(Preprint) AAS 23-116

A COMPARISON OF DEEP REINFORCEMENT LEARNING ALGORITHMS FOR EARTH-OBSERVING SATELLITE SCHEDULING

Adam Herrmann* and Hanspeter Schaub†

Deep reinforcement learning (DRL) has shown promise for on-board planning and scheduling, particularly for Earth-observing satellites (EOS), due to the ability of trained policies to achieve optimal planning and scheduling performance at very fast execution times. However, the question of which DRL algorithms are best suited for EOS scheduling problems has not been comprehensively explored. This work compares value-based and policy-based reinforcement learning algorithms for EOS scheduling. A reference problem in which a satellite attempts to maximize the amount of science data collected from two separate instruments while managing resource constraints such as power and reaction wheel speeds is constructed. Value-based and policy gradient DRL algorithms are applied to the problem and compared on the basis of performance, training time, and model complexity. Proximal policy optimization (PPO), a policy-based method, and MCTS-Train, a value-based method, are applied to the problem for various combinations of hyperparameters. PPO is shown to produce the highest performing policies, achieving 0.750 average reward out of a maximum of 1 reward for networks with $8 \cdot 10^4$ trainable parameters. PPO also produces the smallest models that are capable of achieving 0.696 average reward with only 184 trainable parameters. MCTS-Train produces at most 0.614 reward, with the best performing model using $8 \cdot 10^4$ trainable parameters. Both algorithms require the same order of magnitude of training time for large hyperparameter searches, but PPO is faster if only a single network is required.

INTRODUCTION

Spacecraft planning and scheduling is the process by which the set of tasks and the corresponding order in which they are executed for a spacecraft to achieve its mission objectives are computed. Historically, planning and scheduling is a ground-based process in which a planning software computes a spacecraft plan that is uplinked to the spacecraft for open-loop execution. Many different techniques may be used to generate the plan, including mixed-integer programming and metaheuristic optimization.[1–4] However, a ground-based planning and open-loop execution paradigm is brittle to opportunistic science events or tasks taking either longer or shorter than expected. The CASPER[5] system at NASA's Jet Propulsion Laboratory addresses this issue by using iterative repair to correct apriori spacecraft plans as necessary. However, reinforcement learning, which may execute a policy in a closed-loop fashion on-board spacecraft after training, has recently emerged as another potential

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

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

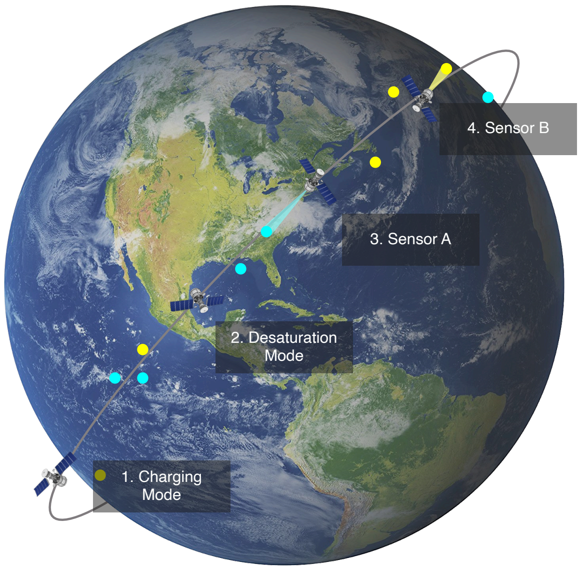


Figure 1: Multi-Sensor Earth-Observing Satellite Scheduling Problem.

technique for on-board planning and scheduling due to its optimality potential and fast execution time.

Past work has demonstrated the ability of value-based[6,7] and policy gradient[8,9] reinforcement learning methods to solve the Earth-observing satellite (EOS) scheduling problem. However, little work has compared these two categories of deep reinforcement learning algorithms for the purposes of solving EOS scheduling problems. The objective in a reinforcement learning problem is to learn a policy that maps states to actions, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, to maximize a numerical reward signal. Value-based reinforcement learning algorithms do this implicitly by solving for the value or action-value function and then compute the policy afterwards. However, small errors in the computation of the value function may lead to large errors in the derived policy. Furthermore, the model required to approximate the value function may be larger than the model required to solve only for the optimal policy. Policy-based methods solve for the policy directly to avoid these issues, but they have a tendency to get stuck in local optima because they rely on gradient-ascent to compute the policy. Harris and Schaub apply policy gradient methods, specifically proximal policy optimization (PPO), to an EOS scheduling problem. To improve convergence, the authors use shielded deep reinforcement learning.[8] In shielded deep reinforcement learning, a low-fidelity safety MDP is constructed using the states most relevant to resource constraint violations. A shield policy is derived that ensures the low-fidelity safety MDP will never enter an off-nominal mode. The shield is then utilized in training and deployment, ensuring that the agent does not take unsafe actions. Reference 8 utilizes the shield within PPO, demonstrating that SDRL can improve the speed of convergence of policy-gradient methods for EOS scheduling. Reference 6 utilizes the shield policy as a heuristic policy in Monte Carlo tree search, using the state-action value estimates generated by MCTS to train a neural network approximation of the value function. This training architecture is referred to as MCTS-Train. The heuristic policy demonstrates a large performance improvement over a random rollout policy. However, these results are not compared to the DRL approaches (shielded and otherwise) taken by Reference 8. This work compares PPO (unshielded) and heuristic-rollout MCTS-Train.

First, the EOS scheduling problem is formulated where the objective is to maximize the amount of science data collected using two sensors while avoiding resource constraint violations. Resources modeled on-board the spacecraft include power and reaction wheel speeds. This problem is depicted in Figure 1. The algorithms applied to solve the algorithm are then described. Value-based (MCTS-Train) and policy gradient (PPO) methods are applied to solve a Markov decision process formulation of the problem. For each algorithm, an extensive hyperparameter search is presented. Each of these algorithms are then compared on the basis of performance and the model complexity required to achieve optimal performance.

REFERENCE PROBLEM

Overview

In the reference problem constructed for this comparison, a spacecraft in low-Earth orbit attempts to maximize the sum of images collected with the correct sensor while managing the battery charge level and reaction wheel speeds. A planning horizon of 270 minutes is split into 90 decision-making intervals, called planning intervals. At each planning interval, the spacecraft enters into one of four possible modes. The modes include: a.) spacecraft charging, b.) reaction wheel desaturation, c.) imaging with sensor A, and d.) imaging with sensor B. This problem is depicted in Figure 1. Several different subsystems are modeled in the problem.

Simulation Architecture

The reference problem is simulated using the Basilisk[*] astrodynamics software architecture. Basilisk provides a Python interface for scripting but implements the simulation code in C/C++ for speed.[10] A system diagram of the simulation is provided in Figure 2. There are seven separate tasks - a dynamics task, an environment task, and five separate flight software-related tasks. Each task contains a collection of modules most relevant to the task description. Each colored block represents a separate module, which have certain inputs and outputs to and from other modules that are not depicted in the system diagram for simplicity.

The Basilisk simulation models several subsystems which are relevant to the planning and scheduling problem. A full attitude control system is implemented to simulate a representative spacecraft mission where many systems are coupled to the attitude dynamics. Various location-pointing reference frames are switched between based on the specific mode and passed to an attitude error computation module, which then passes the attitude error to an MRP feedback control law. The location pointing frames include a sun-pointing reference for battery charging, an anti sun-pointing reference for desaturation (to ensure a power penalty when desaturating), and a nadir-pointing reference for imaging. The MRP feedback control law sends torque commands to the three reaction wheels, which change the dynamics of the spacecraft. The reaction wheels are modeled after the Honeywell HR16 reaction wheels. A momentum dumping module is also implemented, which maps reaction wheel momentum to thrust commands to remove momentum from the reaction wheels. The thrusters are modeled after the Moog Monarc-1 thrusters. Random external torque is implemented to build up momentum in the reaction wheels over time.

A power system is also simulated, leveraging Basilisk's high-fidelity dynamics capabilities to accurately compute power consumption and generation. Simulated solar panels generate power based on incidence angle, panel area, and efficiency. The effects of eclipse are also considered.

[*]`http://hanspeterschaub.info/basilisk`

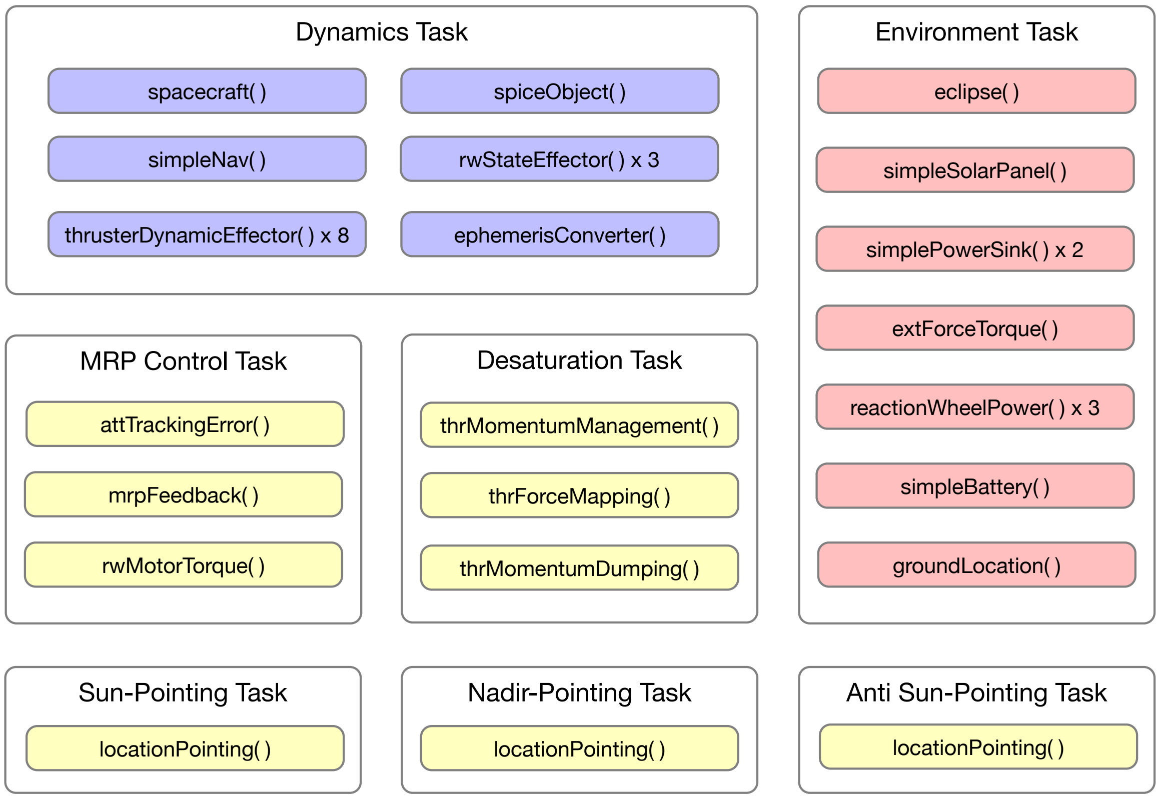


Figure 2: Basilisk Simulation Framework Illustration.

Generated power is stored in a modeled battery, and the imagers consume power from the battery. Static power draw is also included.

Markov Decision Process

The multi-sensor Earth-observing satellite scheduling problem is formulated as a Markov decision process (MDP), which is defined by the 5-tuple $(\mathcal{S},\ \mathcal{A},\ T,\ R,\ \gamma)$. An MDP is a sequential decision-making problem in which an agent observes the current state s_i and selects an action a_i following a policy $\pi : \mathcal{S} \times \mathcal{A}$. The policy maps states to actions. The agent takes the action a_i and observes a new state s_{i+1} and receives a reward r_i based on the reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. MDPs follow the Markov assumption, meaning the next state is conditionally dependent only on the current state and action: $T(s_{i+1}|s_i, a_i) = T(s_{i+1}|s_i, a_i, s_{i-1}, a_{i-1}, ..., s_0, a_0)$.

State Space All relevant state information for the purposes of maintaining the Markov assumption must be included in the state space, $\mathcal{S}$. The state space for this problem was originally developed by Nazmy et al.,[11] but has been modified slightly and is summarized below:

- Spacecraft relative position in the SEZ frame, ${}^{\text{SEZ}}\mathbf{r}_{\text{s/c}}$
- Spacecraft relative velocity in the SEZ frame, ${}^{\text{SEZ}}\mathbf{v}_{\text{s/c}}$
- L^2 norm of MRP attitude error, ϵ_{att}
- L^2 norm of attitude rate, $||{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}||$

- L^2 norm of reaction wheel speeds normalized by maximum wheel speed, $||\mathbf{\Omega}||/\Omega_{\text{max}}$
- Stored battery charge normalized by maximum charge, z/z_{max}
- Target access indicator
- Target sensor type
- Eclipse indicator

The relative position and velocity of the spacecraft in the SEZ frame, as well as the target access indicator and target sensor type states, are included in the state space to aid the decision-making agent in completing the science objectives. The SEZ position and velocity are transformed into canonical coordinates using Equations (1) and (2). The position is normalized by the radius of the Earth at the equator, and the velocity is normalized by the velocity of a circular orbit at the equator of the Earth.

$$\mathbf{r}_{\text{can}} = \frac{\mathbf{r}}{R_{\text{eq}}} \tag{1}$$

$$\mathbf{v}_{\text{can}} = \frac{\mathbf{v}}{\sqrt{\mu/R_{\text{eq}}}} \tag{2}$$

The target access indicator indicates whether or not the spacecraft has access to the target location (0 or 1). The target sensor type denotes the desired sensor type of the upcoming target (0 for target A, 1 for target B). The attitude states provide information on the attitude control system. The battery charge, eclipse indicator, and L^2 norm of the reaction wheel speeds provides state information for the purposes of resource management.

Action Space A discrete action space is constructed using mode-based approach where each mode represents a high-level spacecraft behavior. The low-level behavior of each mode is dictated by the attitude reference and on/off states of each spacecraft subsystem. Each mode is entered for a total of three minutes, which is primarily constrained by the rate at which the attitude control system can converge to the attitude reference. The spacecraft can enter into four separate modes: a.) charging, b.) desaturation, c.) image with sensor A, and d.) image with sensor B.

In the charging mode, the spacecraft turns off the instruments and points its solar panels at the sun. In the desaturation mode, the spacecraft points the solar panels away from the sun and the spacecraft thrusters are used to remove momentum from the reaction wheels. The panels point away from the sun in order to construct a scenario in which there is a tradeoff between the spacecraft charging and desaturation modes in terms of power. In the imaging modes, the spacecraft points its instruments in the nadir direction and takes an image once the spacecraft is within the elevation and range requirements of the target.

Transition Function It is difficult to construct an explicit transition function using conditional probabilities that accurately captures the state transitions in the multi-sensor Earth-observing satellite scheduling problem. Therefore, the transition function is represented with a generative model $G(s_i, a_i)$ given in Equation (3). A generative model returns a new state s_{i+1} and reward r_i by integrating equations of motion, sampling a probability distribution, or some combination of both. In this problem, the generative transition function is the previously described Basilisk simulation.

$$s_{i+1},\ r_i = G(s_i, a_i) \tag{3}$$

The Basilisk simulation is wrapped within a Gym* environment, which is a standard interface for decision-making agents. The Gym interface turns the Basilisk modes on or off, runs the simulation, constructs the observations, computes the reward, and returns the information back to the decision-making agent. The Gym environment used in this work, called the leoObservingMultiSensor_env(), may be found on the develop branch of the Basilisk Gym Interface† library.

Reward Function The reward function was first described by Nazmy et al.[11] and will be summarized here. A piecewise reward function is formulated such that the maximum undiscounted cumulative reward is 1 and the minimum is -1. The reward function is provided in Equations (4) and (5).

$$R(s_i, a_i, s_{i+1}) = \begin{cases} -1 & \text{if failure} \\ \dfrac{f}{N} \cdot \dfrac{1}{1+\epsilon_{\text{att}}^2} & \text{if } el_{\text{s/c}} > el_{\text{min}} \text{ and } a_i \in \{\text{Image A},\ \text{Image B}\} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$f = \begin{cases} 1 & \text{if } a_i == a_{\text{preferred}} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The first condition checked for is failure. The failure condition is true if the battery is drained to zero charge or any of the reaction wheel speeds exceed the maximum speed. If failure does not occur and the spacecraft enters the imaging mode corresponding to the correct sensor type, a reward of 1.0 is returned. This reward is scaled by $1/N$, the maximum number of imaging targets, and $1/(1+\epsilon_{\text{att}}^2)$. If the spacecraft takes every possible image with zero attitude error and never fails to manage its resources, the cumulative reward will be 1.

METHODS

MCTS-Train

MCTS-Train is a reinforcement learning pipeline inspired by AlphaZero[12] that utilizes Monte Carlo tree search[13] to generate estimates of the state-action value function and supervised learning to regress over these estimates. References 6 and 14 show that MCTS-Train produces near-optimal policies for Earth-observing satellite scheduling problems. A diagram of MCTS-Train is shown in Figure 3, and the associated algorithm is presented in Algorithm 1.

In the first step of MCTS-Train, Monte Carlo tree search is used to generate estimates of the state-action value function, $\hat{Q}(s, a)$, following the policy found by MCTS. Monte Carlo tree search works by simulating hundreds or thousands of interactions with the environment, building a search tree based on the experience in the environment. At each step through the environment, MCTS runs a number of simulations-per-step to determine what the next best action to take is. This is done through a combination of selection steps that exploit current knowledge of the state-action value function and rollout steps that execute a random or hand-crafted policy if MCTS reaches a state is has not encountered yet. For the purposes of data generation, MCTS solves the MDP for thousands of initial conditions, generating on the order of 10^4–10^5 data points to form the training

*https://www.gymlibrary.dev/

†https://bitbucket.org/avslab/basilisk-gym-interface/src/develop/basilisk_env/envs/

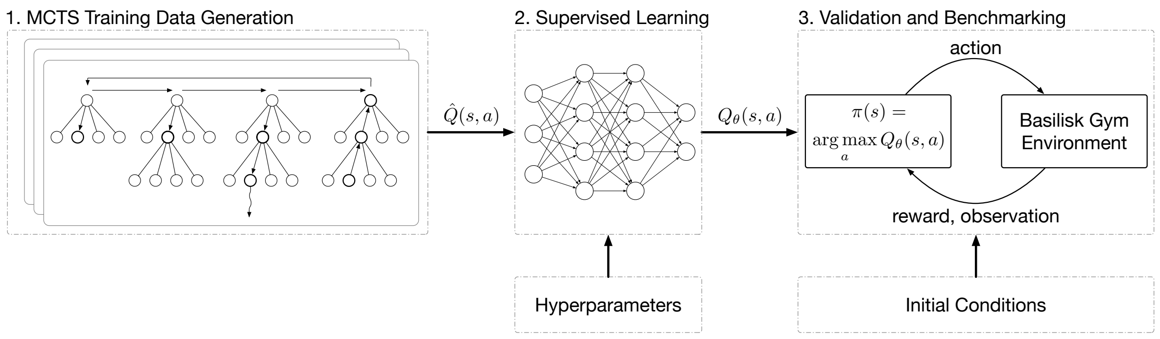


Figure 3: MCTS-Train Architecture.

data set $\mathbf{Q}$. After the data generation step, supervised learning is applied over the dataset to produce a neural network approximation of the state-action value function, $Q_\theta(s,a)$. Optionally, many different neural network hyperparameters can be input into the training process for experimentation purposes. These hyperparameters include the size of the networks, the probability of dropout, the activation function, and parameters specific to the activation function. Typically, mean squared error is used for the loss function,

$$L(\theta) = \sum_{s_i \in \mathbf{Q}} \Big(Q_\theta(s_i, a_i) - \hat{Q}(s_i, a_i)\Big)^2, \tag{6}$$

and the Adam optimizer is selected as the optimization algorithm to update the weights of the network(s). After training, the neural networks are validated in the environment using the following policy:

$$\pi(s) = \arg\max_a Q_\theta(s,a) \tag{7}$$

Proximal Policy Optimization

Proximal policy optimization (PPO) is a popular policy gradient method based on Trust Region Policy Optimization (TRPO) that avoids the use of TRPO's hard constraint that ensures the size of the policy update is not too large.[15] PPO does this by reformulating the loss function to penalize steps too far from the current policy using a clipping function. This is provided in Equation (8), where $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and $\hat{A}_t$ is an estimate of the advantage at time t. The ϵ parameter is usually set around 0.2.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta),\ 1-\epsilon,\ 1+\epsilon)\hat{A}_t)\Big] \tag{8}$$

The stable-baselines3* (SB3) implementation of proximal policy optimization with multiprocessing is utilized for this work. A custom policy is implemented so that additional hyperparameters, such as dropout and parameters specific to the activation functions, can be modified, which is not supported by SB3's vanilla PPO implementation. The parameters of the neural network are shared between the policy and value function and are updated using Equation (9), where c_1 and c_2 are coefficients that control the contribution of the value function loss and entropy bonus, S.

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\Big[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\Big] \tag{9}$$

*https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

Algorithm 1 MCTS-Train Algorithm.

```
    1.) generate training data
1:  initialize set of training data, $\mathbf{Q}$ = {}
2:  for k = 1:N
3:      initialize env, MCTS
4:      for i = 1:MAX STEPS
5:          $a_i$ = MCTS.selectAction($s_i$)
6:          $s_{i+1}$, $r_i$, info, done = env.step($a_i$)
7:      $\mathbf{Q} \cup \{$MCTS.$\hat{Q}(s_i, a_i) \; \forall \; s_i, \; a_i \in$ MCTS$\}$
8:      MCTS.clear()

    2.) train networks
9:  initialize hyperparameters
10: initialize set of trained networks, $\mathbf{Q}_\theta = \{\}$
11: for hp $\in$ hyperparameters
12:     initialize $Q_\theta$ with hyperparameters
13:     $Q_\theta$.train($N$ epochs)
14:     $\mathbf{Q}_\theta \cup \{Q_\theta\}$

    3.) validate trained networks
15: initialize env
16: initialize performance metrics
17: for $Q_\theta \in \mathbf{Q}_\theta$
18:     reward_sum $= 0$
19:     $s_i$ = env.reset()
20:     for i = 1: MAX STEPS
21:         $Q(s_i, a_i)$ = $Q_\theta$.predict($s_i$)
22:         $a_i = \arg\max_{a_i} Q(s_i, a_i)$
23:         $s_i$, $r_i$, info, done = env.step($a_i$)
24:         reward_sum $+= r_i$
25:     update performance metrics with reward_sum, env.metrics
```

During each iteration of the policy, N actors collect experience in the environment. The loss function is computed using the collected data and the policy is optimized using the Adam optimizer for K epochs. The full algorithm is provided in the original paper by Schulman et. al. in 15.

RESULTS

MCTS-Train

The Monte Carlo tree search algorithm is central to the MCTS-Train pipeline. Before MCTS can be utilized within the MCTS-Train pipeline, it must be correctly parameterized. Specifically, the

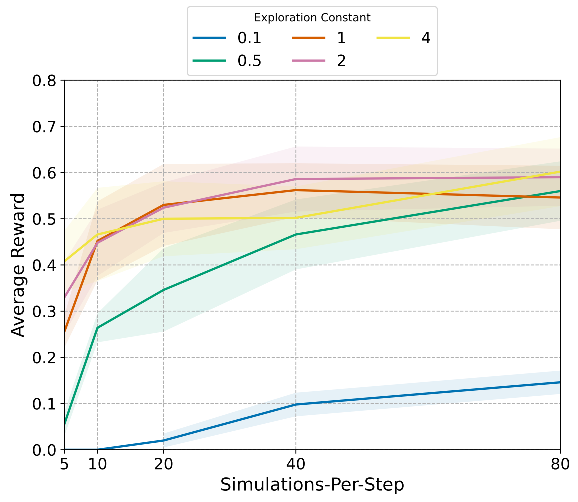
Algorithm 2 Proximal Policy Optimization Algorithm.

```
1: initialize policy $\pi_{\theta_{\text{old}}}$
2: initialize $N$ actors
3: for iteration 1:MAX ITERATIONS
4:     for actor i = 1:N
5:         run $\pi_{\theta_{\text{old}}}$ in environment for MAX STEPS
6:         compute advantage estimates $\hat{A}_1 \cdots \hat{A}_{\text{MAX STEPS}}$
7:     optimize $L(\theta)$ wrt $\theta$, with $K$ epochs and batch size $M \leq N(\text{MAX STEPS})$
8:     $\theta_{\text{old}} \leftarrow \theta$
```

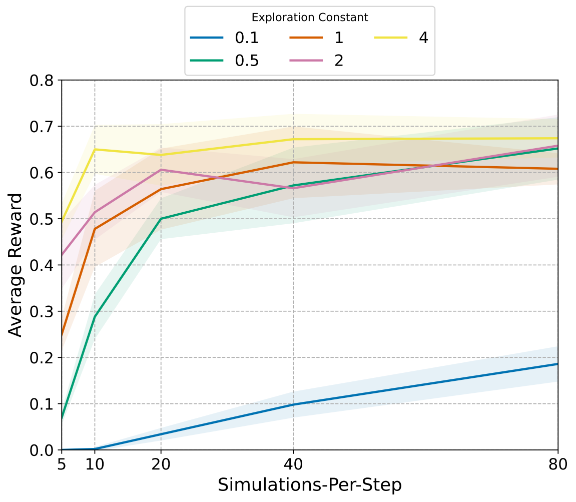
exploration constant and the number of simulations-per-step through the environment must be set. If the exploration constant is too low, MCTS will not explore the search space and default to the first action available. If the exploration constant is too large, MCTS will equalize the number of times each action is taken and inadequately exploit the knowledge it does have about which actions are good and which are not. In the case of the number of simulations-per-step, too few simulations-per-step will result in very shallow trees. Too many simulations-per-step will result in unnecessary computation time.

Before generating the training data, an experiment is first performed to determine which exploration constant and number of simulations-per-step are required to generate policies with good performance at as few simulations-per-step as possible. Exploration constants of $\{0.1, 0.5, 1, 2, 4\}$ and simulations-per-step of $\{5, 10, 20, 40, 80\}$ are utilized for the experiment. Furthermore, two separate rollout policies are explored. First, a more conservative rollout policy that takes safety actions at normalized reaction wheel speeds of $||\mathbf{\Omega}||/\Omega_{\text{max}} = 0.5$ and normalized battery charge of $z/z_{\text{max}} = 0.5$ is utilized. Then, a more aggressive rollout policy that takes safe actions at $||\mathbf{\Omega}||/\Omega_{\text{max}} = 0.8$ and $z/z_{\text{max}} = 0.2$ is used. The results of this experiment are provided in Figure 4. For the conservative rollout policy, performance plateaus just below 0.6 reward for exploration constants of $\{1, 2\}$ and simulations-per-step of $\{40, 80\}$. The more aggressive rollout policy plateaus somewhere between 0.6-0.7, and all hyperparameter combinations either perform the same or better than the conservative rollout policy.

Given the results of the MCTS hyperparameter experiment, the aggressive rollout policy with an exploration constant of 4 and 20 simulations-per-step is selected for data generation. MCTS is used to solve the planning problem for 1,000 unique initial conditions. Using 30 processes in parallel, the training data took 40 hours to generate using a 3.8 GHz AMD 3960x Threadripper CPU with 64 GB of RAM. After data generation, a hyperparameter search is performed over various neural network hyperparameters. After these networks are trained, they are benchmarked in the environment using random initial conditions. For the first network hyperparameter search, $\{1, 2, 4\}$ hidden layers and $\{10, 20, 40, 80, 160\}$ nodes per hidden layer are utilized. The results of this experiment are provided in Table 1. The networks trained in under an hour using an Nvidia 3070 graphics card. In general, networks between 80-160 nodes wide with 2-4 hidden layers are required to produce good performance. Furthermore, the addition of a small amount of dropout appears to help with overfitting, resulting in better performance for the larger networks. However, these networks are rather large, and most networks achieve very poor performance. Several other experiments, not shown here, are performed over the batch size and number of training epochs. However, these did not result in any appreciable increase in performance. Future work will further investigate smaller



(a) Conservative Rollout Policy.



(b) Aggressive Rollout Policy.

Figure 4: MCTS Hyperparameter Search Over Exploration Constant and Simulations-Per-Step.

Table 1: MCTS-Train Network Size Search. Batch Size = 4.5e4. Epochs = 5e3

(a) $p(\text{dropout}) = 0.0$

	Nodes				
Hidden Layers	10	20	40	80	160
1	-0.393	-0.485	-0.495	-0.219	0.180
2	-0.456	-0.096	-0.249	0.229	0.564
4	-0.402	-0.341	-0.123	-0.018	0.614

(b) $p(\text{dropout}) = 0.05$

	Nodes				
Hidden Layers	10	20	40	80	160
1	-0.414	-0.384	-0.183	0.204	0.471
2	-0.422	-0.483	-0.353	0.641	0.558
4	-0.428	-0.470	-0.204	0.516	0.552

networks, as well as the impact of the rollout policy in MCTS on the final trained policy.

Proximal Policy Optimization

Several hyperparameter searches are conducted for proximal policy optimization to ensure the algorithm is correctly parameterized for the multi-sensor EOS scheduling environment. In the first hyperparameter search, a search over the number of hidden layers and the number of nodes in each hidden layers is conducted. Two separate dropout rates are considered, $p(\text{dropout}) = \{0.0, 0.05\}$. The batch size is set to $2 \cdot \text{NUM CORES} \cdot \text{MAX STEPS} = 2 \cdot 46 \cdot 90 = 8280$, where NUM CORES is the number of actors working in parallel and MAX STEPS is the maximum number of steps in the environment. Only 5 epochs are selected for each update. This combination of batch size and number of epochs is selected to keep training stable. After training, the performance of each policy is benchmarked on a set of $3 \cdot 46$ random initial conditions. The results of this hyperparameter search are presented in Table 2. Networks that achieve less than 0.3 reward are highlighted in red, networks that achieve at least 0.3 but less than 0.6 reward are highlighted in yellow, and networks that achieve at least 0.6 reward are highlighted in green. Based on this search, one would assume that larger networks are required to converge to high-performing policies as only policies with 80 - 160 nodes and 2 - 4 hidden layers achieve more than 0.6 reward.

Based on the results of the previous hyperparameter search, one would assume that between $8 \cdot 10^3$

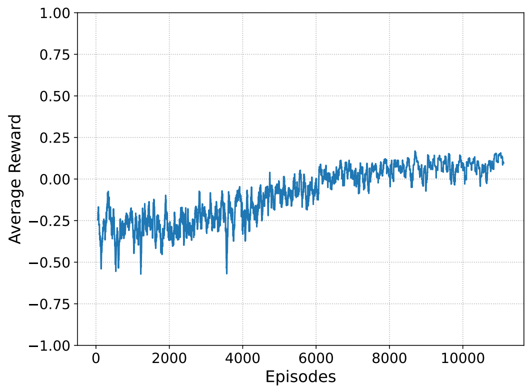
Table 2: PPO Network Size Search. Batch Size = 8280. Epochs = 5

(a) $p(\text{dropout}) = 0.0$

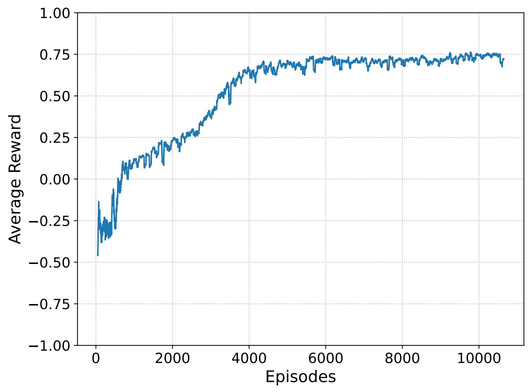
Hidden Layers	Nodes 10	20	40	80	160
1	0.111	0.174	0.305	0.441	0.505
2	0.112	0.163	0.496	0.663	0.690
4	0.134	0.328	0.711	0.722	0.750

(b) $p(\text{dropout}) = 0.05$

Hidden Layers	Nodes 10	20	40	80	160
1	0.113	0.156	0.285	0.389	0.507
2	0.095	0.251	0.380	0.684	0.724
4	0.100	0.178	0.328	0.736	0.717



(a) 1 Hidden Layer. 10 Nodes Wide.



(b) 4 Hidden Layers. 160 Nodes Wide.

Figure 5: Example Reward Curves for Batch Size = 8180. Epochs = 5. No Dropout.

– $8 \cdot 10^4$ trainable parameters are required to produce good policies. However, before dismissing smaller networks as not having enough capacity to learn good policies, more searches should be conducted to determine if smaller networks can be tuned to produce good policies. It is desirable to have smaller policies because the proposed use case is on-board planning and scheduling where memory is limited. Larger networks take up more memory and require more execution time. Another hyperparameter search is conducted over the batch size and number of epochs per batch of data to determine if these parameters can be tuned further. The number of hidden layers is fixed at 4, and the number of nodes are fixed at 20, resulting in a policy with around $1.6 \cdot 10^3$ trainable parameters. The set of training epochs considered is $\{5, 25, 50\}$, and the set of batch sizes considered is $\{2070, 4140, 8280\}$. The results of this search are presented in Table 3 for three different dropout rates, $\{0.0, 0.05, 0.1\}$. First and foremost, it is apparent that a very small amount of dropout or no dropout is preferable. Second, performance is typically best for large numbers of epochs (>5). Batch sizes of 4140 and 8280 perform poorly for ≤ 5 epochs. If larger batch sizes are used, then large numbers of training epochs are required. With either a small batch size or a large batch size and number of training epochs, the small policy now performs about as well as the larger policies.

To ensure that the conclusions drawn about a large number of epochs and a small batch size holds for all networks, the first experiment is repeated for a batch size of 2070 and 50 epochs. These results are presented in Table 4. Again, it is shown that no dropout is preferable. The smallest networks perform best without any dropout. It is also shown that the small batch size and large

Table 3: Search over Batch Size and Number of Epochs. Hidden Layers $= 4$. Nodes $= 20$.

(a) $p(\text{dropout}) = 0$.

	Batch Size		
Epochs	2070	4140	8280
5	0.657	0.370	0.294
25	0.666	0.705	0.674
50	0.617	0.642	0.656

(b) $p(\text{dropout}) = 0.05$.

	Batch Size		
Epochs	2070	4140	8280
5	0.180	0.110	0.215
25	0.629	0.603	0.439
50	0.652	0.544	0.677

(c) $p(\text{dropout}) = 0.1$.

	Batch Size		
Epochs	2070	4140	8280
5	0.158	0.133	0.05
25	0.404	0.264	0.210
50	0.632	0.498	0.570

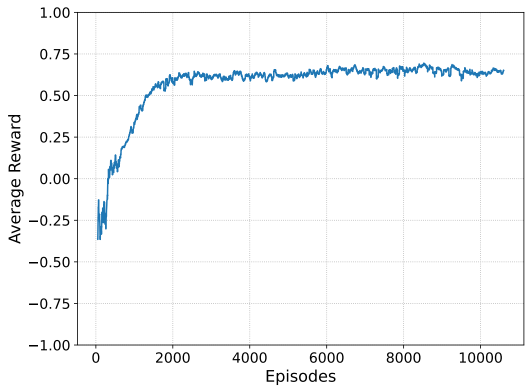


Figure 6: Reward Curve for 4 Hidden Layers. 20 Nodes. Batch Size = 2070. Epochs = 50.

number of epochs works well for network smaller than those presented in the prior experiment. As the networks get larger, though, performance begins to degrade to below the levels of the first experiment performed with the large batch size and small number of epochs. This suggests that for small networks, small batch sizes and many epochs are preferable, but for larger networks, larger batch sizes and fewer epochs are preferable. This relationship is well demonstrated for small networks as evidenced by experiment 2. However, more experiments would need to be conducted to prove this for larger networks. In the interest of time, and because larger networks are not preferable for on-board execution, this experiment is not performed in this work.

Comparisons

This work compares PPO and MCTS-Train for the multi-sensor EOS scheduling problem on the basis of performance, training time, and required model complexity. PPO produces the best performing networks across all sizes of networks after detailed hyperparameter tuning. The maximum reward achieved by PPO is approximately 0.75 reward. This is achieved both in training and in validation, as evidenced by Figure 5b. This performance is not even achieved by MCTS running 80 simulations-per-step. Furthermore, after function approximation, the highest reward achieved by

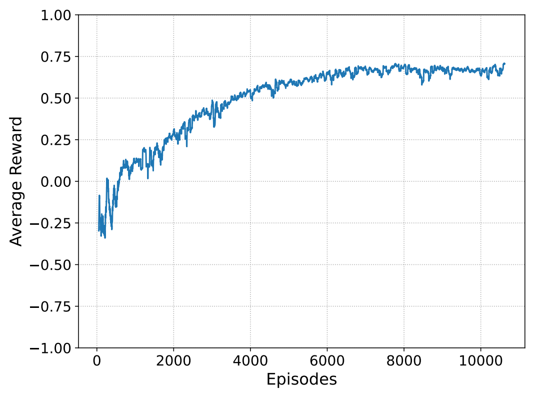
Table 4: PPO Network Size Search. Batch Size = 2070. Epochs = 50.

(a) $p(\text{dropout}) = 0$.

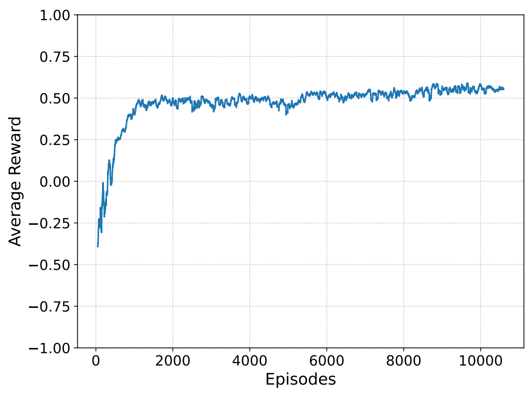
Hidden Layers	Nodes 10	20	40	80	160
1	0.696	0.678	0.681	0.668	0.637
2	0.677	0.677	0.637	0.612	0.556
4	0.698	0.637	0.587	0.556	0.532

(b) $p(\text{dropout}) = 0.05$.

Hidden Layers	Nodes 10	20	40	80	160
1	0.485	0.586	0.633	0.652	0.640
2	0.498	0.625	0.670	0.613	0.621
4	0.338	0.563	0.600	0.605	0.570



(a) 1 Hidden Layer. 10 Nodes Wide.



(b) 4 Hidden Layers. 160 Nodes Wide.

Figure 7: Example Reward Curves for Batch Size = 2070. Epochs = 50. No Dropout.

MCTS-Train is 0.64. This is an unexpected result considering that MCTS-Train produces optimal policies in References 6 and 14. However, this work uses planning horizons that are 90 decision-making intervals long. Past work uses 45 decision-making intervals. With 90 decision making intervals and 4 actions, the total number of potential trajectories is 4^{90}. In past work, there are only 4^{45} potential trajectories. While this is still a large amount, 4^{90} trajectories likely requires exponentially more simulations-per-step to produce optimal behavior.

PPO typically requires 3.5 hours of training time for each network. MCTS-Train requires 40 hours to generate the data and less than an hour to train the networks. Therefore, it is difficult to compare the two algorithms on execution time. If the goal is to perform large hyperparameter searches, they both take on the order of days to finish the experiment. However, if the goal is to train a single network, PPO is far superior as MCTS-Train will always require the intensive training data generation step.

After detailed hyperparameter tuning, PPO achieves more than 0.6 reward for most of sizes of neural networks. MCTS-Train, however, only achieves more than 0.6 reward for a couple of networks, which are extremely large. Based on the experiments, it appears that PPO's required model complexity is much smaller than MCTS-Train's. This is not a surprising result because policies are theoretically easier to learn than value functions. A small error in the computation of the value function can lead to a large error in the resulting policy. However, more work must be performed to determine if this is the explanation for MCTS-Train's relatively poor performance and if anything can be done to improve the algorithm's performance, especially when considering the fact that MCTS itself was not able to achieve the reward found by PPO. Furthermore, other value-based and policy-based methods should be compared to determine if this holds for those algorithms as well.

CONCLUSION

In this work, a multi-sensor Earth-observing satellite (EOS) scheduling problem is formulated as a Markov decision process (MDP). A high-fidelity astrodynamics simulation wrapped within a Gym environment is created for the problem to train decision-making agents. Proximal policy optimization (PPO) and MCTS-Train are applied to solve the problem and compared to one another on the basis of performance, training time, and required model complexity. PPO is shown to be the far superior algorithm in terms of performance and required model complexity. For hyperparameter searches, both algorithms take about the same amount of time to execute. However, if only a single network is required, PPO is the better algorithm because MCTS-Train takes 40 hours to generate the batch of training data.

Future work will compare other value-based methods (such as DQN) to other policy-gradient or actor-critic methods (i.e. A2C). Furthermore, additional EOS and small body science operations MDPs and environments will be investigated to provide a more comprehensive comparison for deep reinforcement learning EOS planning and scheduling. The hyperparameter searches for MCTS-Train will be expanded to determine if other combinations of parameters can produce better training data and better networks.

ACKNOWLEDGEMENT

This work is partially supported by a NASA Space Technology Graduate Research Opportunity (NSTGRO) grant, 80NSSC20K1162. This work is also partially supported by the Air Force STTR Program, FA8649-22-P-0833.

REFERENCES

[1] 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.

[2] X. Chen, G. Reinelt, G. Dai, and A. Spitz, "A Mixed Integer Linear Programming Model for Multi-Satellite Scheduling," European Journal of Operational Research, Vol. 275, No. 2, 2019, pp. 694–707, https://doi.org/10.1016/j.ejor.2018.11.058.

[3] G. Peng, R. Dewil, C. Verbeeck, A. Gunawan, L. Xing, and P. Vansteenwegen, "Agile Earth Observation Satellite Scheduling: An Orienteering Problem with Time-Dependent Profits and Travel Times," Computers & Operations Research, Vol. 111, 2019, pp. 84–98, https://doi.org/10.1016/j.cor.2019.05.030.

[4] S. Spangelo, J. Cutler, K. Gilson, and A. Cohn, "Optimization-based Scheduling for the Single-satellite, Multi-ground Station Communication Problem," Computers and Operations Research, Vol. 57, May 2015, 10.1016/j.cor.2014.11.004.

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

[6] A. P. Herrmann and H. Schaub, "Monte Carlo Tree Search Methods for the Earth-Observing Satellite Scheduling Problem," Journal of Aerospace Information Systems, 2021, pp. 1–13, 10.2514/1.I010992.

[7] Y. He, L. Xing, Y. Chen, W. Pedrycz, L. Wang, and G. Wu, "A Generic Markov Decision Process Model and Reinforcement Learning Method for Scheduling Agile Earth Observation Satellites," IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020.

[8] A. T. Harris, Autonomous Management and Control of Multi-Spacecraft Operations Leveraging Atmospheric Forces. PhD thesis, University of Colorado at Boulder, 2021.

[9] X. Zhao, Z. Wang, and G. Zheng, "Two-phase Neural Combinatorial Optimization with Reinforcement Learning for Agile Satellite Scheduling," Journal of Aerospace Information Systems, Vol. 17, No. 7, 2020, pp. 346–357.

[10] 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.

[11] I. Nazmy, A. Harris, M. Lahijanian, and H. Schaub, "Shielded Deep Reinforcement Learning for Multi-Sensor Spacecraft Imaging," American Control Conference, Atlanta, Georgia, June 8–10 2022.

[12] 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.

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

[14] A. Herrmann and H. Schaub, "Autonomous On-board Planning for Earth-orbiting Spacecraft," IEEE Aerospace Conference, Big Sky, MT, March 5-12 2022.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," 2017, 10.48550/ARXIV.1707.06347.