Unpublished, 2024

Planning Coordinated Observations and Downlinks for Multiple Satellites with Limited Data Storage for Wildfire Danger Prediction

Richard Levinson[1,2], **Vinay Ravindra**[1,3], **Sreeja Roy-Singh**[1,3]
[1] NASA Ames Research Center, Moffett Field, CA.
[2] Intelligent Systems Division
[3] Bay Area Environmental Research Institute
rich.levinson@nasa.gov, vinay.ravindra@nasa.gov, sreeja.nag@nasa.gov

Abstract

We present new planning methods, results and analysis for a novel application designed to improve wildfire danger prediction. We developed a centralized, ground-based planner to generate coordinated observation data collection and downlink plans for multiple small satellites with limited storage capacity.

Each observation target is associated with a science value (reward) based on WFPI-based Large Fire Probability (WLFP) data produced by the U.S. Geological Society. The planner maximizes the sum of science rewards collected for all observed targets on all satellites. Satellites are coordinated to avoid duplicate observations of a target by more than one satellite. The planner seeks to collectively observe as many targets as possible for the whole set of satellites, without exceeding storage capacity or dipping below minimum battery charge on any individual satellite.

The large scale of this problem is a challenge. We present experiments for a scenario with a system of 4 satellites and a 12-hour plan horizon using real world data. In this scenario, each satellite has 1800 potential images to collect, but can only store 60 images at a time, with limited downlink time available to free up storage. Data storage is modeled in Python as a First-In-First-Out queue for each satellite, with a queue capacity of 60.

The search space is generated and constraints are enforced procedurally, using a novel extension of Python which uses non-deterministic choice points to define command choices (decision variables) for each satellite. A binary decision variable is defined for every time point when a satellite can (a) collect data or not, or (b) downlink data or not. The number of decision variables explodes as the number of satellites and plan horizon increase. For our experiment scenario, there are 28,234 binary decision variables, producing a search space with $2^{28,234}$ nodes.

We present a novel use of Monte Carlo Tree Search (MCTS), adapted to be used as the search engine to explore this huge search space produced by the choice points embedded in Python code. To speed up performance, we use MCTS Tree Parallelization where each parallel process constructs a separate MCTS tree. We present experiments comparing solver speed and solution quality under various planner configurations, including: greedy heuristic vs. random simulation policy, soft vs. hard constraints, pure stochastic simulation vs. MCTS reinforcement learning, number of MCTS simulations, and number of parallel processes.

1 Science Problem and Application

Wildfires are unplanned fires, which may result from human activity (campfires, arson, escaped prescribed burn projects), lightning, volcanic activity, etc. In the United States (U.S.), from 2013 to 2022, there were an average of 61,410 wildfires annually and an average of 7.2 million acres impacted annually (Congressional Research Service, 2023). The economic impact of wildfires depends on their location, size, duration, and severity and can be extensive. The average annual federal spending in the U.S. on fire suppression totaled $2.5 billion (in 2020 dollars) between 2016 and 2020 (Congressional Budget Office, 2022).

Remote sensing by satellites has helped in monitoring of pre/active/post fire conditions. Within the active fire condition, early wildfire detection, and subsequent monitoring of progression of the fire front are possible at an global scale by satellites. For example, Fire Information for Resource Management System (FIRMS) distributes Near Real-Time (NRT) active fire data from the Moderate Resolution Imaging Spectroradiometer (MODIS) aboard the Aqua and Terra satellites, and the Visible Infrared Imaging Radiometer Suite (VIIRS) aboard SNPP and NOAA 20 (NASA, 2023). This is used to estimate fire-perimeters. In the post fire condition, post fire damage, and risk of landslides, floods are evaluated from satellite data. An example is the Burned Area Reflectance Classification (BARC), a satellite-derived data layer of post-fire vegetation conditions (United States Geological Survey, n.d.).

The focus of this paper is on the use of multiple spacecraft for monitoring in the pre-fire conditions over the contiguous U.S. The likelihood of a wildfire occurrence can be evaluated from the pre-fire condition (weather, vegetation, soil moisture, human activity, etc.) of a location, which can then help in preparedness of emergency response or activities geared towards mitigating the risk (such as prescribed burns). The term “fire danger” is used to quantitatively define the risk of a wildfire.

The distributed spacecraft mission considered in this work is the NASA CYGNSS mission (Ruf et al., 2018). It consists of 7 active spacecraft each carrying a GNSS Reflectometry (GNSS-R) instrument. The instrument gathers GNSS L1 reflected signals of the surface, and geophysical properties such as soil-moisture are inferred from it. At any given point in time, L1 signals from GNSS networks such as the

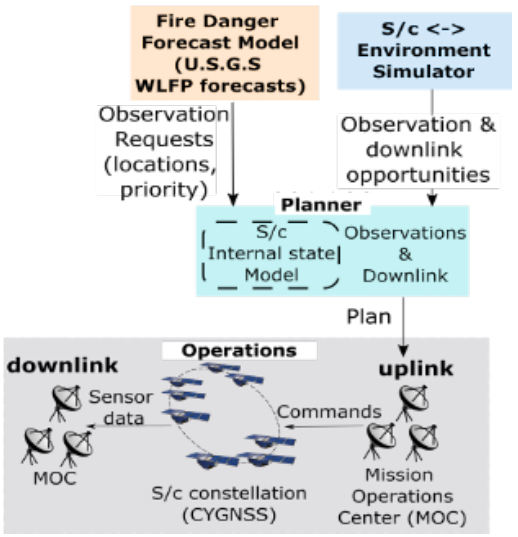

Figure 1: Concept of operations for satellite-based wildfire danger remote sensing

GPS, Galileo, GLONASS, Beidou are reflected off the surface, and the CYGNSS satellites receive the reflected signal within the field-of-view of its receiving antenna. Usually, the instruments are always kept ON in a low data volume gathering mode, where the information is clipped and compressed (with loss). We are interested in a high data volume gathering mode termed the RawIF mode, where observations in raw format can be retained and downlinked at locations of interest.

Figure 1 illustrates the setup for monitoring of the wildfire risk using CYGNSS satellites. The U.S. Geological Survey (USGS) fire danger forecast model is utilized. 7-day forecast products for fire potential index, large fire probability, and fire spread probability are produced and made available in public domain. The product of interest in this work is the Wildland Fire Potential Index (WFPI)-based Large Fire Probability (WLFP) forecast, which is an estimate of risk (probability) that a new fire will burn to more than 500 acres (202 ha) (U.S. Geological Survey, 2023).

Figure 2 shows the USGS WLFP day-1 forecast of 1 Aug 2020. This was produced over a 1km grid over the contiguous U.S., and each grid-point is assigned a WLFP value in the range 0 to >7. We remove grid-points associated with snow/ice, water, marsh, etc., which do not have a valid WLFP value. The fine 1km grid is upscaled by a factor of 1/10, to yield 47k grid points to reduce the spatial scale of the problem. (The upper limit of the upscaling factor is dictated by the size of the sensor footprint.) Observation requests are issued for all the (upscaled) grid points. The observation priorities are equal to the forecasted WLFP, i.e. we target locations of high pre-dicted WLFP.

The Earth Observation Simulator (EO-Sim) is utilized to calculate observation opportunities and ground-station contacts (Ravindra et al, 2021). The satellite orbits are propagated (forecasted) using a J2 analytical propagation model. At each second, calculations are done to produce the grid-points which could be observed (were the RawIF mode

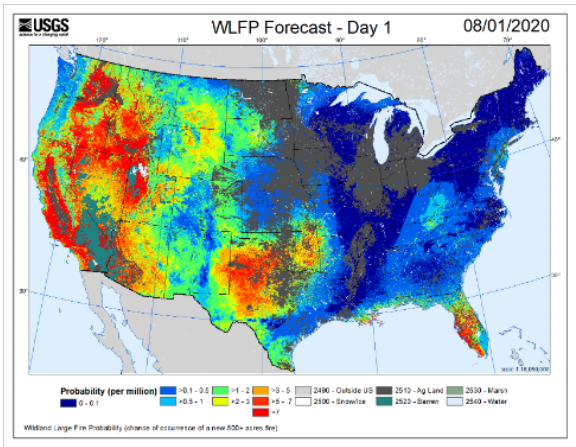

Figure 2: USGS WLFP forecast (day 1) of 1 Aug 2020

enabled). Three ground-stations are considered: Prioranet ground stations in (1) South Point, HI, USA, (2) Santiago, Chile and (3) Western Australia. Contact opportunities with these ground-stations are calculated to yield time-periods at which data downlink is possible.

Planner: The function of the planner is to schedule the observations and data downlinks, subject to resource (data storage capacity and downlink rate) and physical (limited opportunities to observe and downlink) constraints.

Operations: The concept of operations involves the plan being verified by the Mission Operations Center (MOC) and uploaded to the satellites. Data is downlinked and is utilized to improve the predictions of the fire danger (Ravindra et al., 2023).

Planning challenges:

- The key challenge is the huge search space, a sequence of $2^{28,234}$ binary choices for our experiments. This is due to modeling multiple coordinated satellites with 1 second temporal resolution over large planning horizons (12 hours in our experiments). The small 1-second temporal resolution of the planning model is related to the satellites' speed.
- Integrated model for data collection and data downlink, with extremely limited data storage capacity. It takes 1 second to collect an image, and 20 seconds to downlink 1 image, and we can only store 60 images at a time.
- Modeling a First-In-First-Out (FIFO) data storage queue is a major challenge for declarative Mixed Integer Program (MIP) equations (we're still working on it), but easy for the procedural model presented in this paper.
- Coordinating the operations of distributed satellites.
- Every image we collect contains multiple target locations. The planner can only choose images not individual targets, but our objective is based on the targets. As far as we are aware, this is an extra degree of freedom we have not seen in related research, which introduced complications. Part of this challenge is related to coordinating the satellites so they do not observe the same target.

Feature	Levinson et al., 2022	Current article
Observation planning	Yes	Yes
Data storage capacity & downlinking	No	Yes
Maneuvering	Yes	No
Power consumption/ production with minimum power constraint	Yes	Yes
Science application	Soil Moisture monitoring	Wildfire danger prediction
Scale: # target locations	1.7 million targets around the world	47k targets in continental US.

Figure 3: Comparison with (Levinson et al., 2022)

- Observation opportunities to targets, and ground-station contacts are aperiodic. Periodicity would have helped reduce the problem size, but the nature of the orbital dynamics of the satellites results in no definite pattern to exploit.

Related research: Within the surveyed literature, (Levinson et al. 2022) has elements in common with that presented in this paper. That planning problem involved determining the optimal series of observations to be made by a constellation of satellites over a spatial domain encompassing majority of the land surface of Earth. The land surface was represented by a set of discrete point locations and each point was assigned a scientific value. Satellites had the ability to slew and collect rewards which varied based on the observation angle. The key differences with respect to our work are summarized in Figure 3.

(Cho et al., 2018) present a Mixed Integer Program (MIP) solution for generating observation and downlink plans for multiple satellites with data storage capacity limits. Although similar to our problem in some ways, it's very different in others. First, they are scheduling generalized "observation" and "downlink" tasks without tracking details of when any specific target location is observed or downlinked. In contrast, each of our observations collect data on multiple target locations, and our science rewards are tied to each target rather than the "observation" task.

Another major difference is that they separate downlink planning from observation planning as to different planning problems. They first produce a downlink plan (the time windows when each satellite can downlink data), and then schedule the observations to maximize the rewards from all selected observations. The downlink windows generated by the first step are used to constrain the observation tasks, while maximizing rewards for all observation tasks, and enforcing a constraint that data storage never exceeds capacity.

Their objective function is not a function of downlinking, as the scientific value is obtained only by observation during the observation scheduling phase. They don't track when image data for any specific target location is down-linked while our objective depends on each target being both observed and downlinked. This means they don't deal with complications such as a target location's data being only partially downlinked at any specific time. Tracking which target data has been downloaded is important because it only becomes useful when it reaches the hands of users.

They provide experimental results comparing their system vs. a first-in-first-out (FIFO) "greedy" heuristic (fill up storage ASAP), and a model without storage (or energy) capacity constraints to provide an upper bound on the objective function.

(Herrmann and Schaub, 2023) use Monte Carlo Tree Search (MCTS) for satellite observation scheduling and downlinking with capacity constraints for a single satellite, but they are focused a very different application than ours. They use MCTS to "play" against a high-fidelity Markov Decision Process (MDP) in order to generate training examples for a neural net. The neural net produces a reactive state-action *policy* for onboard (online) execution, to optimize observation collection for a single satellite. MCTS is used offline to generate state/value estimates to train a neural net and produce a policy for online execution (on-board the satellite). No "optimal" plan is produced as the MCTS plans are only training cases for the reactive online policy. They evaluate how well their policy handles situations which are outside the training set. In contrast, we focus on optimizing complete plans for specific situations known in advance. Their model includes storage capacity limit and downlink rates.

Flow Shop Scheduling is a variant of job shop scheduling where each job must flow through a fixed sequence of different machines. Our application involves a model of collecting each image in a First-In-First-Out (FIFO) queue so that images are downlinked in the same order they are collected. This can be seen as a variant of flow shop, where the FIFO queue represents a sequence of machines each image must flow through. Each image flows from the back of the queue to the front. Our application differs from standard flow shop because the # of machines each job must flow through depends on how much data is already in the queue, while in standard flow shop all jobs flow through exactly the same # of machines. In our case, if the queue is empty, then the job only flows through 1 machine. If the queue is full, then the last job flows through as many jobs as the queue size. Also, classic flow shop requires all jobs are scheduled, but we have too many observation jobs to do them all, so some jobs will never enter the flow shop at all. It's an oversubscribed problem.

(Xiao et al., 2019) present a variation of "flow shop" where there are only 2 machines: observe and downlink. They present a MIP model for a "multi-satellite observation and data downlink scheduling problem". They integrate observations and data downlinks but have no storage limit (assuming infinite capacity), and all targets must be observed, in contrast with our application. They support FIFO data collection/downlink queue model like us, where tasks are

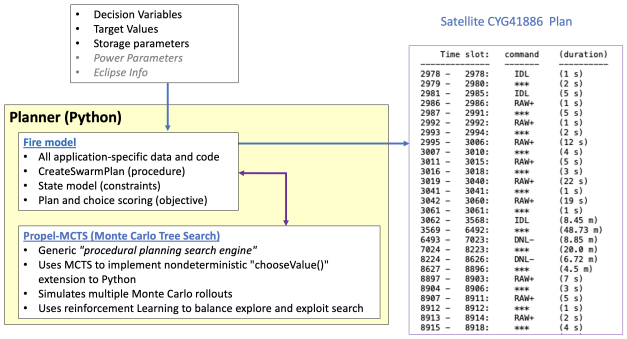

Figure 4: Planner Architecture

downlinked in the same order they are collected. They also support a "non-permutation" schedule where the collection and downlink sequences may be different which we do not do.

It's unclear how to add a storage capacity constraint to their model because that requires ensuring capacity is never exceeded, at any time. They represent time intervals rather than individual timepoints. We'd need to add constraints to enforce storage capacity for every individual timepoint without significantly increasing the search space to model every tick within their time windows.

Their problem size is relatively small, 10 "tasks" (observations or downlinks) in four days while ours is approximately 2000 tasks in 12 hours. Their objective function is different. Since all tasks must be completed, they are minimizing completion time (makespan), while we are given the completion time (our 12-hour plan horizon) and are maximizing the science rewards collected for the subset of targets which get selected for the plan.

Figure 4 shows the planner inputs, processing, and output. It produces a plan for each satellite, shown on right, which specifies what command to execute for every second in the 12-hour plan horizon. The command "***" indicates "forced idle" times when there are no opportunities to either collect an observation or downlink. The command "RAW" and "DNL" are the names of our observation and downlink commands, respectively. The + and - symbols indicate when data is added or removed from storage, respectively.

Decision Variables: A key input is a set of *decision variables*. We define a set of decision variables $cmd_{i,t}$, each representing the command choice for sat s_i at time t. $\forall t \in \{$all timepoints when sat i has either a target viewing opportunity or a downlink opportunity$\}$. The duration of each timepoint (i.e., the temporal step size) is 1 second. The reason for this 1 second granularity is related to the speed of the spacecraft.

Figure 5 shows examples of these decision variables. The first variable says sat 2 at time 28 has a choice of observing targets 27 and 39, or it may remain idle. The second variable says sat 3 at time 42 has a choice of downlinking data to the ground station in Hawaii, or remaining idle.

```
{"var": "s2.t28", "choices":["OBS.27,39", "IDL"])
{"var": "s3.t42", "choices: ["DNL.HI", 'IDL'])
```

Figure 5: The *choices file* defines the planner's search space

A preprocessing step produces this list of decision variables with their domains (choices) shown in figure 5. The preprocessor reads a variety of files which specify when each satellite has observation and downlink opportunities, and it consolidates them into a single "choices" file in the above format which defines the planner's search space. The variables are sorted chronologically. For example, all commands for tick 10 will be chosen before any commands for ticks > 10.

We don't model every second in the horizon, only those times when there is a choice to make. There are no decision variables for times when there are no observation or downlink opportunities. The experiments presented below have 4 satellites and a 12-hour plan horizon, resulting in a total of 28,234 binary decision variables. This means the number of nodes in the search space of unique plans is $2^{28,234}$.

This huge search space is a key challenge for the planner: How to find a needle (good plan) in a haystack (infinite plan variations) within a couple of hours of search? We are aiming for the ability to upload new satellite plans every 3 hours, which dictates our solver time limit.

Another key input to the planner is a list of target values, which specify the scientific value (reward) for observing and downlinking every target which appears in the choices file. This value is a function of the USGS. WLFP forecast data in Fig 2. (U.S. Geological Survey, 2022).

The planner's objective function maximizes the aggregate reward for the collective system of satellites, where those rewards are based on the target values for images collected and downlinked.

Planner methods: Our planner is implemented in a new version of Propel: The Program Planning and Execution Language (Levinson 1995; 2005; 2020). This new version, dubbed "Propel-MCTS" integrates non-deterministic Python with Monte Carlo Tree Search (MTCS). It allows non-deterministic choice points (assignment statements) to be embedded anywhere in Python, and uses MCTS as the search engine that searches through the space of program variations defined by those choices.

All versions of Propel support automatic planning with a procedural action model (in this case, Python). This is part of a line of research on integrated planning and reactive execution, where the planner uses a procedural model, sometimes called an "operational model" (Patra et al.). Propel is also related to systems designed for tightly integrated planning and reactive execution including IDEA (Muscettola et al., 2000; 2002) and (Neufeld et. al., 2018; Neufeld 2020).

This application does not currently operate in a reactive execution environment. However, we leveraged the procedural modeling flexibility of Python for rapid prototyping and development to implement our application within a matter of weeks. In particular, Python makes it very easy to implement our application's dynamic constraint enforcement (described below). In contrast we have spent many months developing a declarative MIP mode which quickly becomes intractable and is not yet scalable to work on even our smallest test case. We are also preparing for future application

```
def createSwarmPlan():
   # Top-level application code
   #     executed (simulated)on each rollout
1  initializeState()
2  decisionVars = initializeDecisionVars()
3  while decisionVars:
4     var = decisionVars.pop()
5     choices = commandChoices(var)
6     cmd = chooseValue(choices,"greedy/random")
7     state = updateState(cmd)
8     decisionVars = propagateChoice(cmd)
```

Figure 6: Sequential decision procedure in Propel with choice points

scenarios to support active wildfire management which will require more reactive execution.

Figure 6 shows our Propel application's sequential decision making procedure. Line 1 initializes the state model for each satellite (data storage is empty and battery charge is full). Line 2 creates all the decision variables. We create one decision variable for every time t when sat i has an observation or downlink opportunity.Each variable's domain is either {observe, idle} or {downlink, idle}.

Line 3 tests if any decision variables remain unresolved. If there are open decision variables, then pop the first one from the list. They are chronologically sorted, so this will be the next variable in sequence, although it could be for any of the satellites. Examples of this variable are shown in figure 5. Each has a name and a set of (binary) choices. Line 5 retrieves the choices for the current variable and line 6 calls `chooseValue` with those choices to add to the MCTS tree.

`chooseValue` statements (line 6) define the search space explored by MCTS. `chooseValue` takes an optional heuristic parameter to sort the choices, typically an application-specific, greedy heuristic. If no heuristic is specified, then choices are chosen randomly.

After each choice is made, the state is updated on line 7 to reflect adding or removing data from storage, and updating the battery charge estimate. The *State Model* is a python dictionary representing the state of each satellite (data storage and battery charge) over the course of the plan horizon. The state is updated after each command is added to a satellite's plan.

Data Storage Model: Our data storage model is based on the CYGNSS satellite specs (Ruf et al., 2018). Each observation collects one "image" (corresponding to 1 second of observation) which may capture several targets (Figs 5 7). Each observation (image) command fills up 96.22 megabits of storage, and each downlink command frees up 4 megabits (about 5% of an image). If storage is full, it takes about 20 downlink commands to free up enough space to collect another image. Images are stored in a First-In-First-Out (FIFO) queue, so that they are collected and downlinked in the same order.

Figure 7 illustrates how images are placed in the FIFO queue. Each image in the queue is associated with a set of targets captured by the image, the aggregate target value for all targets in the image, and the percent of each image downlinked at any given time. The objective is a function of the percentage of each image which is downlinked. One unique

Queue Position	Image ID	Targets	Image Value	Downlink %
1	21	6,39,25,40	57	100
2	22	39, 44, 22	23	65

Figure 7: Data storage queue where images are collected and % of image downlinked is tracked

aspect of our application is that each image collects data on multiple targets and our objective is to maximize the target values. This is an extra complication for the planner to deal with compared to related work where each image corresponds to an individual target and reward.

Constraints are enforced on line 8 of figure 6, by `propagateChoice()` which uses forward checking (Russell and Norvig, 2020) to rule out any future choices which are inconsistent with the newly chosen command. For example, if an observation command fills up storage to capacity, then we remove all future observation choices until the next downlink opportunity. The decision variables are updated as a result.

This forward constraint checking is a powerful feature. When `propagateChoice()` removes one of the binary choices for a variable, then that variable doesn't need to be a branch in our search space. Ultimately the number of choices in any given plan is less than the original number of decision variables because many variables are removed along the way through choice propagation.

Coordination across the swarm is achieved by enforcing a "no duplicate observations" constraint. When one satellite observes a target, that target is removed from the choices of all future observations. All constraints are implemented in Python in `propagateChoice(cmd)`, which operates on the state dictionary that is updated on line 7.

Constraints

1. No observations allowed when storage is full
2. No downlinks allowed when storage is empty
3. Always downlink if storage is not empty
4. Never allow battery charge to dip below 70%
5. No duplicate target observations (across all satellites)

The full set of constraints enforced by `propagateChoice()` are shown above. The first 4 are enforced per satellited and the last is enforced across the entire set of satellites. Initially, we did not have constraint 3. The Greedy heuristic prefers to downlink vs. re-main idle because downlinking will contribute to the objective while idle will not. However, we discovered that soft constraint (preference) was not utilizing every downlink opportunity. We added constraint 3 as a hard constraint which removes the "idle" choice for every downlink opportunity when storage is not empty. This "forced down-link" had a significant impact on solution quality as will be shown in our experimental results.

Objective Function: The planner's objective is to:

maximize $\sum_i imageValue(i) \forall i \in$ {all images in all satellite plans}. We calculate the imageValue as the sum

```
imageValue(i,t) =   (Σ_t targetValue(t)/2)
        + ((Σ_t targetValue(t)/2)*(downlinkPct(t)))
                          ∀ t ∈{targets in image i)
```

Figure 8: imageValue calculation used in objective function

of the target value for all targets in the image. The image value is split in half: 50% of value is collected when image is taken, and 50% of value is collected as image is downlinked (pro-rated by %downlinked). The image value in the objective is calculated as shown in Figure 8, which determines the value of any image at a given time, which is a function of the % of image downlinked at that time.

Greedy heuristic: The `chooseValue()` statement (line 6) takes an optional local "greedy" heuristic, which sorts choices based on their expected contribution to the objective. Observations and downlinks both contribute to the objective (figure 8), while IDLE does not, so observation and downlink commands are preferred over IDLE. If the optional heuristic is not provided, then choices are random.

Why MCTS? The code in Fig 6 is similar to that used by (Levinson et al., 2022). A key difference is the search engine underneath which explores the space created by the `chooseValue` statements. (Levinson et al., 2022) used a search engine which maintained a search node for every explored choice, to facilitate backtracking. That proved to be a huge performance problem because it required more memory than available on the Mac laptop where it ran. That performance problem is the primary reason we chose to use MCTS, which requires only a single process per tree, resulting in linear scaling of time and space requirements as the problem size grows.

With MCTS, the code in Figure 6 is simulated on every rollout with different command choices selected on each simulation. MCTS helps because our search space is huge, and MCTS is designed for huge search spaces where a balance between exploring new choices vs. exploiting prior choices which worked well is critically important for avoiding bias and local minima. MCTS uses a domain-independent heuristic called Upper Confidence Bound for Trees (UCT) to control the bias (preference) for exploit vs. explore.

Figure 9 illustrates our MCTS tree. The root node starts at time 0 (T=0), and each node has two children for a binary command choice: [OBS, IDL] or [DNL, IDL]. These choices are defined by the *choices file* shown in figure 5. Each child node represents a choice for a time later than its parent. A "rollout" also called a "simulation" involves following a sequence of decision variable choices from the root (time T = 0) to the end of the plan horizon (T = 43,200 for our 12-hour test scenario). At the end of each rollout, the objective is backpropagated through the tree to update the reinforcement learning statistics, tracking the # of times each node is visited by a rollout and the total accumulated reward from all rollouts which passed through it.

For this application, we are using MCTS *"off-label"* because we generate & collect whole plan vs. incrementally planning for 1 step at a time. This is because we are not executing the plan and we don't have exogenous events or

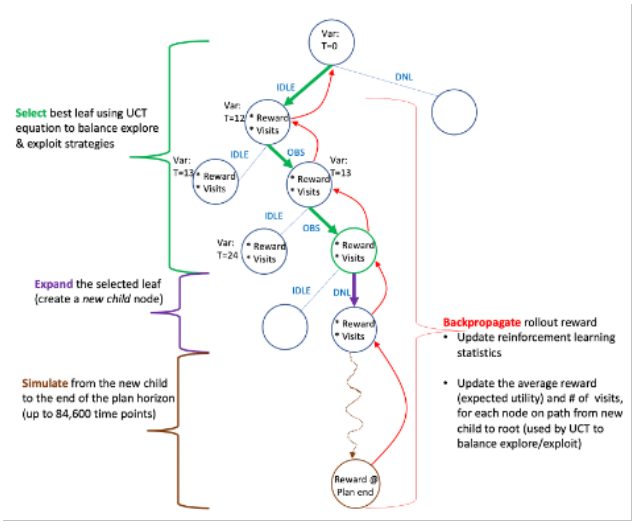

Figure 9: MCTS tree for our application. Each node corresponds to a binary decision variable

an "opponent". If we wanted to use MCTS the usual way as a two-player game, we could view the storage limit and the observation and downlink opportunity time windows, as the "moves" of an "opponent" (satellite, environment). For example, if we greedily fill up all storage ASAP, then our "opponent" blocks our ability to collect any more data until we free up some storage space. Future work may also include observation and downlink execution failures, such as the image sensor failing, or a downlink failing because the ground station is busy servicing a higher-priority mission. Propel itself does support using MCTS in a traditional reactive manner, where a single next step is selected and executed, and has been used that way in other applications.

An important contribution of this work is adapting the MCTS stages *select, expand, simulate* and *update* to work on a search space generated by nondeterministic Python. `chooseValue()` statements can be embedded anywhere in arbitrary Python code, defining the search space. MCTS is the search engine which explores that space using reinforcement learning and balancing exploration and exploitation via the UCT equation. During the select stage of each rollout, `chooseValue()` uses UCT to choose which node to expand. The *expand* stage and *simulation* stages both use the same policy for making choices: either the greedy heuristic or random, whichever is specified in the `chooseValue` statement. Experiments below explain how/when a mix of greedy and random is also used.

New MCTS stage for procedural models: Due to the procedural nature of the Python code within which choice points are embedded, we needed to add a new MCTS stage: *replay*, which occurs between the *select* and *expand* stages as follows: When an MCTS tree leaf is selected to be expanded, and that leaf is not the root, then we must *"replay"* all the choices from the root to the leaf to match the Python computational context for each choice point. Since each rollout starts a *new* execution of the Propel application shown in Figure 6, we need to replay all of the plan's prior choices to

retrace the plan choices from the beginning of the rollout to the current choice being selected (which may be embedded in multiple loops, conditional and subroutine calls). For example, if a choice point is encountered on the 3rd iteration of a while loop, or if some state condition is true when a choice point is reached, then we need to set up that same context so that the computational context match when the leaf node is expanded.

Parallel MCTS: To speed up solver performance we leverage the 10 CPU cores available on our Mac laptop. We use Python's multiprocessing module to exploit multiple CPU cores. (Steinmetz and Gini, 2021) and (Chaslot et al., 2008) describe 3 ways to leverage multi-processing for MCTS: Leaf parallelization, Tree parallelization, and Root parallelization. We implemented Root Parallelization, and Tree parallelization is in development.

With root parallelization, each process creates and manages a separate MCTS tree, and no information is shared between trees. Each process creates a tree by running N rollouts (Monte Carlo simulations). For each tree (process), the rollout with the highest objective score is tracked. After all processes finish, the rollout with highest objective across all processes is selected as the final plan.

The rollout (simulation) policy defines how choices are made during the MCTS expand and simulation stages. The policy may be 100 % greedy, 100 % random, or a mix of the two, which is described further in the next section.

2 Evaluation

All experiments use the same scenario which includes 4 satellites and a 12-hour plan horizon. The planner's search space (defined by the choices input file) is identical for all test cases. The test cases differ only in the planner configuration. All experiments were run on a 2023 MacBook Pro, M2 Pro chip, 32 GB, with 12 total CPU cores (8 performance and 4 efficiency).

Our experiments are designed to quantify the impact of several planner configuration parameters on solution speed and quality. We define solution speed as the amount of solver time required and solution quality is measured by 3 quantities: Objective score, # of images collected, of images downlinked.

Figure 10 shows the test cases used to evaluate solver performance and solution quality for different planner configurations. The cases are sorted from highest objective score to lowest. The columns are: test case ID, # of rollouts, # of parallel processes, rollout policy (greedy/random or % random), *'hrd dnl'* is the hard constraint for downlinking (constraint 3), reinforcement learning enabled (L), solve time for all processes to complete all rollouts, objective score. The last column is the ratio of # of downlinked targets over the # of observed targets. In cases with multiple rolls and/or multiple processes, the stats for the highest scoring objective are shown.

The rollout policy describes how choices are made for node expansion and simulation. It is either 100 % greedy (G) or 100% random (R). With multiple processes, each process uses greedy with some percentage of choices being random.

ID	# rolls	# procs	Greedy/ Random	Hrd Dnl	L	Time	Obj	downlinked/ observed
1	**40K**	**10**	**R**	**Y**	**Y**	**22.6h**	**6131**	**521/2024**
2	20K	20	R	**Y**	**Y**	22.3h	5758	521/1944
3	10K	10	R	**Y**	**Y**	6 h	5654	521/1850
4	4K	20	95% R	**Y**	**Y**	4.3 h	5354	521/1746
5	10K	10	90% R	**Y**	**Y**	6.25 h	5278	521/1654
6	4K	20	95% R	**Y**	N	4.3 h	5271	521/1670
7	10K	10	90% R	**Y**	**Y**	7 h	5238	521/1681
8	4K	10	90% R	**Y**	**Y**	2.3 h	5144	521/1673
9	1	1	R	**Y**	N	5.1 s	5116	521/1677
10	4K	10	90% R	Y	N	2.25 h	5097	519/1624
11	1	1	R	**Y**	N	5.2 s	5049	521/1601
12	1	1	R	**Y**	N	5.1 s	4995	521/1569
13	40K	1	**G**	**Y**	**Y**	20.5 h	4581	521/1366
14	30K	1	**G**	N	**Y**	19.3 h	4456	521/1297
15	10K	10	**G**	**Y**	**Y**	5.5 h	4211	521/1195
16	4K	20	5% R	N	**Y**	5.15 h	4193	511/1185
17	3K	20	10% R	N	**Y**	4.02 h	4189	506/1185
18	2K	20	15% R	N	**Y**	2.48 h	4187	499/1194
19	4K	10	10% R	N	**Y**	3.13 h	4176	508/1181
20	3K	10	10% R	N	**Y**	2.05 h	4166	506/1175
21	2K	10	10% R	N	**Y**	82 m	4160	505/1186
22	1K	10	10% R	N	**Y**	40 m	4123	506/1163
23	1K	20	5% R	N	**Y**	71 m	4115	514/1156
24	1	1	**G**	**Y**	N	5 s	3987	521/1106
25	1	1	**G**	N	N	5 s	3987	521/1106
26	1	1	R	N	N	4.8 s	3891	381/1323
27	1	1	R	N	N	4.8 s	3664	377/1192
28	1	1	R	N	N	4.9 s	3521	380/1175

Figure 10: Evaluation test cases and results

The percentage listed in the column is the random percentage for the highest scoring rollout in in that process, among all rollouts in that process. This is the same rollout used for the time, obj, and downlinked/observed statistics in the rightmost 3 columns.

No learning can occur with only a single rollout (cases 9-11, 23-27). With multiple rollouts and learning is disabled (cases 6 and 10), it is pure stochastic simulation and the best result over all simulations is shown in the table. In this mode without learning, MCTS stages select, expand, and backpropagate are skipped. Each rollout starts with the root node in "simulation" stage. This configuration is designed to assess the impact of the UCT reinforcement learning on solution quality. Cases 9, 11, and 12 are same test. They are all single rollout and 100% random, so we repeated them 3 times to demonstrate stochastic variance with random choices. The same is true for cases 26-28. These cases represent "pure" stochastic search, without any reinforcement learning (L). The difference between the first set (9,11, and 12) and the second set 26-28, is that second set has the hard downlink constraint turned off, resulting in much lower objective.

In most cases above, when multiple processes are used, then 1 process is greedy (0% random), and 1 process is 100% random, and all other processes have different random percentages, evenly spaced, ranging between 0 and 100. If there are 10 processes, the random percentage for each process increases from 0 by increments of 10. For example, if there are 20 processes, the random percentages increase in increments of 5%. The exceptions are cases 1,2, and 3, which are denoted with "R" in the greedy/random column.

For those cases we forced all processes to be 100% random.

Analysis of results: Solve time scales linearly with # of rollouts (2 to 4 secs/rollout). Solve time does not increase with # of processes up to 10 processes because we have 10 CPU cores on our laptop. Multiple CPUs are still leveraged up to about 20 processes after which there are diminishing returns on speedup.

Mandatory downlinks (hard constraint 3) results in significant improvement of objective score and showed the greedy heuristic is weaker than we originally thought. Without the hard constraint, the best scores are with 5-10% randomization, leading us to the hypothesis that the heuristic was pretty good. However, after adding the hard constraint for mandatory downlinks, the best scores are with 90-95% randomization, suggesting our greedy heuristic is not as good as we thought. We believe this improvement is because higher randomization results in more diverse training cases for MCTS.

The reinforcement learning provided by UCT helps to improve the objective score, compared to cases where learning is turned off (pure stochastic simulation). This shows that although we use MCTS off-label, UCT helps.

Figure 10 tests are sorted in from best objective to worst. Test 1 is the best result with objective 6,131, with 40K rollouts and 10 processes. This case does not use the greedy heuristic at all. Each of the 10 processes make 100% random choices. Test 2, the second best, has 20K rollouts and 20 processes. Both cases 1 and 2 have a total of 400K rollouts, but 1 is better because each of it's 10 processes is learning from 40K MCTS training simulations (rollouts), while in case 2, each of the 20 processes is learning from 20K training simulations. This provides clear evidence that MCTS produced better results (higher objective) with the random simulation policy and with the reinforcement learning of UCT.

Results clearly demonstrate that 521 is the maximum number of images which can be downlinked for our 12-hour scenario. Future work will chain together multiple horizons so any collected data not downlinked in one horizon will be downlinked in the next horizon.

3 Conclusion

Our experiments have provided initial answers to our research questions about what is the best planner configuration for our application. Our best configuration was: 40,000 rollouts with 10 processes vs. 20,000 rollouts with 20 processes. Both cases did 400,000 rollouts but it was clearly better to have fewer processes, because that each MCTS process learned from more rollouts rather than spreading out those training cases among more processes which did not share results.

We have learned the recommended way to implement our proposed solution for distributed satellite observation downlink planning is:

- Have the hard downlink constraint
- Have a (100%) Random rollout policy (vs. Greedy) produces better solutions by learning from more diverse training examples.
- Investigate the returns on the # processes on the particular machine the code is set to run, and assign the appropriate # processes.
- Do as many rollouts as time permits to learn from more

Contributions: We developed a novel solution for an important and novel approach to improved wildfire danger prediction. We presented novel planning methods for managing our application's huge search space. We presented empirical evidence about which planner configurations work best.

- Our experiments helped answer key research questions for our application, including: what planner configuration produce the highest objective functions, what is a good objective score, and how long does the planner need to run to achieve a good objective score.
- Adapted MCTS for non-deterministic Python. Identified and implemented a new MCTS stage, *Replay*, required for procedural action models.
- Propel-MCTS has demonstrated value for rapid development of procedural planner constraints (vs. declarative models like MIP).
- Integration of MCTS with Python has many potential benefits for a range of other applications, particularly those requiring integration of planning and reactive execution. Propel-MCTS enables planning with a procedural action representation (hierarchical subroutines, loops, conditionals), which can be directly executed for both planning and reactive execution.

Future Work: We are developing methods for MCTS Tree parallelization, where multiple processes will share and update a single MCTS tree, so the learning statistics will be shared between all processes. With our current root parallelization, no learning is shared between processes. We are developing a Mixed Integer Programming (MIP) model to provide an optimality benchmark to compare with our MCTS results. We plan to evaluate different values for the UCT tuning parameter C, which controls search bias towards exploration vs. exploitation. We plan to double the problem size to 8 satellites and a 24-hour plan horizon.

4 Acknowledgments

References

Chaslot, G., Winands, M., Herik, H.. (2008). Parallel Monte-Carlo Tree Search. 60-71. 10.1007/978-3-540-87608-36.

Cho, D. H., Kim, J. H., Choi, H. L., Ahn, J. (2018). Optimization-based scheduling method for agile earth-observing satellite constellation. Journal of Aerospace Information Systems, 15(11).

Congressional Research Service. (2023). Wildfire Statistics. Retrieved from https://sgp.fas.org/crs/misc/IF10244.pdf Congressional Budget Office. (2022, June). Wildfires. https://www.cbo.gov/publication/57970

Gurobi. 2022. https://www.gurobi.com Accessed: March 3, 2022.

Herrmann, A., Schaub, H. (2023). Reinforcement Learning for the Agile Earth-Observing Satellite Scheduling Problem. IEEE Transactions on Aerospace and Electronic Systems.

Levinson, R. 1995. A General Programming Language for Unified Planning and Control. Artificial Intelligence, Vol. 76. Special Issue on Planning and Scheduling. https://www.sciencedirect.com/science/article/pii/000437029400075C

Levinson R. 2005. Unified Planning and Execution for Autonomous Software Repair, ICAPS 2005, Workshop on Plan Execution. https://icaps05.icaps-conference.org/documents/ws-proceedings/ws7-allpapers.pdf

Levinson, R., 2020. Integrated Planning, Execution and Goal Reasoning for Python. ICAPS 2020, workshop on Integrated Execution (IntEx).

Levinson, R., Niemoeller, S., Nag, S., Ravindra, V. (2022, June). Planning Satellite Swarm Measurements for Earth Science Models: Comparing Constraint Processing and MILP Methods. In Proceedings of the International Conference on Automated Planning and Scheduling (Vol. 32, pp. 471-479).

Levinson, R., Nag, S., and Ravindra, V. 2021. Agile Satellite Planning for Multi-payload Observations for Earth Science, Proceedings of the International Workshop on Planning and Scheduling for Space (IWPSS). 2021.

Muscettola, N., Dorais G.., Fry,C., Levinson, R., Plaunt, C. 2000. A Unified Approach to Model-Based Planning and Execution. The 6th Int'l Conf. on Intelligent Autonomous Sys-tems. Venice.

Muscettola, N., G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt, 2002. "IDEA: Planning at the core of autonomous reactive agents," in Proc. of the 3rd International NASA Workshop on Planning and Scheduling for Space, 2002

Nag S., et al., 2019. "Autonomous Scheduling of Agile Space-craft Constellations with Delay Tolerant Networking for Reactive Imaging," presented at the International Conference on Planning and Scheduling (ICAPS) SPARK Applications Workshop, Berkeley, California, U.S.A., 2019

Nag, S., Moghaddam, M., Selva, D., Frank, J, Ravindra, V., Lev-inson, R., Azemati, A., Aguilar, A., Li, A., Akbar, R., 2020. D-Shield: Distributed Spacecraft with Heuristic Intelligence to Enable Logistical Decisions, Proc. of the 2020 IEEE Interna-tional Geoscience and Remote Sensing Symposium.

Nag, S., Moghaddam, M., Selva, D., Frank, J., Ravindra, V., Levinson, R., Azemati, A., Gorr, B., Li, A., Akbar, R. 2021. "Soil Moisture Monitoring using Autonomous and Distribut-ed Spacecraft (D-SHIELD)", IEEE International Geoscience and Remote Sensing Symposium, Virtual, July 2021 NASA. (2023). FIRMS: Fire Information for Resource Man-agement System. Retrieved from https://firms.modaps.eosdis.nasa.gov/

Neufeld, X., Mostaghim, S., Brand, S. (2018). A Hybrid Approach to Planning and Execution in Dynamic Environments Through Hierarchical Task Networks and Behavior Trees. Artificial Intelligence and Interactive Digital Entertainment Conference.

Neufeld, X., 2020. Long-term planning and reactive execution in highly dynamic environments, Dissertation, Otto-von-Guericke-Universität Magdeburg. http://dx.doi.org/10.25673/35675.

Patra, S., Mason, J., Ghallab, M., Nau, D., Traverso, P., 2021. Deliberative acting, planning and learning with hierarchical operational models, Artificial Intelligence, Vol. 299

Ravindra, V., Ketzner, R., Nag, S. (2021, July). Earth Obser-vation Simulator (EO-Sim): An Open-Source Software for Observation Systems Design. In 2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS (pp. 7682-7685). IEEE.

Ravindra, V., Roy Singh, S., Moghaddam, M., Nelson, K., Levinson, R., Melebari, A., Kannan, A. (2023). Distributed Spacecraft with Heuristic Intelligence to Enable Logistical Decisions for Global Navigation Satellite System Reflectometry (GNSS-R) of Wildfires. Presented at Earth Science Technology Forum. https://esto.nasa.gov/forums/estf2023/Presentations/S5P3RavindraSinghI

Russell, S., Norvig, P. 2020. Artificial Intelligence: A Modern Approach. Fourth Edition. Pearson.

Ruf, C. S., Chew, C., Lang, T., Morris, M. G., Nave, K., Ridley, A., Balasubramaniam, R. (2018). A new paradigm in earth environmental monitoring with the Cygnss small satellite constellation. Scientific reports, 8(1), 8782.

Steinmetz, E., Gini, M., "More Trees or Larger Trees: Parallel-izing Monte Carlo Tree Search," in IEEE Transactions on Games, vol. 13, no. 3, pp. 315-320, Sept. 2021, doi: 10.1109/TG.2020.3048331.

United States Geological Survey. (n.d.). BAER FAQs. Retrieved from https://burnseverity.cr.usgs.gov/baer/

U.S. Geological Survey. (2023). WFPI-based Large Fire Probability (WLFP). Retrieved from https://www.usgs.gov/fire-danger-forecast/wfpi-based-large-fire-probability-wlfp

Xiao, Y., Zhang, S., Yang, P., You, M., Huang, J. (2019). A two-stage flow-shop scheme for the multi-satellite observation and data-downlink scheduling problem considering weather uncertainties. Reliability Engineering System Safety, Vol. 188.