These approaches find optimal solutions for large sets of ground targets, but are brittle to uncertainty, ground station outages, and last-minute target requests. They do not generalize to initial conditions that have not been solved for. Spangelo et al. formulate the EOS 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 manage spacecraft resources [12]. 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 targets by scheduling observation and downlink tasks [13]. Nag et al. account for uncertainty in attitude estimation and solve the multiple targets EOS scheduling problem for an agile spacecraft using dynamic programming to maximize the weighted sum of captured images, demonstrating large performance improvements over nadir-pointing instruments and a huge speed-up in computation due to the use of dynamic programming [14]. In a real operational scenario, the solutions generated by the aforementioned algorithms are executed open-loop on board spacecraft. Replanning must occur in the event of a resource constraint violation, change in the target requests, or a ground station outage. Future planning solutions should operate in a closed-loop implementation to minimize the impacts of mismodeling and respond to new targets requests without having to rerun computationally intensive optimization algorithms. It should also be noted that the majority of the literature for the EOS scheduling problem considers large sets of ground targets for imaging. This work focuses on collecting nadir-pointing, target-agnostic science data while managing resource constraints in closed-loop manner. No specific imaging targets are considered. However, the nadir-pointing science mode can be fed with a new target request each planning interval.

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 of execution after training and ability to generalize across training data. In the small-body domain, RL is well-suited to solve many challenging problems that require fault handling or rigorous treatment 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 [15]. 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 [16]. Chan and Agha-mohammadi solve the small-body mapping problem by formulating it as a partially-observable Markov decision process (MDP) and applying the REINFORCE algorithm to maximize map quality by learning when to execute thrust and imaging commands [17]. 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 [18–20]. 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. [21]. Although policy gradient methods offer several benefits, such as stochastic policies and good performance in continuous state and action spaces [22], only convergence to local maxima can be guaranteed because they fundamentally rely on gradient ascent, 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 policies that can be generalized using state–action value function regression via 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 converges to the globally optimal action as the number of simulations-per-step approaches infinity [23]. 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 [24]. Silver et al. demonstrate that MCTS combined with a state–action value and policy network can achieve superhuman performance in the game of Go [25]. MuZero, a model-based variant of AlphaZero, also demonstrates the power of MCTS methods on a myriad of canonical RL problems [26]. Fedeler and Holzinger extend MCTS to the space situational awareness domain, demonstrating telescope tasking for tracking space objects using a novel MCTS algorithm [27]. Eddy and Kochenderfer formulate a multiple targets EOS tasking problem as a semi-MDP and apply MCTS to compute planning solutions, which are compared with rule-based and mixed-integer programming solutions [28]. Although these methods demonstrate the ability to generate solutions to spacecraft planning problems, none have considered the use of MCTS as a form of data generation for state–action value function neural networks. AlphaZero takes a similar approach for the game of Go, but uses the state–action value and policy network to replace MCTS rollouts and augment the exploration bonus because of the large action space in the game of Go [25]. In this work, the relatively small action space does not necessitate the use of a value and policy network to be executed within MCTS. Therefore, state–action value estimates are computed by MCTS and regressed over using neural networks. The state–action value function neural networks provide a compact representation of planning solutions that may be generalized over a range of low Earth orbits for rapid execution on board spacecraft.

In this paper, the formulation and implementation of the EOS MDP is first described in detail. MCTS, training data generation, and state–action value function neural network regression are then discussed. The results from hyperparameter searches of both MCTS and neural network architectures are presented. The MCTS hyperparameter search investigates different combinations of the exploration bonus, number of simulations-per-step, and rollout policy. A low-fidelity safety MDP is constructed to form a safety-aware rollout policy that can avoid resource constraint failures and find downlink opportunities. The neural network architecture search investigates how the size of the networks, activation functions, and other parameters specific to the activation functions affect performance. The learned behavior of each neural network architecture is investigated, and the sensitivities of each trained neural network to orbital parameters and epochs outside of training distributions are explored to determine the effects of an erroneous orbit insertion and performance months after training. A genetic algorithm is implemented to provide a baseline for determining the optimality of each algorithm, and a comparison between each MCTS method is made on the basis of optimality, execution time, and generalizability to determine which algorithms are good candidates for execution on-board spacecraft.

II. Markov Decision Process

In this formulation of the EOS scheduling problem, a satellite in a 500-km-altitude orbit makes operational decisions to collect and

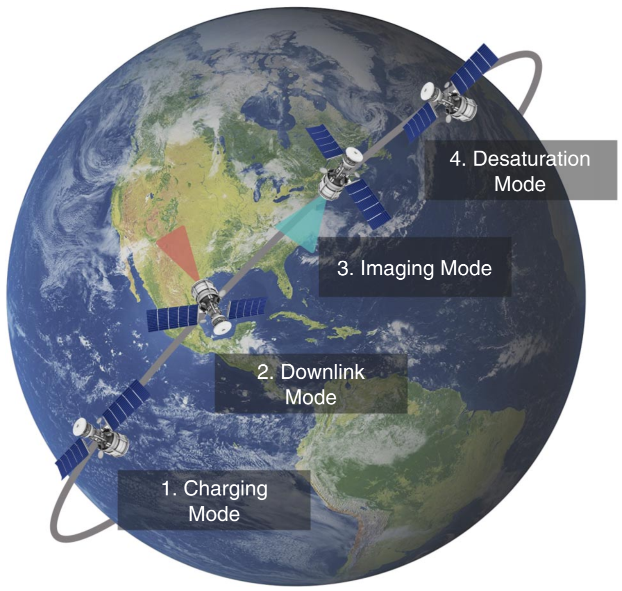


Fig. 1 Scheduling of flight modes.

downlink science data to any of seven different ground stations around the Earth as shown in Fig. 1. A decision-making agent steps through the planning horizon, which is defined as the total length of the simulation, $t_{\max}$. The planning horizon is broken into several planning intervals of length t_{mode}. The planning intervals are defined as the discrete decision-making intervals at which the agent takes actions. The objective of EOS scheduling problem is to maximize the total amount of science data downlinked over the length of the planning horizon while safely managing the resources on-board the spacecraft (i.e., the battery level, data buffer level, and reaction wheel speeds). Science data are collected by pointing an imager in the nadir direction toward Earth. Specific imaging targets are an interesting extension of this work but are beyond the scope of this paper.

The EOS scheduling problem is formulated as a finite-horizon, deterministic 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, $\pi : \mathcal{S} \to \mathcal{A}$, that maps states to actions. At the next state s_{i+1}, the agent receives a reward r_i based on a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. An MDP follows the Markov assumption, which states that the next state is conditionally dependent only on the current state and action. This is represented by Eq. (1), where $T(s_{i+1}|s_i, a_i)$ represents the probability of transitioning to state s_{i+1} conditioned on state s_i and action a_i.

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

An MDP is defined by the five-tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$. The state space, action space, and reward function are described in detail in the following subsections. The transition function $T(s_{i+1}|s_i, a_i)$ is represented by a generative model $G(s_i, a_i)$. Because of the complexity of the underlying problem dynamics, an explicit transition function using conditional probabilities cannot be constructed for this problem. A spacecraft dynamics simulator is used to generate the next state and reward based on the dynamics of the underlying problem, as shown in Eq. (2).

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

A. State Space

The state space for the problem, $\mathcal{S}$, is given by

$$\mathcal{S} : \left\{ {}^{\mathcal{P}}\hat{\boldsymbol{r}}, {}^{\mathcal{P}}\hat{\boldsymbol{v}}, \boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}, {}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}, \hat{\boldsymbol{\Omega}}, z, k, b, h, q_{1:7}, p \right\} \qquad (3)$$

To help with convergence during function approximation, all states are normalized between the range of [0, 1] or [−1, 1]. Planet-centered, planet-fixed position and velocity vectors normalized by their magnitude, expressed in planet-centered, planet-fixed frame $\mathcal{P}$ components as ${}^{\mathcal{P}}\hat{\boldsymbol{r}}$ and ${}^{\mathcal{P}}\hat{\boldsymbol{v}}$, are included in the state space of this MDP. The reason for this is twofold. First, the state–action value 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 was visible to the spacecraft. Because access is computed at the end of the interval, the position and velocity vectors provide function approximators with a state that indicates a high-value state if ground station access is approaching. Second, 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 nonintuitive decisions related to resource management or data collection. Theoretically, radius and velocity allow function approximators to generalize behavior to all orbits within the training distributions. Although 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 alone) renders this lack of state information relatively inconsequential in terms of the stationarity of the transition function.

To keep the satellite within resource constraints, several other states are included in $\mathcal{S}$. The modified Rodrigues parameter (MRP) [29] attitude error $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$, the 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 included in the state space, so the function approximator can correlate other constraint violations with low-value states. The percentage of the planning interval spent downlinking data, h, is also included in the state space to represent how much of the access time is used by the satellite. The percentage of the total planning horizon that has already passed, p, is included in the state space because the problem is finite horizon. Two identical states that only differ in p will not have equal value because the state closer to the end of the planning horizon will not be able to use downlink opportunities past the end of the planning horizon. By including p, the function approximator can compute more accurate value estimates.

B. Action Space

The action space $\mathcal{A}$ includes four separate flight modes: image, downlink, charge, and desaturate.

$$\mathcal{A} : \{\text{Image}, \text{Downlink}, \text{Charge}, \text{Desaturate}\} \qquad (4)$$

The action space is constructed following a mode-based planning approach, abstracting high-level behaviors whose low-level behavior is dictated by the interaction between different subsystems in the simulation. The mode-based planning approach breaks the complex, continuous behavior into a set of discrete actions to make the planning problem tractable [18]. When a mode is selected, the dynamics are propagated using RK4 integration at a 1 s time step, $\Delta t = 1$ s. A time step of 1 s is required to accurately propagate the nonlinear attitude dynamics. For each mode, the dynamics propagation occurs for a 6-min planning interval, t_{mode}. Six minutes is selected to ensure that attitude tracking errors have converged and are negligible by the end of the planning interval and to give the satellite enough time to remove the momentum in the reaction wheels during desaturation. If the planning window is too short, the flight modes may become unstable. If the windows are too large, the safety constraints may be missed.

C. Reward Function

The reward function is formulated to reflect an EOS scheduling problem where the objective is to maximize the science data returned while managing resource constraints. The reward function is defined as the amount of data downlinked over each planning interval in megabytes, H_i, if no resource management failure occurs. A $+1$ success bonus is included in the reward if the agent reaches the end of the planning horizon without failing. In the EOS MDP, a failure constitutes a violation of resource constraints. The failure modes considered are 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.

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

If the agent does fail at any point during planning, a reward of -1000 is returned and the episode terminates immediately. The reward penalty for failure was sized such that it is larger than the maximum amount of positive reward the agent can receive due to downlinking over the entire planning horizon. Because the problem is finite horizon, a $\gamma = 1.0$ is used.

$$R(s_i, a_i, s_{i+1}) = \begin{cases} H_i & \text{if !failure} \\ H_i + 1 & \text{if } t \geq t_{\max} \text{ and !failure} \\ -1000 & \text{if failure} \end{cases} \tag{6}$$

III. Earth-Observing Satellite Simulation

A. Simulation Architecture

The EOS scheduling problem is simulated using the Basilisk Astrodynamics Software Framework [30] (http://hanspeterschaub.info/basilisk). The modularity and speed of Basilisk allows for a high-fidelity simulation with cross-couplings between spacecraft subsystems to be constructed and quickly executed. The use of a complex simulation provides a realistic dynamics environment that the algorithms discussed in this paper can learn from and be validated on. Furthermore, the framework provides the opportunity to simulate real flight software used on board spacecraft, which lends itself to future autonomy work that may fly on board a real spacecraft.

A complete diagram of the associated Basilisk modules may be found in Fig. 2. Each module in the diagram represents a distinct, modularized block of code that receives inputs from other modules, performs computations, and sends outputs to modules subscribed to its messages. The Basilisk simulation includes a full attitude control system to simulate a representative spacecraft mission where systems are coupled to the physical attitude dynamics. The Hill and inertial reference frames are switched between based on the specific flight mode and passed to an attitude error computation module as the reference attitude state message. Next, the attitude tracking error message is evaluated and passed on to an MRP feedback control law. The MRP feedback control law sends motor torque commands to reaction wheels to change the dynamics of the spacecraft. The reaction wheels are modeled after the Honeywell HR16 reaction wheels. A momentum dumping module is implemented as well. It maps reaction wheel momentum to thruster on-time commands to remove momentum from the reaction wheels. The thrusters are modeled after the Moog Monarc-1 thrusters. Attitude perturbations are incorporated through the use of a faceted drag model and random external disturbance torques to build up momentum in the reaction wheels. Orbital perturbations like multibody gravity effects (including Earth, sun, and the moon) and Earth J_2 perturbations are also implemented. A summary of the dynamics models and the relevant states they impact are provided in Table 1.

The parameters of the spacecraft subsystems are found in Table 2. 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 relative to the sun location, panel area, and efficiency. Earth eclipse effects are also considered when determining solar panel performance. 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 omnidirectional, 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 3. The ground stations are selected from a list of stations used by NASA's Near Earth Network [31]. A Boulder, Colorado, ground station is also included. The ground stations are selected from the list such that they are located within the minimum and maximum boundaries of the randomly generated orbits.

B. Spacecraft Modes

A mode-based planning approach is taken to transform complex spacecraft behavior into discrete actions. Table 4 provides the details of the four flight modes of the spacecraft. 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 downlink data only if a ground station is accessible. A ground station is accessible if the spacecraft is within the minimum elevation listed in Table 3. This is continually checked during the propagation of the dynamics. If the ground station is not accessible, the transmitter 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 toward the sun while the thrusters are used to remove momentum from the reaction wheels. Like the charging mode, all power and data models are turned off.

C. Gym Environment

To complete the construction of the generative transition model, the Basilisk simulation is wrapped in a Gym environment, building upon previous open-source work (http://github.com/atharris/basilisk_env) from the AVS Laboratory. Gym environments provide a standardized interface and set of test environments that decision-making algorithms can act on and learn from (https://gym.openai.com/). A simple depiction of this interface is provided in Fig. 3. 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 Basilisk simulator, which turns the relevant models on or off and executes the dynamics for the specified amount of time, t_{mode}.

IV. Monte Carlo Tree Search Methods

A. Solving Markov Decision Processes

Solutions to MDPs involve finding 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.

$$\pi^*(s_i) = \underset{\pi}{\operatorname{argmax}} V^{\pi}(s_i) \tag{7}$$

The optimal value function $V^*(s)$ is defined iteratively using the Bellman equation:

$$V^*(s_i) = \max_a \left(R(s_i, a_i) + \gamma \sum_{s_{i+1} \in \mathcal{S}} T(s_{i+1} | s_i, a_i) V^*(s_{i+1}) \right) \tag{8}$$

The optimal value function is the value of the optimal policy. Furthermore, the state–action value function $Q(s_i, a_i)$ is the value

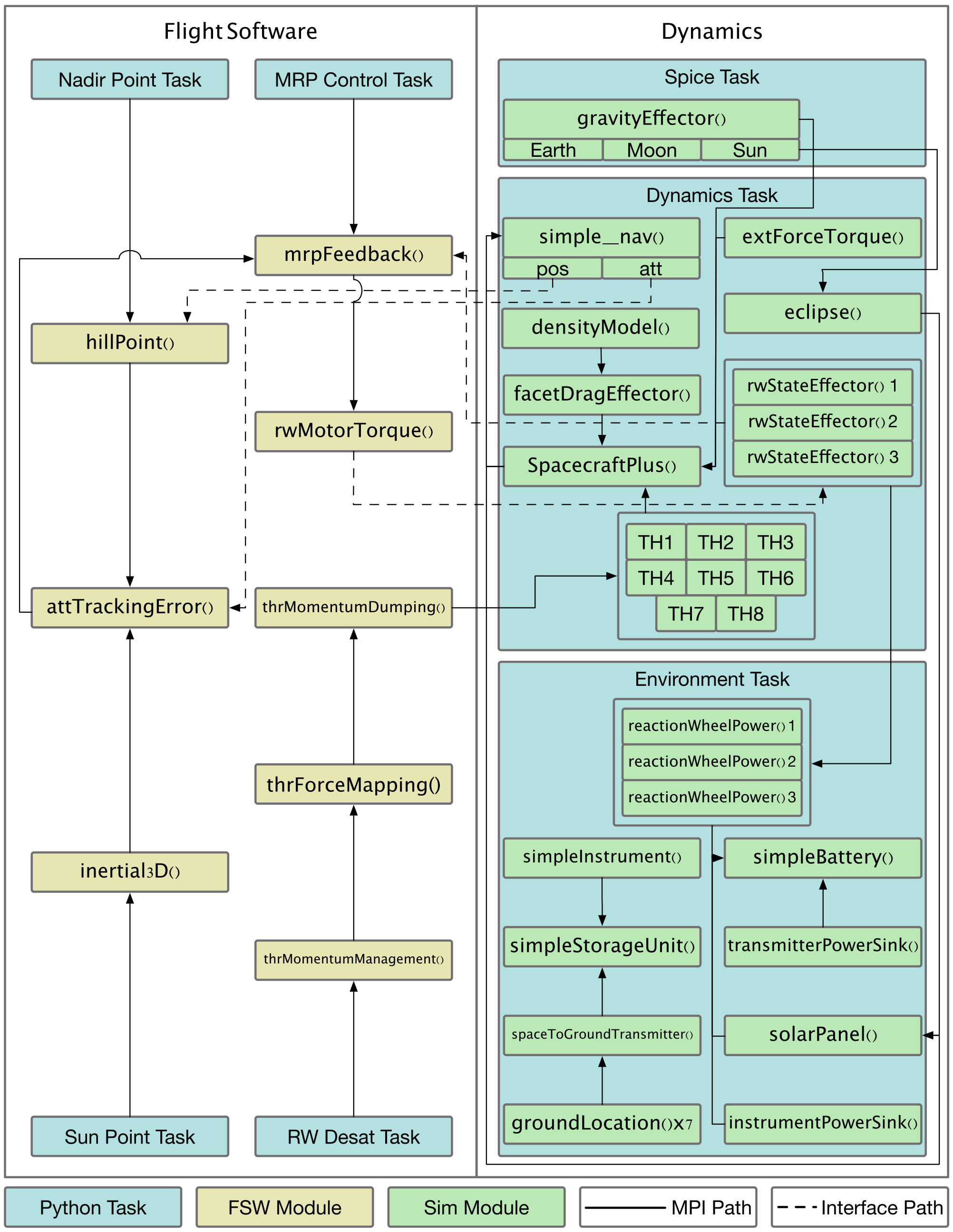


Fig. 2 Basilisk simulation flow diagram.

Table 1 Dynamics models

Model	Translation	Attitude
Multibody gravity	×	
Earth J_2	×	
Facet drag	×	×
Random force and torque	×	×
Reaction wheels		×
Thrusters	×	×

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 $Q^*(s_i, a_i)$.

$$V^*(s_i) = \max_a Q^*(s_i, a_i) \tag{9}$$

If the optimal state–action value function is known, the optimal deterministic policy is found with Eq. (10).

$$\pi^*(s_i) = \arg\max_a Q^*(s_i, a_i) \tag{10}$$

In this work, the focus is on solving for the state–action value function $Q(s_i, a_i)$. Many algorithms have been posed to solve MDPs. Two of the most well-known algorithms, policy iteration [32] and value iteration [33], 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. Although these techniques provide exact solutions, they require an explicit transition function and discrete state space. Conversely, online algorithms solve MDPs 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 only require a generative model to represent the transition function, which is well suited for spacecraft operations planning. Therefore, this work solves for the state–action value function using an online algorithm, MCTS.

Table 2 Spacecraft parameters

Parameter	Value
General spacecraft parameter	
Mass	330 kg
Dimensions	$1.38\ \mathrm{m} \times 1.04\ \mathrm{m} \times 1.58\ \mathrm{m}$
Inertia	$\mathrm{diag}(82.1, 98.4, 121)\ \mathrm{kg \cdot m^2}$
Power system	
Solar panel area	$1.0\ \mathrm{m^2}$
Solar panel efficiency	0.20
Instrument power draw	30 W
Transmitter power draw	15 W
Battery capacity	$80\ \mathrm{W \cdot h}$
Attitude control system	
Maximum wheel speeds	6000 RPM
Maximum momentum	$50\ \mathrm{N \cdot m \cdot s}$
Maximum wheel torque	$0.2\ \mathrm{N \cdot m}$
Maximum thrust	0.9 N
Thruster minimum on time	0.02 s
RW1 spin axis	$\hat{\boldsymbol{b}}_1$
RW2 spin axis	$\hat{\boldsymbol{b}}_2$
RW3 spin axis	$\hat{\boldsymbol{b}}_3$
Data and communications system	
Data buffer storage capacity	1 GB
Instrument baud rate	4 Mbps
Transmitter baud rate	4 Mbps

Table 3 Ground station parameters

Location	Latitude	Longitude	Elevation, m	Minimum elevation angle, deg
Boulder, CO (USA)	40.015 N	105.27 W	1600	10
Ka Lae, HI (USA)	19.897 N	155.58 W	9.0	10
Merritt Island, FL (USA)	28.318 N	80.666 W	0.91	10
Singapore	1.3521 N	103.82 E	15	10
Weilheim, Germany	47.841 N	11.142 E	560	10
Santiago, Chile	33.449 S	70.669 W	570	10
Dongara, Australia	29.245 S	114.93 E	34	10

Table 4 Flight modes

	Mode			
Basilisk task and model	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

B. Monte Carlo Tree Search

MCTS, specifically the 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 [34] and shown in Fig. 4. 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.

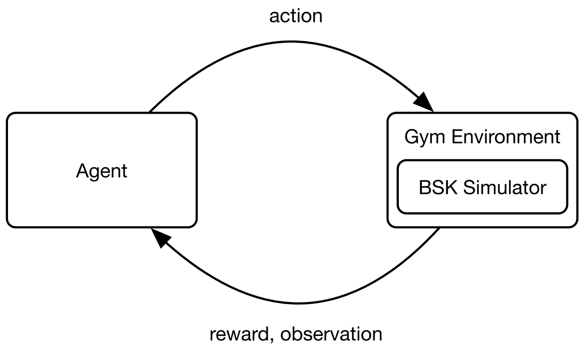


Fig. 3 OpenAI gym framework.

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 horizon 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 the end of the planning interval is reached.

Two rollout policies are explored: random and heuristic. The random rollout policy selects random actions unless a downlink opportunity is present (i.e., any of the states $q_{1:7}$ are nonzero), which initiates a downlink mode. The heuristic rollout policy considers the states most relevant to constraint violations to determine which action will guarantee that 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 Eqs. (11–14). A summary of the safety limits in comparison to the failure limits, both with and without units, are displayed in Table 5. Although the body rate of the spacecraft is not considered in the reward function, the body rate is important for determining when the desaturation mode should be taken over spacecraft charging when several other safety states have exceeded their limits. 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 and Schaub employ for shielded deep RL in the EOS scheduling problem [19]. However, Harris and Schaub use the shield to conservatively bound an 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 by avoiding constraint violations during rollout.

$$|{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}| \geq 1\text{e-}2\ \mathrm{rad/s} \rightarrow \text{Tumbling} = \text{True} \tag{11}$$

$$|\boldsymbol{\Omega}| \geq 400\ \mathrm{rad/s} \rightarrow \text{Saturated} = \text{True} \tag{12}$$

$$Z \leq 40\ \mathrm{Whr} \rightarrow \text{Low Power} = \text{True} \tag{13}$$

$$B \geq 0.8\ \mathrm{GB} \rightarrow \text{Buffer Limit} = \text{True} \tag{14}$$

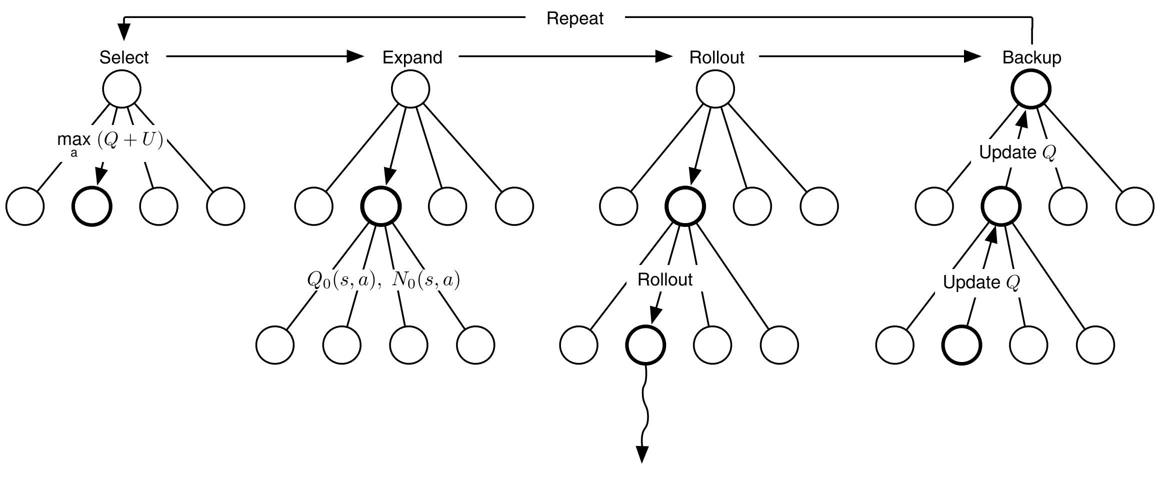


Fig. 4 Upper confidence bound for trees.

Table 5 Failure and safety MDP limits

Resource state	Failure limits		Safety limits	
	Unit	Unitless	Unit	Unitless
Wheel speeds	628 rad/s	1	400 rad/s	0.64
Battery level	0 W · h	0	40 W · h	0.5
Data buffer level	1 GB	1	0.8 GB	0.8

Table 6 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

Based on the state of the safety MDP, an expert-derived 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 6. Like the random rollout policy, if the image action is selected (which means that the system state is nominal in terms of the safety variables), but any of the ground station access variables $q_{1:7}$ are nonzero, downlink is initiated instead.

C. State–Action Value Function Neural Network Regression

The ultimate goal of the state–action value function neural network regression is to generalize the state–action value functions computed using MCTS with a neural network representation, $Q_\theta(s_i, a_i)$. MCTS is used to generate search trees that are regressed over using different neural network architectures. Only the state–action value pairs associated with states that MCTS executes planning from are used for neural network regression. In other words, only the main search tree is used because states far removed from the main tree are visited very few times, resulting in poor state–action value estimates. In order for the trees generated by MCTS to be used, the intermediate state–action value pairs found using MCTS must be updated with the realized reward after MCTS finishes stepping through a planning problem. Once an entire planning problem has been solved by MCTS, the reward received while stepping through the environment is used to compute new state–action 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 Fig. 5, where the realized reward is backed up through the main search tree.

Figure 6 depicts the process 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. 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 are used for training and 10% is used for validation to monitor overfitting. After generation and assembly of the data, neural networks are trained to produce state–action value function approximators, $Q_\theta(s_i, a_i)$. 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 is selected. The neural network policy is the selection of the action associated with the highest state–action value pair at each step through the environment. This is represented in Eq. (15):

$$\pi_\theta(s_i) = \arg\max_{a_i} Q_\theta(s_i, a_i) \tag{15}$$

Table 7 shows the distributions that the initial conditions are drawn from for training purposes. The eccentricity, inclination, longitude of the ascending node, argument of periapsis, and true anomaly are drawn from fairly large distributions such that the agent experiences a range of low Earth orbits. The semimajor axis is initialized to 6871 km in all cases. Although some variation in altitude is inherent due to the eccentricity and the previously described perturbations, a distribution for the semimajor axis is not included to demonstrate how the agent generalizes to semimajor axes above 6871 km. This allows for the robustness of a single semimajor axis solution to be tested over a range of mission altitudes. Also provided in Table 7 are the initial conditions for the spacecraft attitude and rate, reaction wheel speeds, and initial battery charge. The data buffer always begins with no data collected. A planning horizon, $t_{\max}$, of 270 min (approximately three orbits) is selected to balance computation time with a challenging operational scenario.

V. Results

A. Monte Carlo Tree Search Hyperparameter Search

A hyperparameter search is conducted to determine the best MCTS hyperparameter combination for generating neural network training data. In the following search, different combinations of two key

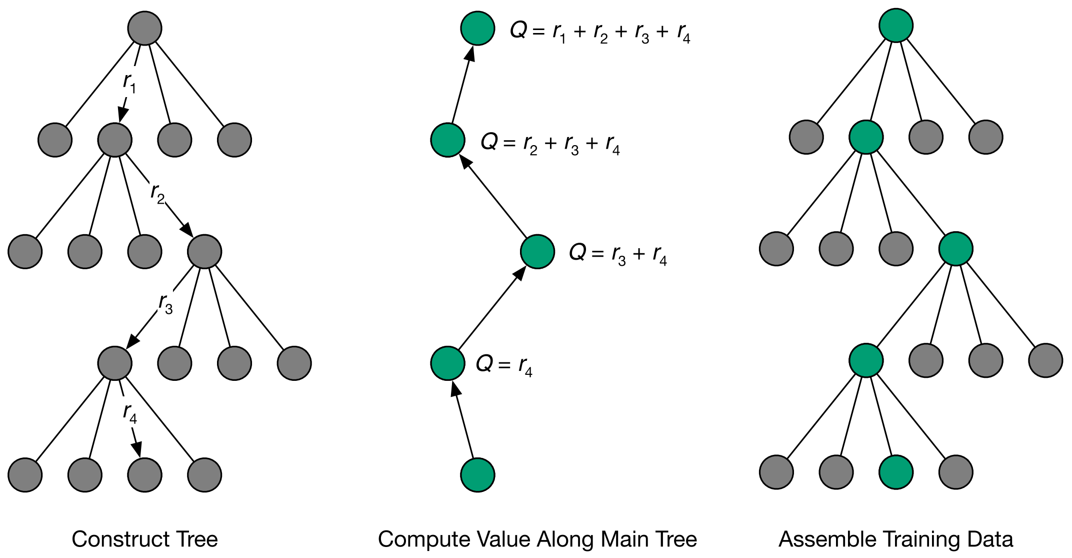


Fig. 5 UCT tree value computation.

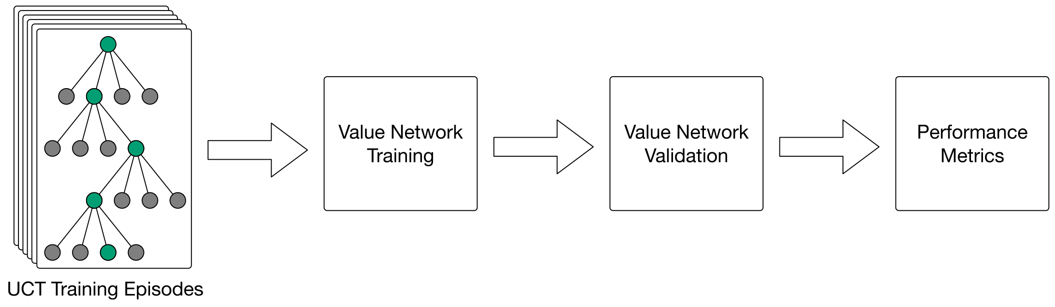


Fig. 6 Value network training.

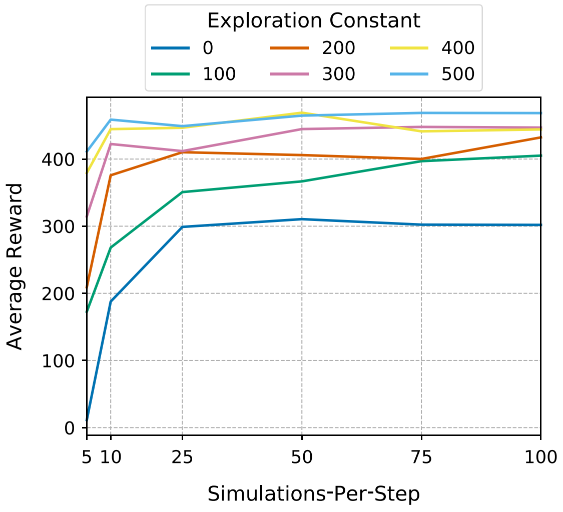
Table 7 Mission simulation parameters

Parameter	Value
Orbit parameters	
Semimajor axis, a	6871 km
Eccentricity, e	$\mathcal{U}[0, 0.01]$
Inclination, i	$\mathcal{U}[40, 60]$ deg
Longitude of ascending node, Ω	$\mathcal{U}[0, 20]$ deg
Argument of periapsis, ω	$\mathcal{U}[0, 20]$ deg
True anomaly, f	$\mathcal{U}[0, 360]$ deg
Spacecraft parameters	
Disturbance torque, $\boldsymbol{\tau}_{\text{ext}}$	2×10^{-4} N · m
Attitude initialization, $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$	$\mathcal{U}[0, 1.0]$ rad
Rate initialization, ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$	$\mathcal{U}[-1e\text{-}05, 1e\text{-}05]$ rad/s
Reaction wheel speeds	$\mathcal{U}[-4000, 4000]$ RPM
Initial battery charge	$\mathcal{U}[30, 50]$ W · h
Planning horizon	
Maximum planning horizon, $t_{\max}$	270 min
Planning interval, t_{mode}	6 min
Integration time step, Δt	1 s

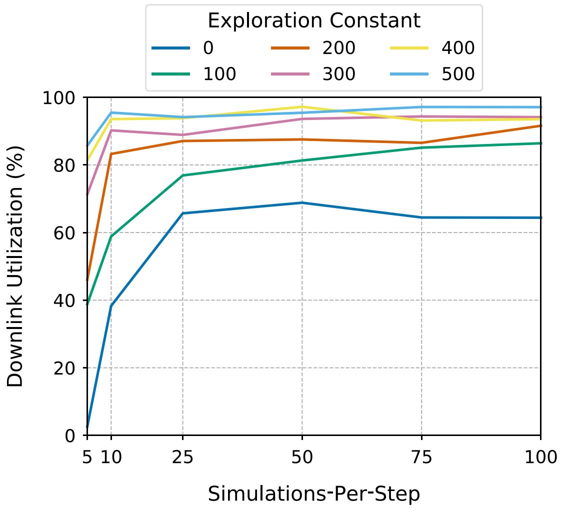
parameters for MCTS are tested: the exploration constant c and the number of simulations-per-step. The exploration constant scales the exploration bonus during the search step, and the number of simulations-per-step determines how many simulations MCTS executes at each step through the environment. The hyperparameter search is also conducted for both types of rollout policies described in Sec. IV.B: random and heuristic. Each hyperparameter combination is evaluated based on average episodic reward, downlink utilization, and the resource management success rate. The downlink utilization is a measure of how effectively the agent uses downlink opportunities. It is defined as the amount of time the agent downlinks data (i.e., sends data to a ground station, not spends in the downlink mode) divided by the amount of time downlink windows are available (i.e., the amount of time the spacecraft is within the ground station's field of view as specified by the elevation in Table 3).

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. Although 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 to the optimal action as the number of simulations-per-step approaches infinity [23]. While MCTS achieves acceptable performance for this problem at 10 simulations-per-step, it takes at least an order of magnitude more simulations-per-step to converge to the optimal solution.

The same hyperparameter search is conducted for a random rollout policy, and the results are provided in Fig. 8. Both average reward and downlink utilization are much lower than the values generated by the heuristic rollout policy. In Fig. 9, the resource management success rate is shown. Fifty simulations-per-step is the minimum required number for most exploration constants to achieve a 100% success

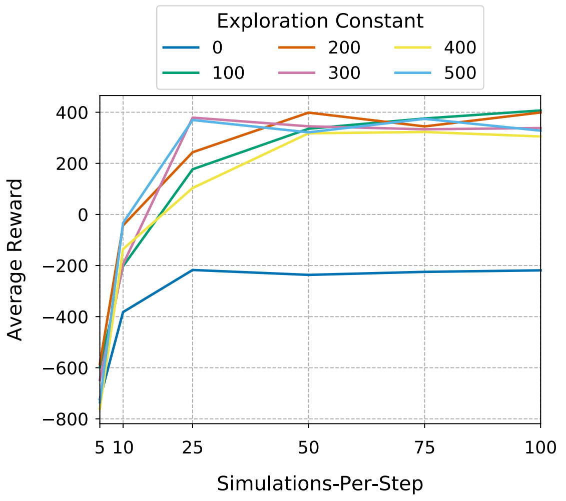


a) Average Episodic Reward

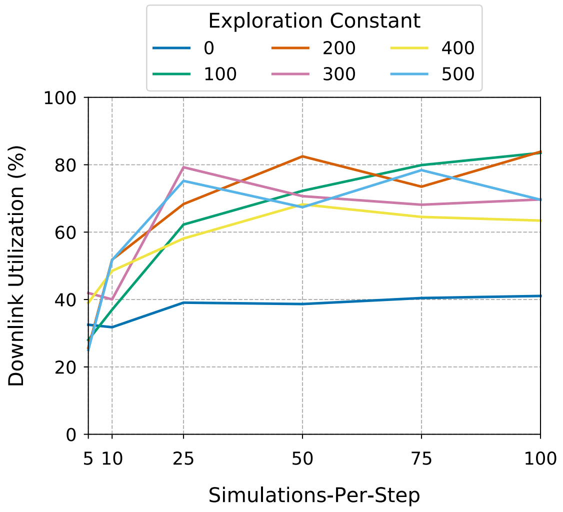


b) Average Downlink Utilization

Fig. 7 UCT hyperparameter search: Heuristic Rollout Policy.



a) Reward



b) Downlink Utilization

Fig. 8 UCT hyperparameter search: Random Rollout Policy.

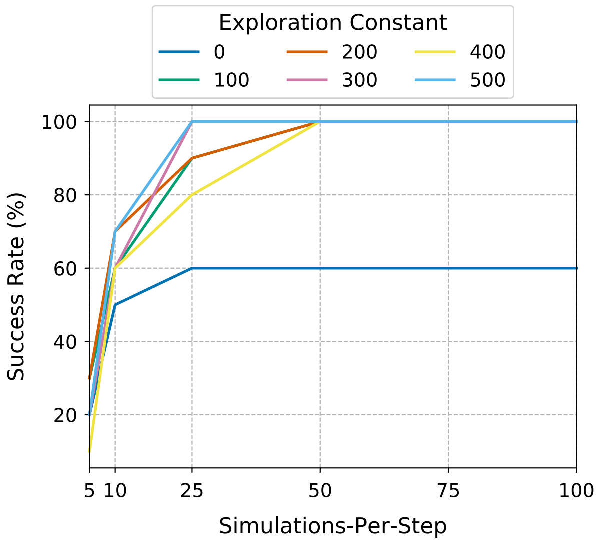


Fig. 9 Resource management success rate.

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. As a result, MCTS only explores states where there is high reward as the resource constraint violations have been avoided. Conversely, the random rollout policy provides no guarantees on avoiding resource constraints, so MCTS spends its simulation budget learning where the low reward states are.

B. State–Action Value Network Hyperparameter Search

As described in Sec. IV.C, state–action value training data are generated by MCTS. The selected MCTS hyperparameters for generating these data are an exploration constant of $c = 500$, 10 simulations-per-step, and a heuristic rollout policy. The selected hyperparameters balance the quality of the solutions with total execution time. For reference, MCTS can generate a solution for a single initial condition in 25 min using 10 simulations-per-step. This can be extrapolated linearly for more simulations-per-step, demonstrating the need to keep this parameter as low as possible. The training data are generated using 1200 unique initial conditions. To determine the best neural network architecture,

Table 8 Leaky ReLU general architecture search

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

a hyperparameter search is conducted over the number of hidden layers, the number of nodes per hidden layer, the activation function, dropout rate, and hyperparameters specific to the activation function. Each network is trained over 3000 epochs using a mean squared error (MSE) loss function.

The first hyperparameter search explores the performance of the state–action value networks when varying the number of hidden layers, the number of nodes per hidden layer, and activation functions of each layer. The dropout rate, which is the probability that a node will be dropped during a training epoch to avoid overfitting [35], is held constant at 0.25. Furthermore, the α parameter for Leaky ReLU is kept at the default of 0.3. The parameter α controls the slope of the Leaky ReLU activation function for $x < 0$. The performance is benchmarked using total reward, downlink utilization, and total time to execute. In Table 8, the performance of each Leaky ReLU network architecture is provided. For each architecture, the top number is the average reward, and the bottom number is the average downlink utilization. An identical study using a hyperbolic tangent activation function is not shown due to relatively poor performance. 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 are preferred to increase the speed of training and execution.

A more detailed hyperparameter search is performed to determine the best combination of parameters when the number of nodes per hidden layer is held constant at 250 nodes per layer. The dropout rate, the number of hidden layers, and α are all varied during the hyperparameter search. As demonstrated in Table 9, the overall architecture is relatively robust to the hyperparameters. Each dropout rate produces networks that achieve greater than 95% downlink utilization. Furthermore, each number of hidden 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.

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 networks have learned which planet-centered, planet-fixed position and velocity vectors are correlated with ground station access. In Fig. 10, the average time (expressed as the percent time over each planning interval) several agents spend in each mode averaged over the 100 initial conditions is shown. For the purposes of readability, only 10 agents that demonstrate the breadth of solutions are selected. Furthermore, all of the selected agents are from Table 9 and achieve greater than 95% downlink utilization. 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 and 40% of the time in the downlink mode but achieve 95% downlink utilization. Other architectures spend 60–70% of the time in the downlink mode to achieve the same performance. Downlink windows are available for an average of 5.98% of the planning horizon, so in both cases the agents are spending more time attempting to downlink than is necessary. However, it is unlikely that the agents are randomly achieving this high reward considering the consistency of the performance over the 100 test conditions. This suggests that they have learned where the ground stations are located in terms of the planet-centered, planet-fixed position and velocity vectors to some degree. This is encouraging for future work, especially for multiple targets scenarios in which the radius and velocity of multiple targets are input states and each target is its own action.

Table 9 Leaky ReLU architecture: 250 nodes per layer

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

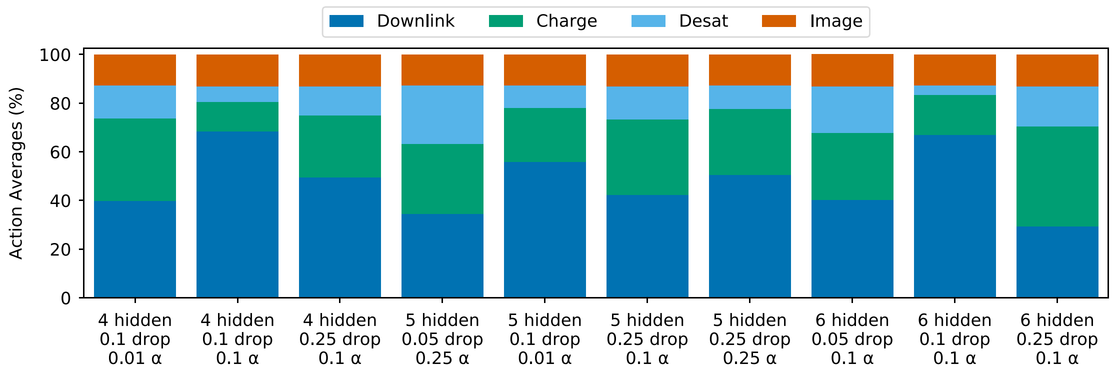


Fig. 10 Average action percentages: specific search.

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 Fig. 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 are limited by the size of the data buffer, and the other spacecraft modes do not include a penalty (other than power draw), so the spacecraft can split its time between the other three modes, converging to various local minima while achieving the same performance. Another learned behavior demonstrated by a few networks highlights the dependence on the activation function. When the hyperparameter search in Table 8 is repeated for a hyperbolic tangent activation function, 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 $Q_\theta(s_i, a_i)$.

C. Robustness

The data presented in the previous section make a strong case for generalization within the training data distributions provided in Table 7. However, the training data distributions do not cover all low Earth orbits. Specifically, the semimajor axis is always initialized to 6871 km. Furthermore, the initial epoch is held constant in training. Each planning horizon begins on May 4, 2021. In this section, the effects of an erroneous orbit insertion into a higher semimajor axis orbit and a changing epoch are studied.

The performance of six neural network architectures is measured as the change in semimajor axis is increased from 0 to 2000 km in increments of 100 km. In Fig. 11, the average episodic reward increases with the semimajor axes of the orbits until Δa is between 500–750 km. This initially happens because the agents have more ground station access as the semimajor axis increases. Average episodic reward then begins to decrease as resource constraint failures begin to occur, as shown in Fig. 12. The reward decreases before reaching a local minima, where the reward penalty for failing to manage resources is offset by the increase in the length of the downlink windows. The resource management failures occur almost entirely due to data buffer overflows. Although this may seem nonintuitive at first due to the increase in the length of the downlink windows, consider the fact that the frequency of new downlink windows decreases because of the larger semimajor axis. The agent anticipates upcoming downlink opportunities based on the planet-centered, planet-fixed position and velocity vectors.

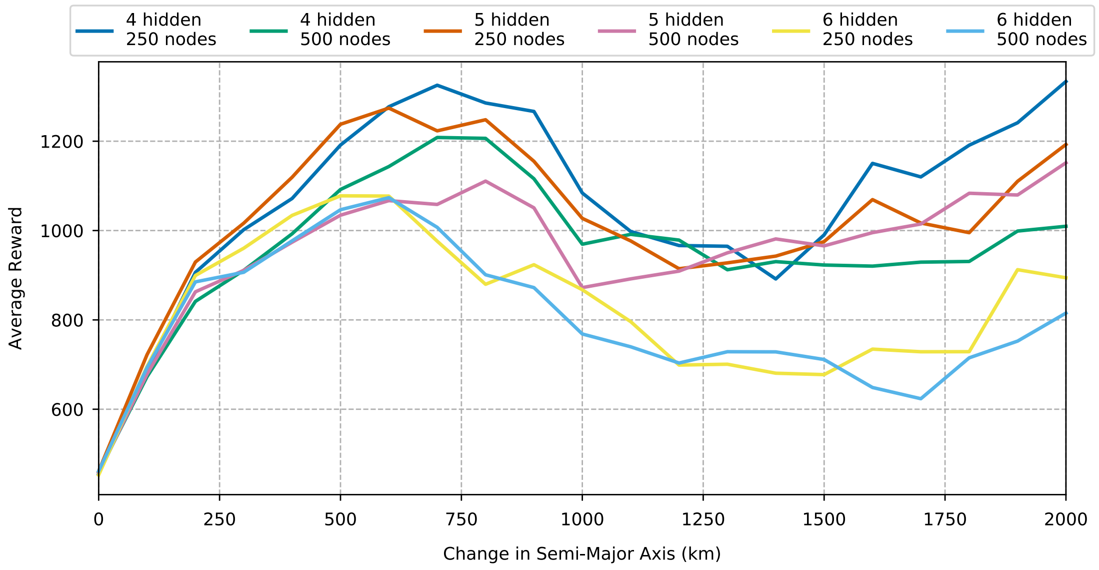


Fig. 11 Average episodic reward as a function of semimajor axis.

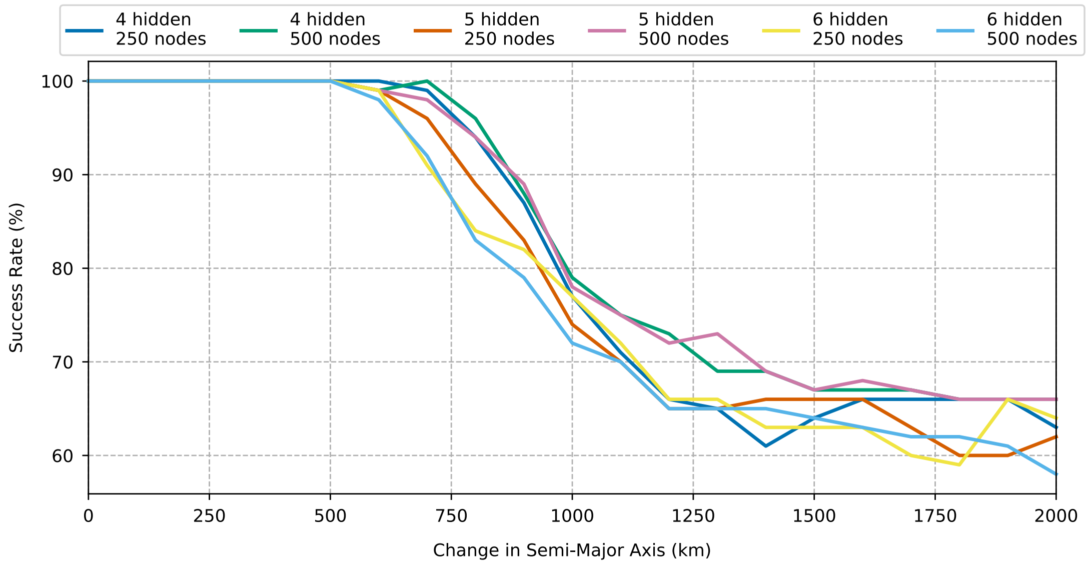


Fig. 12 Resource management success rate as a function of semimajor axis.

However, these do not occur and the agent overflows the data buffer. To rectify this issue, a state would need to be included in the state space that quantifies the relative size of the semimajor axis within the training distribution, normalized between [0, 1]. Regardless of the resource constraint violations for large deviations in semimajor axis, the trained neural networks can safely generalize up to a Δa of 500 km.

The initial epoch of each simulation is moved 3, 6, and 9 months forward in time, and the effect on performance, specifically resource management, is studied. When the epoch is moved to 3 and 9 months in the future, some network architectures from Table 9 fail to manage power on-board the spacecraft and drain the battery, receiving a large reward penalty. For the 3-month change in epoch, 13 of the 36 trained agents produce resource management failures. For the 9-month change in epoch, 18 of the 36 trained agents produce resource management failures. When the epoch is moved 6 months into the future, the performance of the agents in Table 9 degrades the most; 29 of the 36 agents produce resource management failures. This is largely due to the change in the position of the sun relative to the Earth and the spacecraft and some agents becoming overfit on the relative position of the three during training. Furthermore, some agents anticipate ground station availability that never happens, suggesting that they are overfit on the state that represents the percent of the planning horizon that has passed. This demonstrates the need to vary the epoch during training to prevent this type of overfitting. However, it is encouraging that a number of agents generalize to epochs outside of the training distributions without producing failures.

D. Genetic Algorithm Comparison

To determine the optimality gap of MCTS 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, whereas 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 with respect to the reward function. In the interest of time, only the first 10 out of 100 initial conditions are used. The DEAP evolutionary computation framework (https://deap.readthedocs.io/en/master/) 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 Sec. II. The crossover and mutation probabilities are each set to 0.25. The number of generations and population size are varied between 45 and 200 and between 10 and 20, respectively. In Table 10, the genetic algorithm achieves the optimal solution of 472 reward and 99.8% downlink utilization.

E. Final Comparison

Table 11 displays the reward, downlink utilization, execution time, and total number of Basilisk simulations required for each MCTS method implemented in this paper. The hyperparameter combination that achieves the highest reward is selected for each method. Heuristic MCTS achieves its maximum reward at 75 simulations-per-step with an optimality gap of 0.64%. After training, the neural networks achieve near-optimal performance with an optimality gap of 1.3%. However, the total execution time is several orders of magnitude less than any other algorithm implemented in this work. The value network is the only candidate algorithm in this work for on-board execution where execution speed is paramount on resource-constrained flight processors. Blacker et al. demonstrate that neural networks of a comparable size can execute on radiation hardened processors like the LEON3 in under 10 s [36].

The number of Basilisk simulations refers to the number of simulations required to generate one solution that achieves the demonstrated performance metrics in Table 11. For MCTS, this is computed by multiplying the number of simulations-per-step by the total number of planning intervals. Note that the state–action value networks achieve near-optimal performance by interacting with the environment only one time, and never with a simulated environment, selecting an action after each observation. MCTS and the genetic algorithm require many simulated environment interactions to solve the planning problem. Furthermore, the genetic algorithm yields open-loop tasking solutions. Although MCTS technically yields closed-loop tasking solutions, it is given the truth model of the environment for the purposes of this work. Because of 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 in a closed-loop implementation.

Table 10 Genetic algorithm performance

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

Table 11 Comparison of algorithms

Metric	Random MCTS	Heuristic MCTS	Value network
Average reward	407	469	466
Average downlink utilization, %	83.9	97.1	96.7
Average execution time, s	19,100	11,400	0.0672
No. of simulations	4,500	3,375	0

VI. Conclusions

This work successfully demonstrates the use of MCTS and state–action value function approximation with neural networks to solve the EOS scheduling problem with resource constraints. The performance of MCTS is investigated to determine the best hyperparameter combination for training by varying the rollout policy, exploration constant, 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 function estimates generated by MCTS are regressed over using a variety of neural network architectures. The performance of each network architecture is benchmarked, and it is shown that 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 parameter. The learned behavior of each network is explored, demonstrating that the networks have learned to manage resource constraints and the locations of the ground stations in regard to their planet-centered, planet-fixed position vectors to some degree. Additionally, the neural network architectures demonstrate generalizability within the training distributions and robustness outside of the training distributions. Finally, MCTS and state–action value network regression are compared with a genetic algorithm, which provides an upper bound on reward. The state–action value networks achieve comparable performance to both the genetic algorithm and MCTS, but with a six orders of magnitude reduction in execution time after training, making a state–action value network the only candidate for on-board, autonomous spacecraft planning.

Acknowledgments

This work is supported by a NASA Space Technology Graduate Research Opportunity grant (80NSSC20 K1162). The authors would like to acknowledge the AVS Laboratory for developing the Basilisk Astrodynamics Software Framework. The authors would also like to thank Andrew Harris, who developed a Basilisk Gym Environment and genetic algorithm runner the author could modify for this version of the Earth-observing satellite scheduling problem.

References

[1] Amini, R., Castillo-Rogez, J., and Day, J., "Advancing the Scientific Frontier with Increasingly Autonomous Systems," *Lunar and Planetary Science Conference*, AIAA Paper 2020-2503, 2020.

[2] Shah, V., Vittaldev, V., Stepan, L., and Foster, C., "Scheduling the World's Largest Earth-Observing Fleet of Medium-Resolution Imaging Satellites," *International Workshop on Planning and Scheduling for Space*, Organization for the 2019 International Workshop on Planning and Scheduling for Space, Berkeley, CA, 2019, pp. 156–161.

[3] Berger, R. W., Bayles, D., Brown, R., Doyle, S., Kazemzadeh, A., Knowles, K., Moser, D., Rodgers, J., Saari, B., Stanley, D., and Grant, B., "The RAD750—A Radiation Hardened PowerPC Processor for High Performance Spaceborne Applications," *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*, Vol. 5, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 2001, pp. 2263–2272. https://doi.org/10.1109/AERO.2001.931184

[4] Fukunaga, A., Rabideau, G., Chien, S., and Yan, D., "Towards an Application Framework for Automated Planning and Scheduling," *1997 IEEE Aerospace Conference*, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 1997, pp. 375–386. https://doi.org/10.1109/AERO.1997.574426

[5] Knight, S., Rabideau, G., Chien, S., Engelhardt, B., and Sherwood, R., "CASPER: Space Exploration Through Continuous Planning," *IEEE Intelligent Systems*, Vol. 16, No. 5, 2001, pp. 70–75. https://doi.org/10.1109/MIS.2001.956084

[6] Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davis, A., Mandl, D., Frye, S., Trout, B., Shulman, S., and Boyer, D., "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. https://doi.org/10.2514/1.12923

[7] Chien, S., Tran, D., Rabideau, G., Schaffer, S., Mandl, D., and Frye, S., "Timeline-Based Space Operations Scheduling with External Constraints," *20th International Conference on Automated Planning and Scheduling*, Assoc. for the Advancement of Artificial Intelligence, Menlo Park, CA, 2010, pp. 34–41.

[8] Chien, S., Doubleday, J., Thompson, D. R., Wagstaff, K., Bellardo, J., Francis, C., Baumgarten, E., Williams, A., Yee, E., Stanton, E., and Piug-Suari, J., "Onboard Autonomy on the Intelligent Payload EXperiment (IPEX) CubeSat Mission," *Journal of Aerospace Information Systems (JAIS)*, Vol. 14, No. 6, 2016, pp. 307–315. https://doi.org/10.2514/1.I010386

[9] Chien, S. A., Davies, A. G., Doubleday, J., Tran, D. Q., Mclaren, D., Chi, W., and Maillard, A., "Automated Volcano Monitoring Using Multiple Space and Ground Sensors," *Journal of Aerospace Information Systems*, Vol. 17, No. 4, 2020, pp. 214–228. https://doi.org/10.2514/1.I010798

[10] Chien, S., Mclaren, D., Doubleday, J., Tran, D., Tanpipat, V., and Chitradon, R., "Using Taskable Remote Sensing in a Sensor Web for Thailand Flood Monitoring," *Journal of Aerospace Information Systems*, Vol. 16, No. 3, 2019, pp. 107–119. https://doi.org/10.2514/1.I010672

[11] Globus, A., Crawford, J., Lohn, J., and Pryor, A., "A Comparison of Techniques for Scheduling Earth Observing Satellites," *16th Conference on Innovative Applications of Artificial Intelligence*, Assoc. for the Advancement of Artificial Intelligence, Menlo Park, CA, 2004, pp. 836–843.

[12] Spangelo, S., Cutler, J., Gilson, K., and Cohn, A., "Optimization-Based Scheduling for the Single-Satellite, Multi-Ground Station Communication Problem," *Computers and Operations Research*, Vol. 57, No. C, 2015, pp. 1–16. https://doi.org/10.1016/j.cor.2014.11.004

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

[14] Nag, S., Li, A. S., and Merrick, J. H., "Scheduling Algorithms for Rapid Imaging Using Agile Cubesat Constellations," *Advances in Space Research*, Vol. 61, No. 3, 2018, pp. 891–913. https://doi.org/10.1016/j.asr.2017.11.010

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

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

[17] Chan, D. M., and Agha-mohammadi, A., "Autonomous Imaging and Mapping of Small Bodies Using Deep Reinforcement Learning," *2019 IEEE Aerospace Conference*, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 2019, pp. 1–12. https://doi.org/10.1109/AERO.2019.8742147

[18] Harris, A., Teil, T., and Schaub, H., "Spacecraft Decision-Making Autonomy Using Deep Reinforcement Learning," *AAS Spaceflight Mechanics Meeting*, AAS Paper 19-447, 2019.

[19] Harris, A., and Schaub, H., "Deep On-Board Scheduling for Autonomous Attitude Guidance Operations," *AAS Guidance, Navigation and Control Conference*, AAS Paper 02-117, 2020.

[20] Harris, A., and Schaub, H., "Spacecraft Command and Control with Safety Guarantees Using Shielded Deep Reinforcement Learning," *AIAA SciTech*, AIAA Paper 2020-0386, 2020.

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

[22] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 2018, Chap. 13.

[23] Kocsis, L., Szepesvári, C., and Willemson, J., "Improved Monte-Carlo Search," Univ. of Tartu TR 1, Tartu, Estonia, 2006.

[24] Shah, D., Xie, Q., and Xu, Z., "Non-Asymptotic Analysis of Monte Carlo Tree Search," *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, Assoc. for Computing Machinery, New York, 2020, pp. 31–32. https://doi.org/10.1145/3393691.3394202

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

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

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

[28] Eddy, D., and Kochenderfer, M., "Markov Decision Processes for Multi-Objective Satellite Task Planning," *2020 IEEE Aerospace Conference*, Inst. of Electrical and Electronics Engineers, New York, 2020, pp. 1–12.

[29] Schaub, H., and Junkins, J. L., *Analytical Mechanics of Space Systems*, 4th ed., AIAA Education Series, AIAA, Reston, VA, 2018, pp. 122–132. https://doi.org/10.2514/4.105210

[30] Kenneally, P. W., Schaub, H., and Piggott, S., "Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework," *AIAA Journal of Aerospace Information Systems*, Vol. 17, No. 9, 2020, pp. 496–507. https://doi.org/10.2514/1.I010762

[31] Center, G. S., "Near Earth Network Users' Guide," Rev. 4, NASA TR 453-NENUG, Greenbelt, MD, March 2019.

[32] Howard, R. A., *Dynamic Programming and Markov Processes*, Technology Press of Massachusetts Inst. of Technology, Cambridge, MA, 1960, Chap. 4.

[33] Bellman, R., "A Markovian Decision Process," *Indiana University Mathematics Journal*, Vol. 6, No. 4, 1957, pp. 679–684.

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

[35] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, Vol. 15, No. 56, 2014, pp. 1929–1958, http://jmlr.org/papers/v15/srivastava14a.html.

[36] Blacker, P., Bridges, C., and Hadfield, S., "Rapid Prototyping of Deep Learning Models on Radiation Hardened CPUs," *2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Inst. of Electrical and Electronics Engineers, New York, 2019, pp. 25–32. https://doi.org/10.1109/AHS.2019.000-4

M. J. Kochenderfer
Associate Editor