AAS 20-473

MONTE CARLO TREE SEARCH WITH VALUE NETWORKS FOR AUTONOMOUS SPACECRAFT OPERATIONS

Adam Herrmann* **and Hanspeter Schaub†**

On-board spacecraft task scheduling is a requisite capability for the autonomous exploration of challenging operational environments. Task execution time and resource usage are heavily dependent on the dynamics of the system. Typically, solutions to the spacecraft task scheduling problem consider a limited model of the spacecraft tasks and dynamics or require large amounts of computational power, severely limiting on-board planning and scheduling capability. This paper proposes offline, high-fidelity dynamics simulations and Monte Carlo tree search (MCTS) to compute solutions to the Earth-observing satellite scheduling problem, which are then generalized in a state-action value function approximator that can be executed in real-time onboard spacecraft. MCTS and the state-action value function approximator are deployed in a Basilisk Astrodynamics Software Architecture simulation and compared on the basis of reward, downlink utilization, and computational overhead.

INTRODUCTION

Spacecraft task scheduling is a crucial activity that must be performed for all missions. Many researchers investigate ground-based scheduling techniques, but few investigate on-board task scheduling techniques that grant the spacecraft full control over operational decisions. Full operational control eliminates planning latency, reduces operational cost, and allows the spacecraft to respond to unexpected events and opportunities. However, several challenges preclude full operational autonomy. Limited on-board computational resources and the requirement for verifiable safety guarantees necessitate solutions to on-board scheduling problems that are safe, repeatable, and can be executed using little computational resources.

Of particular interest for autonomous operations is the Earth-observing satellite (EOS) scheduling problem, where one or more Earth-orbiting satellites must make operational decisions to collect and downlink data to one or more ground stations. On-board solutions to the Earth-observing satellite scheduling problem are typically limited in ability but provide large value to spacecraft missions. Chien et al. discuss how weekly ground-based planning and on-board plan repair is used on the Earth-Observing One mission, saving over $1M per year in operations cost.[1] This on-board planner uses a limited model of the spacecraft tasks and is reserved for replanning activities only if a ground-based plan fails. More recently, Chien et al. show that ground-based planning tools can be combined with continuous activity planning on board the IPEX CubeSat to successfully manage 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.

resources such as power, file storage, and downlink bandwidth in the event that onboard replanning is required.[2] While this solution does not achieve full on-board autonomy, it shows that resource management may be accomplished with on-board scheduling software.

Ground-based spacecraft scheduling techniques have been studied extensively, but typical formulations of the problem consider limited operational modes or data, power, and attitude control system dynamics.[3–5] Furthermore, the techniques used to solve these formulations preclude on-board planning due to computational complexity and the types of planning tools used, e.g. Systems Tool Kit (STK). For example, 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.[6] Spangelo et al. assume linear dynamics between decision intervals and develop techniques better suited for ground-based planning due to the disparate tools used for planning.

While other formulations of the EOS scheduling problem make linearity assumptions or consider only limited operational decisions, this formulation of the EOS scheduling problem limits linearity assumptions and includes operational decisions outside of downlinking data. Basilisk*, an open-source astrodynamics and flight software simulation architecture, is leveraged to construct a complex spacecraft operations simulation that couples the astrodynamics, attitude dynamics and controls, and power and data management of an Earth-observing satellite.[7] The high-fidelity Basilisk simulation eliminates many assumptions and provides realistic data to the algorithms discussed within this paper.

In recent years, reinforcement learning has become a viable option for operations in physical systems due to advances in the field. High-fidelity dynamics simulations can be leveraged to train neural networks offline and deploy them on board physical systems for on-the-fly planning. Sadeghi et al. apply reinforcement learning to the collision avoidance problem, using Monte Carlo policy evaluation to train an unmanned aerial vehicle to avoid collisions in the real world with simulation data generated in Blender.[8] In the spacecraft domain, Harris et al. show how offline reinforcement learning methods like Proximal Policy Optimization combined with a safety shield can successfully plan in an Earth-observing satellite environment to collect data while taking into account modes for battery charging and reaction wheel desaturation.[9,10] While Harris et al. focus on shielded techniques that can provide safety guarantees for reinforcement learning agents, this work focuses on Monte Carlo tree search (MCTS) and value function approximation as a start-of-the-art technique for spacecraft scheduling, inspired by the success of similar algorithms at the game of Go. Silver et al. use a similar technique to train a Go-playing agent to achieve superhuman performance.[11] Unlike Silver et al., this paper does not execute the neural network within the MCTS algorithm to continually improve upon the policy and value network. This paper explores using MCTS to generate high-quality value-estimates that are used train a neural network for real-time execution.

In this paper, the EOS scheduling problem is formulated as a Markov decision process (MDP) and high-fidelity Basilisk simulations are leveraged to compute solutions to the EOS scheduling problem. Monte Carlo tree search, specifically the Upper Confidence Bound for Trees (UCT),[12] is applied to step through a Basilisk simulation and compute the schedule for random sets of initial conditions in finite planning horizons. The value estimates computed during Monte Carlo tree search are then used to train a feedforward neural network, which serves as the state-action value

*http://hanspeterschaub.info/basilisk

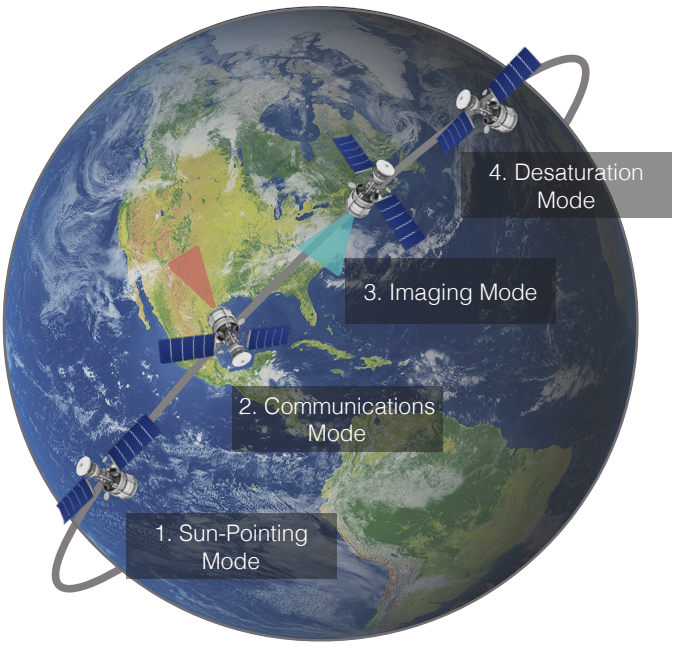


Figure 1: Scheduling of Flight Modes

function approximator. After each training cycle, the state-action value function approximator is used to rapidly compute solutions to the EOS scheduling problem on a test set of initial conditions and performance is measured.

This paper first describes the high-fidelity Basilisk simulation formulated as an OpenAI `Gym` environment that simulates a representative EOS scheduling problem. The MDP formulation of the EOS scheduling problem is discussed, as well as the specific MCTS algorithm used to solve the problem. The design of the feedforward neural network that serves as the state-action value network function approximator is also shown. The problem formulation is then followed up with a discussion of the results and a conclusion.

PROBLEM FORMULATION

Earth-Observing Spacecraft Simulation

The Earth-observing satellite scheduling problem is simulated using the Basilisk Astrodynamics Software Framework. In this formulation of the Earth-observing satellite scheduling problem, a satellite in a 500-km orbit must make operational decisions to collect and downlink science data to any of seven different ground stations around the Earth. 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. Each module falls into one of two categories: flight software or simulation. Each module is added to a separate task grouping, which may be run at its own dynamics rate. There are four flight software tasks - nadir point, MRP control, sun point, and reaction wheel desaturation. Three dynamics tasks - Spice, dynamics, and environment - are also defined. The connections in the diagram represent a message passing interface (MPI) path or standard interface between two or more modules.

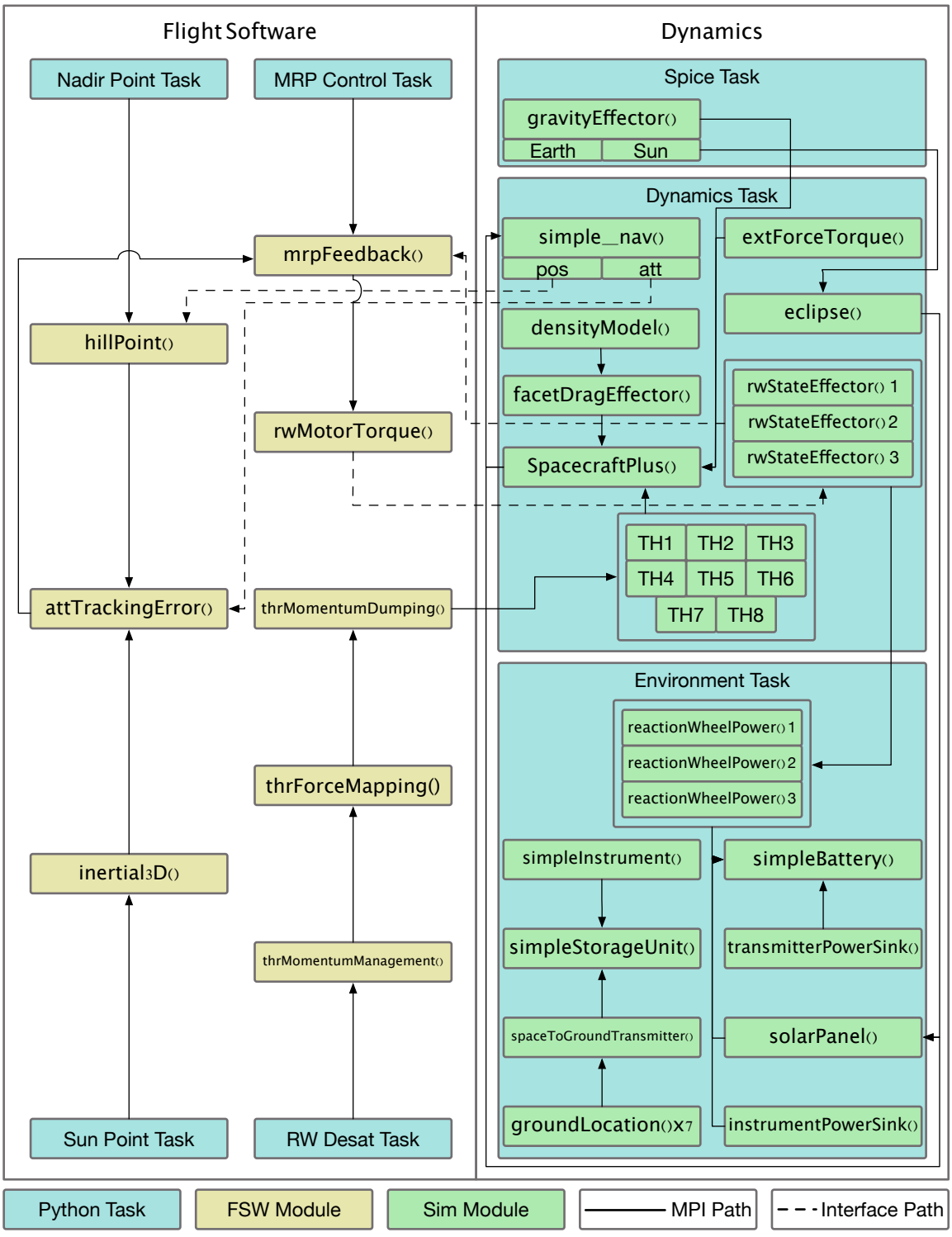


Figure 2: Basilisk Simulation

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 real flight software to be used on board a spacecraft. The simulation architecture described within this paper 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. The `hillPoint` and `inertial3D` modules compute the associated reference frames using position and attitude messages from the `simple_nav` module. The `attTrackingError` module receives the reference frames to compute the MRP tracking error, which is an input to the `mrpFeedBack` module. The `mrpFeedback` module is used to control the reaction wheels onboard the satellite to fulfill the pointing requirements of each flight mode. The `rwStateEffectors` are generated from the Basilisk `rwFactory` and are modeled after the Honeywell HR16 reaction wheels. Momentum management is also simulated using Basilisk. The Basilisk momentum dumping module `thrMomentumDumping` is implemented to map reaction wheel momentums to thruster impulse requests, which are fulfilled by attitude control thrusters. The `thrusterDynamicEffectors` are generated using the Basilisk `thrusterFactory` and are modeled after the Moog Monarc-1 thrusters. The `thrusterDynamicEffectors` are labeled as TH in Figure 2. A fuel budget for momentum dumping is not modeled.

In addition to attitude control, several perturbations are modeled. The Basilisk `densityModel` provides inputs to a facet drag effector module, `facetDragEffector`. An external torque is also applied to the spacecraft to ensure the reaction wheels build momentum over the planning horizon using the `extForceTorque` module.

A power system is simulated in Basilisk, leveraging its high-fidelity dynamics capabilities to accurately compute power consumption and generation. The `solarPanel` module receives state information from `spacecraftPlus` and the eclipse data from the `eclipse` module to compute power generation over a planning interval. Incidence angle, panel efficiency, and panel area are all taken into account. Furthermore, the `instrumentPowerSink`, `transmitterPowerSink`, and `reactionWheelPower` modules are also included in the simulation to model power consumption. Finally, a `simpleBattery` is modeled. The battery receives rates of power consumption from all power nodes and computes net power generation or consumption, maintaining an estimate of the state of charge within the battery.

Lastly, an on-board data management system is modeled. A `simpleInstrument` module is implemented to collect data during observation modes. The instrument sends the data to a `simpleStorageUnit`, which stores the data until a downlink mode is initiated and a ground station is accessible. The `spaceToGroundTransmitter` removes the data from the storage unit and downlinks the data to one of seven `groundLocation` instances, which simulate the different ground stations. 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. Table 2 provides the parameters of each ground station. The ground stations are selected from a list of government and commercial stations utilized by NASA's Near Earth Network.[13] A Boulder, CO ground station is included as well.

The key parameters for each subsystem of the simulated satellite may be found in Table 1. The parameters are selected to create a balanced problem in which the satellite must manage its resources

Table 1: 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	5 W
Transmitter Power Draw	5 W
Battery Capacity	20 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 2: 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

to achieve the objectives within the planning horizon.

The spacecraft tasks are represented by four separate flight modes - observation, downlink, charge, and desaturation. Figure 1 provides an example of the different flight modes. These flight modes are used to manage resource constraints of the simulated spacecraft and achieve the objectives of the mission. During each flight mode, different Basilisk flight software tasks are enabled or disabled. Furthermore, the power and data modules associated with the instrument or transmitter are also turned on or off. Table 3 displays which modules and tasks are enabled and disabled in the flight modes. If a module or task provided in Figure 2 is not listed in the table, it is on for the duration of the simulation.

Gym Environment

Building upon the work in Harris et al, an open-source `Gym` environment* is created to wrap the Basilisk simulation described above. The `Gym` environment allows for an agent to step forward in

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

Table 3: Flight Modes

Mode	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

the simulation by switching into different spacecraft modes (actions). The agent receives a reward based on the action taken and the subsequent system state. After the action is taken and the reward is received, the agent selects a new action and the process continues until the end of the planning horizon or until the agent fails. This process is displayed in Figure 3.

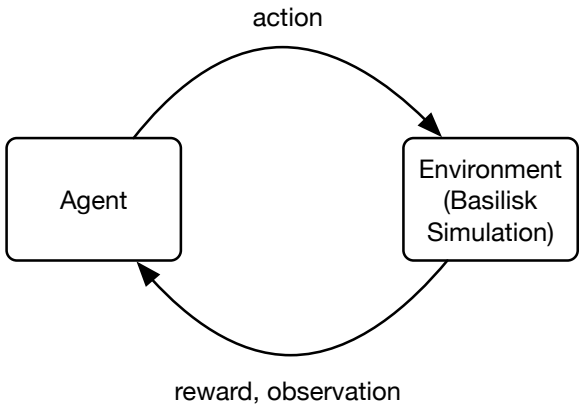


Figure 3: OpenAI Gym Framework

The randomized initial conditions of the simulations are found in Table 4. A narrow range of orbital parameters are selected to expedite training. The eccentricity, inclination, longitude of the ascending node, and argument of periapsis are all bounded so the satellite is restricted to fewer states and the agent does not need to learn how to operate in all low-Earth orbits. The satellite begins with the batteries at less than 50% capacity and no data in the buffer at the start of each planning horizon. All of data that the satellite downlinks to the ground stations is collected during the planning horizon. While this is not necessarily representative of all planning horizons a satellite may encounter, it provides an interesting problem for the algorithms discussed within this paper to solve. All simulations are run on a 2.8 GHz Quad-Core Intel Core i7 with 16 GB of RAM.

A test set of 100 different initial conditions is generated from the table below. The first 10 sets of initial conditions are used to benchmark performance of Monte Carlo tree search. The value network is deployed on this subset of initial conditions as well as the full set.

Markov Decision Process

The Basilisk simulation is formulated as a finite-horizon deterministic Markov Decision Process, defined by the 5-tuple $M = (\mathcal{S}, \mathcal{A}, G, R, \gamma)$. The planning horizon is split up into planning intervals

Table 4: Simulation Parameters

Orbit	
Semi-Major Axis, a	6371+500 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	$\mathcal{U}$[5, 10] Whr
Planning Horizon	
Maximum Simulation Time, t_{max}	90 minutes
General Mode Length, t_{general}	3 minutes
Desaturation Mode Length, t_{desat}	6 minutes

that span the length of the planning horizon. The end of the planning horizon, t_{max}, is defined in minutes. The final interval is defined as the interval that ends with $t_i \geq t_{\text{max}}$.

The variable $\mathcal{S}$ is defined as the state-space for the problem and is shown in Equation (1). 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 so the agent 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 give the agent a sense of when access can be anticipated so the agent can enter a downlink mode before an access window has already passed. 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.

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

To keep the satellite within resource constraints, several other states are added to $\mathcal{S}$. The Modified Rodrigues Parameter (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 agent 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.

$$\mathcal{A} = \{\text{Observation}, \text{Downlink}, \text{Charge}, \text{Desaturation}\} \tag{2}$$

The action-space, $\mathcal{A}$, includes the four separate flight modes previously described - observation, downlink, charge, and desaturation. The observation, downlink, and charging modes last for three minutes each, $\Delta t_{\text{general}}$. A timespan of three minutes was selected to ensure attitude error is negligible by the end of the planning interval. The desaturation mode, however, lasts for a total of six minutes, Δt_{desat}, to give the satellite enough time to dump the momentum in the reaction wheels. During the observation mode, the instrument is pointed in the nadir-direction of Earth to collect data. Data is accumulated in the data buffer and power is consumed at the rates provided in Table 1. During the downlink mode, the transmitter is pointed in the nadir-direction of Earth. Data is only downlinked if a ground station is accessible to the satellite, but the transmitter is powered on for the duration of the interval. During the charging mode, the solar panels are pointed in the direction of the sun to maximize the amount of charge. The transmitter and instrument are turned off.

Several reward functions, R, were explored to produce optimal behavior. The selected reward function is defined as the sum of the amount of data downlinked over each planning interval in megabytes, H_i, multiplied by the discount factor, $\gamma = 0.99$, raised to the interval number, i. A $+1$ success bonus is included in the reward if the agent reaches the end of the planning horizon without failing. The success bonus is included so the agent will finish a planning horizon without failing if no downlink windows remain. A non-discounted success bonus is selected so the satellite does not utilize the desaturation mode to quickly finish the planning interval in a mode that consumes little power.

$$R(s,a) = \begin{cases} H_i \text{ if !failure} \\ H_i + 1 \text{ if } t \geq t_{\text{max}} \text{ and !failure} \\ 0 \text{ if failure} \end{cases} \tag{3}$$

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

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

The transition function, G, is a generative transition function integrated forwards in time by a Basilisk simulation formulated as a `Gym` environment.

METHODS

Monte Carlo Tree Search

Monte Carlo tree search, specifically the Upper Confidence Bound for Trees (UCT),[12] is used to generate spacecraft schedules that maximize the reward over the planning horizon. The full algorithm may be found in Appendix: Algorithms. In UCT, the agent runs a pre-determined number of simulations at each step through the environment to estimate a state-action value function, Q, and associated policy $\pi(s) = \arg\max_a Q(s,a)$ by performing directed searches that exploit intermediate state-action value estimates and undirected searches that execute a rollout policy.[14] The rollout policy selects random actions, but defaults to downlinking data whenever a ground station is in view and there is data within the buffer. Figure 4 demonstrates how UCT performs a single simulation. This process is repeated for a predetermined number of simulations at each step through the planning horizon.

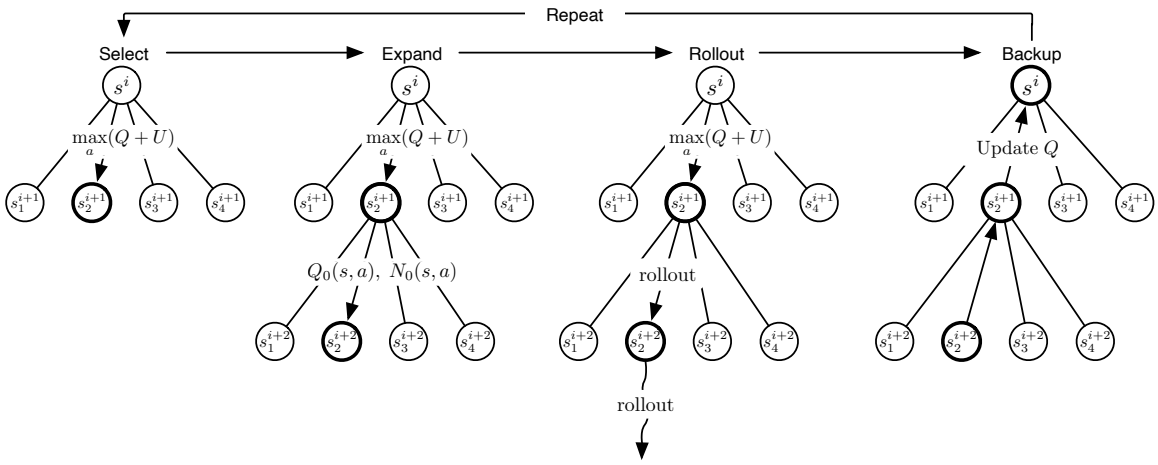


Figure 4: Upper Confidence Bound for Trees

In reinforcement learning, the state-action value function Q is an estimate of how good a certain state and action are - i.e. how much future reward can be expected given the current state and action. The optimal state-action value function is equal to the current reward plus the expected value of the next state. The optimal policy is that which returns the same action as the optimal state-action value function. The optimal state-action value is formally given in Equation (5). This work does not claim to estimate the optimal state-action value function. Instead, this work uses the state-action value function generated in UCT to train a neural network as a state-action value function approximator.

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}[V^*(s')] \tag{5}$$

The depth of search, d, and exploration constant, c, of UCT are both diligently set to achieve optimal results. The depth of the search is set dynamically at each step through the environment so the algorithm always terminates at the end of the planning horizon, t_{max}. An exploration constant $c = 100$ is selected to balance exploration of the search space and exploitation of the intermediate state-action value function.

Value Function Approximation

Q-value estimates are generated by Monte Carlo tree search on randomly generated sets of initial conditions. The action-value estimates are used to train a neural network that serves as a value function approximator. The training pipeline is shown in Figure 5. MCTS is used to compute state-action value estimates and spacecraft task schedules on five sets of initial conditions at a time. After each set of five runs, the state-action value estimates are added to the training set and used to train a feedforward neural network.

A feedforward neural network is constructed using the `Keras` API [*]. All activation functions, loss functions, and dropout functionality described within this section are implemented using the standard functionality available in the `Keras` API. The mean squared error and mean absolute error of the training and validation sets are minimized with the neural network parameters given in Table 5. Dropout, a technique applied to large neural networks that randomly drops neurons and their weights during training, is used for each hidden layer within the neural network to significantly

[*] http://keras.io

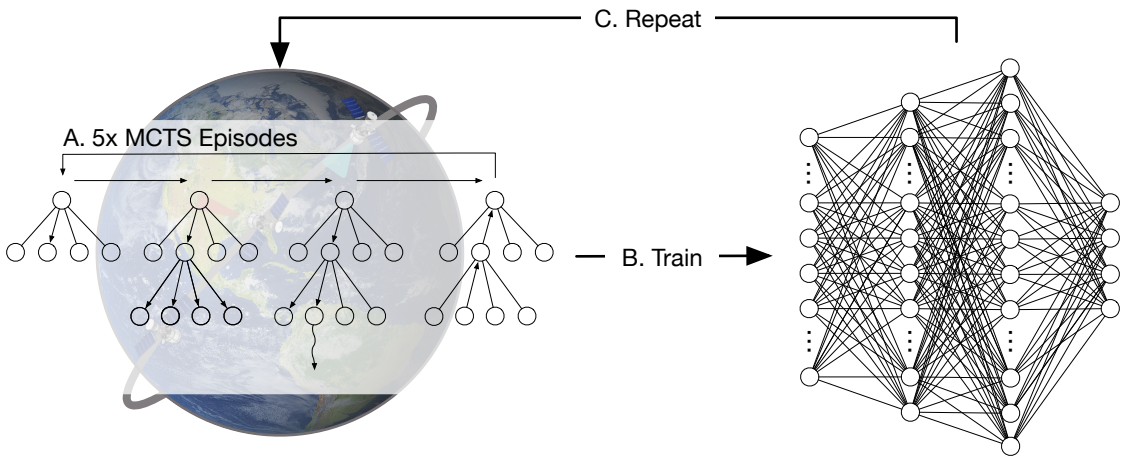


Figure 5: State-Action Value Function Approximation Training Pipeline

Table 5: Feedforward Neural Network Parameters

Layer	Activation	Size	Dropout
Input	—	23	—
Hidden 1	Tanh	100	0.25
Hidden 2	Tanh	150	0.25
Hidden 3	Tanh	600	0.25
Hidden 4	ReLU	200	0.25
Hidden 5	ReLU	100	0.25
Output	—	4	—

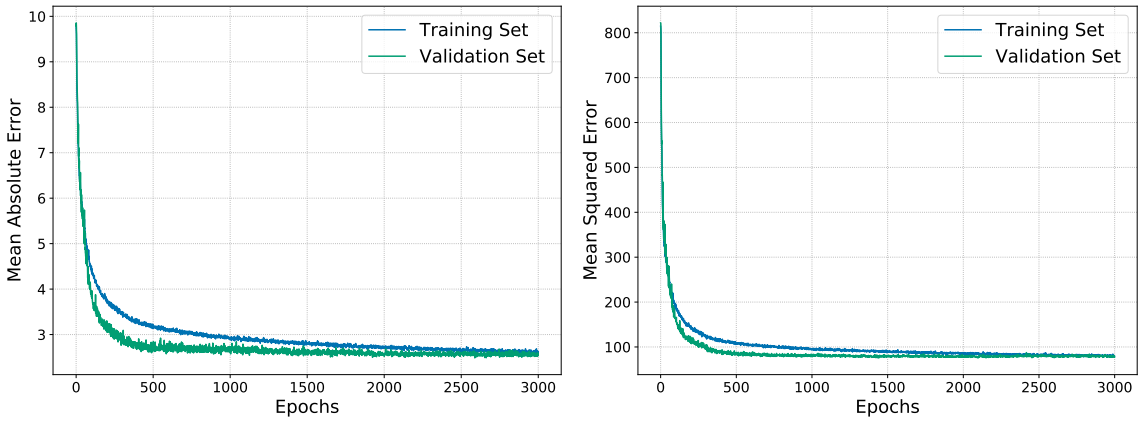
reduce overfitting on the training set and reduce error on the test set.[15] `Tanh` activation functions as the first three hidden layer activation functions and `ReLU` activation functions for the final two hidden layer activation functions also prove to result in better convergence as compared to `Sigmoid` or `Softmax`. `Tanh` activation functions are selected for the first three hidden layers because the state inputs are normalized between [-1, 1], allowing for intermediate computations that are not restricted to an output range of [0, 1]. `ReLU` activation is selected for the last two hidden layers to restrict the neuron output to the range [0, 1]. State-action value should never be less than zero in this MDP formulation.

Network loss and error plots may be found in Figure 6. An `Adam` optimizer using mean absolute error as the loss function results in the best convergence for both mean absolute and mean squared error. Other optimizers and loss functions result in sub-optimal performance for the problem but are not shown here for brevity.

RESULTS

Monte Carlo Tree Search Performance

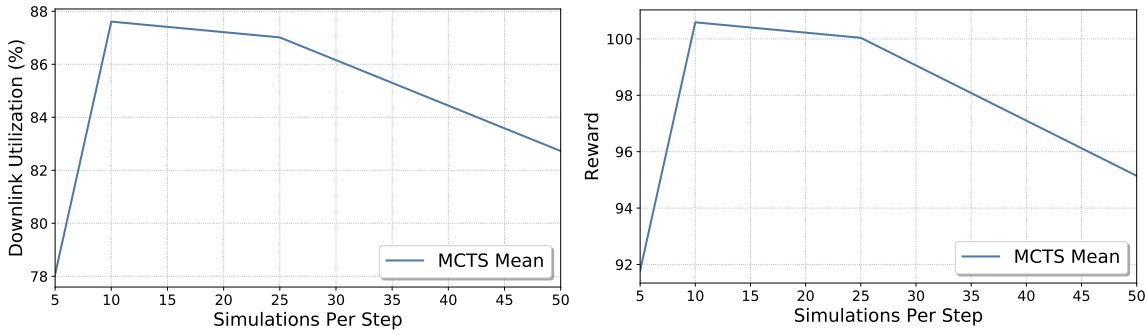
To correctly parameterize the MCTS algorithm and generate a performance baseline, MCTS is executed on the first 10 out of the 100 test sets. Downlink utilization, total reward, and execution time are measured as the number of simulations in MCTS is varied between 5 and 50. Downlink utilization is defined as the amount of time the satellite spends downlinking to the ground stations



(a) Mean Absolute Error **(b) Mean Squared Error**

Figure 6: Value Network Training and Validation Error

over the total amount of time the ground stations are within range of the satellite. In some instances, no solution is 100% optimal in regards to this metric. This is due to the fact that the simulation can start immediately over a ground station, before the agent has time to collect data. While this results in some solutions that at best achieve 50-80% downlink utilization, it provides MCTS a challenging problem in which data must be immediately collected and immediately downlinked to maximize the reward.



(a) Downlink Utilization **(b) Reward**

Figure 7: MCTS Performance - First 10 Initial Conditions

Due to the tradeoff between reward achieved, Q-estimates generated, and execution time, 50 simulations is selected as the number of simulations to use to train the value network. 50 simulations is adequate to generate a quality solution and achieve the maximum amount of reward possible in many cases. However, in some cases, 50 simulations does not achieve the maximum amount of reward shown to be possible. In these cases, 10 or 25 simulations results in more reward. In fact, on average, 10 and 25 simulations results in more reward than 50 simulations. In Figure 7b, 50 simulations achieves about 5 less reward on average. In Figure 7a, 50 simulations results in a 5% reduction in ground station utilization. This is likely due to the greedy downlink within the random rollout policy. It is possible that fewer simulations can result in a better solution if the state action

value function computed in MCTS is higher for the optimal state-action pairs. In this case, fewer simulations that more quickly find the reward will result in a higher state-action value estimate because exploration is reduced and sub-optimal reward is not used to update the state-action value function. While maximizing reward is an important part of this research, so is keeping the spacecraft within its resource constraints. Therefore, high-quality Q-function estimates are weighted more heavily than maximized reward, especially when only a 5% difference in question. More data generated over more simulations per step is deemed more important than a small increase in reward for value network performance.

In Figure 8, the execution time for MCTS is displayed. On average, it takes nearly three hours for a single run of MCTS to complete due to the sheer number of spacecraft simulations that must be executed. Furthermore, Basilisk requires that simulations are reset to the beginning and stepped through to the current state once again after a single simulation of MCTS within a step forward in the environment. The execution time of MCTS could be greatly reduced in future work if functionality is built into Basilisk to save off copies of a simulation and restart them at a later time. However, even with this modification, MCTS itself is not suited for on-board scheduling and should be reserved as a ground-based planning tool.

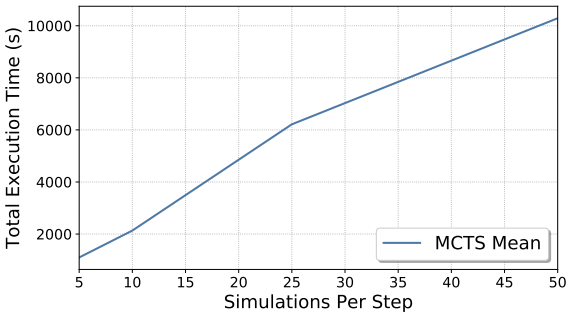
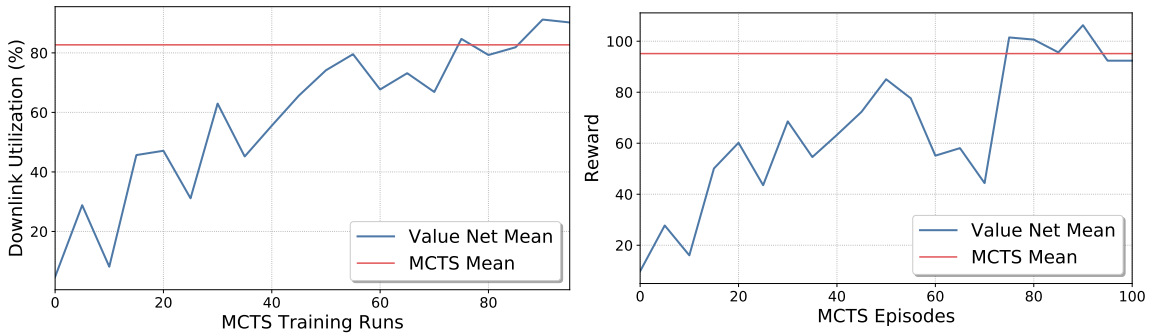


Figure 8: Execution Time

Value Network Deployment

The state-action value network is incrementally trained on five sets of Q-estimates at a time, which are generated by individual runs of MCTS on random initial conditions. After each training cycle, the value network is executed on the first 10 sets of test conditions and the performance is compared to the average of 50-simulation MCTS. In Figure 9, the downlink utilization and reward are plotted as the value network completes episodes. The performance of the value network eventually matches MCTS in both reward and downlink utilization at around 75 simulations. It is perplexing that the value network can perform better than the algorithm used to generate data to train it. However, this is possible due to the wide-array of experience the value network is trained on and demonstrates the ability of value function approximators to learn from experience.

While the state-action value function is able to match the performance of 50-simulation MCTS in terms of reward and downlink utilization, the policy it has learned does not manage the states of the spacecraft well, and most simulations fail to complete the entire planning horizon. This can be seen in Figure 10b. In 50-simulation MCTS, no simulations fail to complete the planning horizon. In execution of the value network, around 80% of simulations fail to complete. The authors hypothesize that the policy the value network has learned ignores the battery charge and



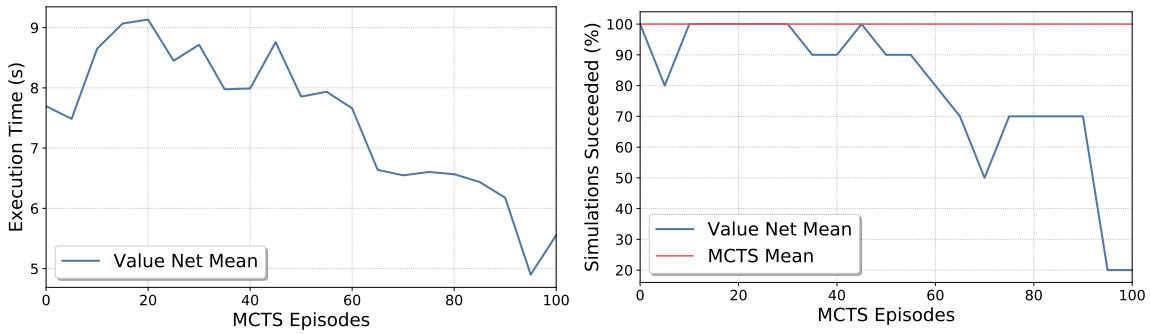
(a) Value Net Downlink Utilization

(b) Value Net Reward

Figure 9: Value Network Performance - First 10 Initial Conditions

space available in the data buffer. The agent attempts to collect as much data as it can throughout the duration of the simulation and downlink when a ground station is in view. This is likely due to the fact that the reward bonus for finishing a simulation is too small compared to the reward received from downlinking data, causing the value of managing states to successfully complete a simulation to be lost in the error of the outputs from the value network.

One benefit to state-action value network execution is that the execution time is several orders of magnitude lower than MCTS. 50-simulation MCTS takes around three hours to compute a solution for a single planning horizon. As shown in Figure 10a, the value network takes seconds to execute on a single planning horizon. If the rate of simulation failure for the value network can be corrected, it may be a viable technique for on-board execution given enough computational power.

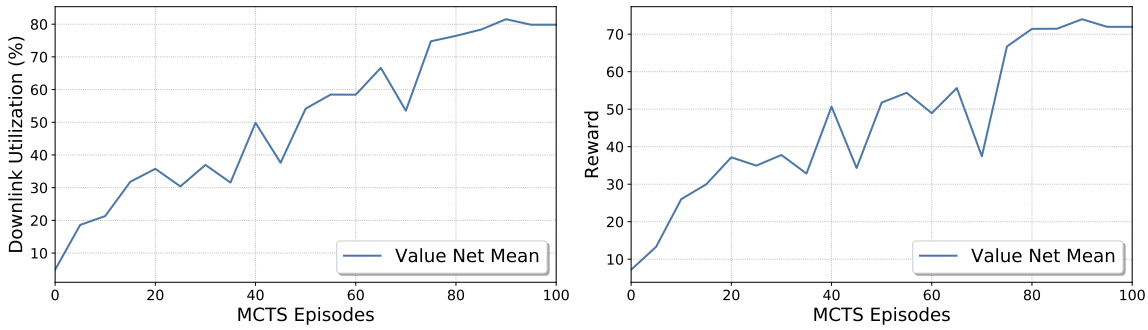


(a) Execution Time

(b) Simulations Successfully Completed

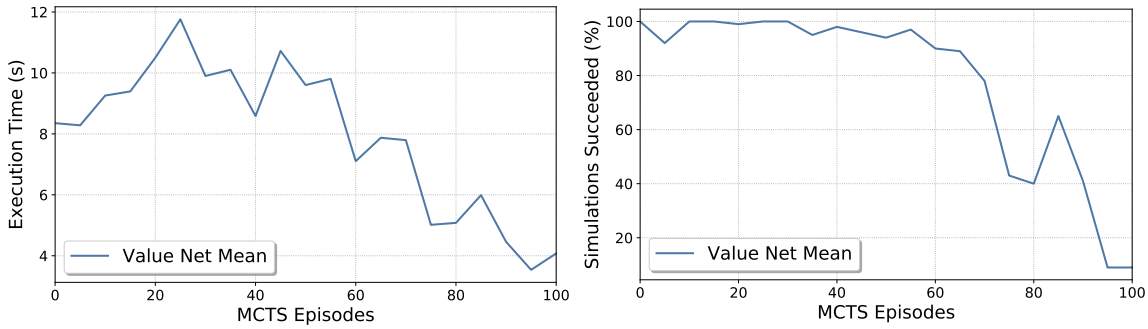
Figure 10: Value Network Execution Time - First 10 Initial Conditions

The value network is also executed on a full set of 100 initial conditions for a more accurate representation of performance. Similarly to the smaller test set of 10 initial conditions, the larger test set shows the value network execution results in a policy that attempts to collect and downlink data, ignoring resource constraints and causing failures. As shown in Figure 11, the value network performance plateaus at around 80 episodes. Unlike the first 10 sets of initial conditions, the full set includes planning horizons with no downlink opportunities, so reward will be lower on average. As shown in Figure 12b, the percent of simulations that complete the planning horizon plunge as the reward increases.



(a) Downlink Utilization **(b) Reward**

Figure 11: Value Network Performance - 100 Initial Conditions



(a) Execution Time **(b) Total Simulation Time**

Figure 12: Value Network Execution Time - 100 Initial Conditions

CONCLUSION

This work successfully demonstrates the use of Monte Carlo tree search (MCTS) as a ground-based spacecraft scheduling technique that considers non-linear dynamics, resource constraints, and multiple spacecraft modes. MCTS is shown to compute quality solutions to the EOS scheduling problem with respect to downlink when possible. MCTS is also shown to generate solutions that complete the planning horizon without violating spacecraft resource constraints. Future work should explore how MCTS performs in comparison to traditional optimization-based approaches such as simulated annealing or job shop scheduling.

This work also demonstrates that state-action value estimates generated by Monte Carlo tree search can be used to train a state-action value network, which may be rapidly executed on the Earth-observing spacecraft scheduling problem with some success. The state-action value network is able to match the performance of MCTS in regards to downlink utilization and total reward, but fails to consider resource constraints and fails planning horizons prematurely. Significant work remains before state-action value functions can be executed onboard spacecraft for planning purposes. In future work, the state-action value function should be used as the rollout policy within MCTS to continually improve upon the state-action value estimates as Silver et al. did for AlphaGo.[11]

Future work should also explore techniques to guarantee that agents successfully finish a planning interval. Applying a shield during training or execution of the state-action value network will likely

result in better performance, as shown by Harris et. al.[10] Better reward functions should also be applied to return a larger penalty for failure or a larger bonus for success, and non-discounted reward should be explored due to the finite-horizon nature of the problem. Furthermore, future work should aim to reduce the assumptions made within this paper. Namely, a fuel budget for the desaturation mode should be developed and the omni-directionality assumption of the antenna should also be removed. A minimum attitude error should also be achieved by the spacecraft before collecting observation data.

ACKNOWLEDGMENT

The author would like to thank the AVS Laboratory for developing the Basilisk Astrodynamics Software Framework that was used to create the simulations. The author would also like to thank Andrew Harris, who put a significant amount of work into developing a `Gym` Environment based on Basilisk that the author could modify and use for this version of the Earth-observing satellite scheduling problem.

REFERENCES

[1] S. A. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye, "Improving the Operations of the Earth Observing One Mission via Automated Mission Planning," 2010.

[2] 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.

[3] A. Globus, J. Crawford, J. Lohn, R. Morris, and D. Clancy, "Scheduling Earth Observing Fleets Using Evolutionary Algorithms: Problem Description and Approach," International Conference on Space Mission Challenges for Information Technology, 2003.

[4] Li Jian and Wang Cheng, "Resource planning and scheduling of payload for satellite with genetic particles swarm optimization," 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), 2008, pp. 199–203.

[5] 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.

[6] S. Spangelo and J. Cutler, "Optimization of Single-Satellite Operational Schedules Towards Enhanced Communication Capacity," AIAA Guidance, Navigation, and Control Conference, 2012, 10.2514/6.2012-4610.

[7] 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.

[8] F. Sadeghi and S. Levine, "CAD2RL Real Single-Image Flight Without a Single Real Image," *arXiv preprint arXiv:1611.04201*, 2016.

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

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

[11] 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.

[12] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. 2015.

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

[14] H. Yu, X. Li, R. M. Murray, S. Ramesh, and C. J. Tomlin, *Safe, Autonomous and Intelligent Vehicles*. Springer Publishing Company, Incorporated, 1st ed., 2018.

[15] 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.

APPENDIX: ALGORITHMS

Algorithm 1 MCTS Execution

Initialize env, $s = s_0$, $r_{sum} = 0$
$t_i = 0$, $t_{\text{max}} = 90$ min, NumSteps $= 30$
while $t_i \leq t_{\text{max}}$
    $d =$ NumSteps $- i$
    $a =$ SelectAction(s, d)
    $(s', r, ep\_over) =$ env.step(a)
    r_{sum} += r
    if ep_over
        break
    $t_{i+1} = t_i + \Delta t$
    i += 1

Algorithm 2 Monte Carlo Tree Search

function SelectAction(s, d)
    for Number of Simulations
        Simulate(s, d)
    return $\arg\max_a Q(s, a)$

function Simulate(s, d)
    if $d = 0$
        return 0
    if $s \notin T$
        for $a \in A$
            $N(s, a) \leftarrow 1$
            $Q(s, a) \leftarrow 0$
        $T \cup s$
        return Rollout(s, d)
    $a \leftarrow \arg\max_a Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s, a) + 1}}$
    $(s', r, ep\_over) \sim G(s, a)$
    if ep_over
        $q \leftarrow r$
    else
        $q \leftarrow r + \gamma$Simulate($s', d - 1$)
    $N(s, a)$ += 1
    $Q(s, a)$ += $\frac{q - Q(s, a)}{N(s, a)}$
    return q

function Rollout(s, d)
    if $d = 0$
        return 0
    if any(q_j) and b > 0
        $a =$ Downlink
    else
        $a =$ rand({Observation, Downlink, Charge, Desaturation})
    $(s', r, ep\_over) \sim G(s, a)$
    if ep_over
        return r
    else
        return $r + \gamma$Rollout($s', d - 1$)