Prototyping an Onboard Scheduler for the Mars 2020 Rover

Gregg Rabideau and Ed Benowitz
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA
Gregg.R.Rabideau@jpl.nasa.gov, eddieb@jpl.nasa.gov

Abstract

Efficiently operating a rover on the surface of Mars is challenging. Two factors combine to make this job particularly difficult: 1) communication opportunities are limited, 2) certain aspects of rover performance are difficult to predict. With limited communications, the rover must be given instructions on what to do for one or more Martian days at a time. In addition, the duration of many rover activities can be hard to predict, which leads to unpredictable energy use. Traditionally, conservatism is used to keep the rover safe and healthy. This approach, however can lead to a measurable loss in rover productivity. To regain some of this productivity, the Mars 2020 mission is prototyping the use of onboard scheduling software. The primary objective of this software is to identify and utilize opportunities that arise when actual rover performance is more efficient than the original, conservative prediction.

Introduction

In this paper, we introduce our work on an onboard scheduler prototype for the Mars 2020 (M2020) rover. We begin with a mission background, and introduce terminology. We then introduce the onboard scheduler, and provide some driving requirements. We discuss the data provided by the ground that the scheduler uses as input. We discuss the timeline library code that is used as a foundation for the scheduler, and discuss its application to this mission. We then discuss our algorithmic approach to the scheduler. We describe our tool chain that assists with the prototyping work. Finally, related work is discussed. Our goal in this paper is to explore the unique scheduling needs of a Mars rover, and to show our current thinking as we continue our prototyping effort.

Mars 2020 Background

Terminology

We first introduce terminology that will be used throughout the paper. Internal to the M2020 team, we have named our prototype the Onboard Planner (OBP). Using AI community terminology, the problem addressed by OBP would be more accurately be described as *scheduling*.

A *sequence* is a file containing a list of time-ordered spacecraft commands. This was used on Mars Science Laboratory mission (MSL) [Grotzinger et al., 2012] and will be used on M2020. A sequence is run in its own task. M2020 will have multiple sequence engine tasks, each of which can run its own sequence.

The length of a solar day on Mars is about 24 hours and 40 minutes and is often called a *sol*. During the Martian night, and at other times, we shut down rovers to recharge. We call this a *sleep*.

Before using an actuator, we often need to heat up the actuator so that it is warm enough to safely use. We call this a *preheat*.

A *comm window* is fixed time interval during which the rover communicates with either the ground or an orbiter.

Handover is defined as the point in time at which we switch from one schedule file to the next. It is typically towards the end of a plan.

Mission Background

The Mars 2020 rover will not only seek signs of habitable conditions on Mars in the ancient past, but will also search for signs of past microbial life. Additional mission goals include characterizing the climate and geology of Mars. The rover includes a drill that can collect core samples from Martian rock and soil, and set the samples aside in a "cache" on the surface of Mars. A future mission could potentially

return these samples to Earth. The payload includes 7 science instruments.

The mission goes successively through 3 main phases, each of which uses a subset of the hardware. During the cruise phase, the spacecraft travels from Earth to Mars. During cruise, trajectory correction maneuvers are performed, refining the trajectory. As the spacecraft nears Mars, the cruise stage hardware is discarded. Next is the Entry Descent and Landing (EDL) phase, which lands the rover on Mars. During this phase, a combination of guided entry, a parachute, powered descent, and finally a sky crane is used to deliver the rover safely to the Martian surface. During the final phase, the surface phase, the Mars rover drives on the surface of Mars, performing scientific investigations with its suite of instruments. The onboard scheduler will only be used during the surface phase.

Further information about the M2020 mission is available at https://mars.nasa.gov/mars2020/.

Flight Computer

M2020 must operate with limited onboard computing resources, and OBP will be given only a fraction of these resources. M2020 uses a BAE Rad750 single board computer [Berger, 2001]. It contains 128 megabytes of volatile DRAM, and is run at 133 MHz. The flight software is able to access 4 gigabytes of NAND non-volatile memory on a separate card.

Flight Software

The M2020 flight software runs on the VxWorks™ operating system, and is written in C. The flight software is decomposed into a number of modules, each of which runs in its own VxWorks™ task. Each module typically communicates with another module via inter-process communication (IPC). IPC messages get put on a priority queue, and each priority can be individually enabled or disabled. In most scenarios, a client will send an IPC message, wait for a reply IPC message, and then proceed.

Heritage from MSL

M2020 has high heritage from the MSL mission. While science instruments are new, most of the hardware is inherited. In addition, significant portions of the M2020 flight software are inherited or modified from MSL flight software.

We can also leverage experience and lessons learned during the operations phase of MSL. Some of these lessons, along with new M2020 requirements, provide the impetus for M2020's onboard scheduler. On MSL, often the process of creating sequences could take over 9 hours for the operations team. M2020 seeks to get this time down to 5 hours.

Due to execution uncertainty, the MSL operations team had to insert margin for sequence duration estimates. This resulted in wasted time and energy onboard. Additionally, MSL could not take advantage of available onboard knowledge. If a sequence finished early or failed, time was often wasted. The details of these productivity challenges can be found in [Gaines et al., 2016].

Onboard Scheduler Overview

Goals

To increase the amount of time spent onboard performing useful activities, and to reduce the amount of time spent on the ground preparing spacecraft commands, the M2020 systems team requested creation of OBP. The hope is that this will improve overall mission performance.

A goal of OBP is to take advantage of onboard knowledge that is not available to the ground ops team in a timely manner. For example, if an activity finishes early, and more energy is available, this is immediately known onboard. Potentially, OBP can take advantage of this knowledge by adding in an extra science activity, filling in an otherwise unused gap in the schedule of activities.

A second goal is reducing the time spent by the operations team in creating commands and sequences. For example, M2020 requires that OBP generate wakeup and preheat activities onboard. This is in contrast to MSL, which required the operations team to create these activities on the ground.

Usage Overview

We now discuss the high-level data flow between the operations team and OBP. Like MSL, M2020 will typically create a set of activities for one to three Martian sols at a time. The ground operations team begins by creating a binary plan file containing all the potential activities that could run. We anticipate this to include on the order of a few hundred activities. The plan file also describes onboard resources that each activity uses, and well as any constraints on or between activities. Each activity is expected to specify on the order of ten resources and constraints. Our expectation is that ground tools generate plan files with higher level human inputs, and that much of the detail and data going into the plan file is generated automatically from drag and drop templates and ground data. We discuss the contents of the plan file in greater detail later in this paper.

Once the plan file is uplinked to the spacecraft, it can be started via command. From this point, OBP is responsible for deciding which activities from the plan file may run onboard. Running at a predetermined rate, OBP generates a valid schedule for the activities in the plan file. Activities scheduled to start before the time of the next OBP cycle are dispatched for execution.

To generate the schedule, the OBP does the following. First, OBP examines the current spacecraft state, in terms of data volume, completed activity status, energy, and others.

This state is used as a starting point and initial condition. Then, for each activity, OBP attempts to find a start time for the activity that does not violate any constraints. The constraints may involve onboard state or parallelism limits, for example. In looking for these valid start times, OBP must take into account the activity's effect on onboard resources. Once the start time is found, OBP's prediction for all resources is updated. In this way, OBP is creating a prediction, over time, of the values of various onboard resources. Energy and data volume are examples of these onboard resources.

Each activity is examined in priority order, and is either placed on the schedule or discarded for this OBP cycle. Any activities that are scheduled to start before the next cycle are sent off for execution at their scheduled start times.

The scheduling and execution cycle continues until the handover time, at which point a new plan file is autonomously loaded and started.

Mandatory vs Optional

The overall philosophy for constructing plan files is as follows. There is a certain subset of the activities that the ground fully expects to have executed. These are called mandatory activities. As the operations team constructs a plan file, they are responsible for ensuring that under nominal circumstances, there are at least enough resources onboard to run all mandatory activities. Several constraints do not apply to mandatory activities. For example, a mandatory activity is allowed to be scheduled even if it would violate the handover limits on energy and data volume. Mandatory activities are not aborted if they run past their duration. OBP still has discretion to move around mandatory activities so long as their constraints are met.

Optional activities are activities that are "nice to have" if there are additional resources available to run them onboard. OBP must not schedule an optional activity if it would prevent the execution of a mandatory activity. As OBP schedules optional activities, it must ensure that they do not violate constraints in this plan or constraints at handover. Optional activities are aborted if they exceed their duration.

Unique M2020 Aspects

There are several challenges to scheduling for a Mars rover, and for our particular mission as well. M2020, like MSL, has limited energy that must be estimated and managed. Both use sleep (shutting down) to use less power and let the batteries recharge. Both have a global constraint on the minimum state of charge, as well as a constraint on the state of charge at handover. As OBP is generating a schedule, it must take these constraints into account. When OBP schedules an activity, it will attempt to include autonomously generated awake activities as needed to perform the current candidate activity, but without violating energy-related constraints. M2020 will have an onboard software energy estimator that integrates various data from hardware to give OBP an approximate state of charge for the battery. This, along with the actual durations of executed activities, will allow OBP to schedule less sleep and more science when energy is available.

Recall that comm windows are time periods during which the rover communicates with the ground or with an orbiter. They are typically fixed in time and cannot be moved. As OBP generates a schedule of activities, it needs to be aware of the comm windows, and avoid creating schedules that have conflicts between comm windows and other activities.

Preheating is another special case for M2020. A given activity may require certain actuators to be at an allowable temperature before use. OBP is required to autonomously schedule preheats required for a given activity. The durations of the preheats and their associated energy usage depend on the ambient temperature on Mars, which depends on the local time of day. Early morning times on Mars, for example, will be colder and require more time to warmup the various devices. Consider an Activity A. The preheat duration depends upon A's start time. So, as OBP considers scheduling A at different times, the preheat activities shrink and grow in duration. The OBP will use a lookup table to determine the required preheat durations for the time at which the heating is requested. The OBP will also need to manage overlapping or redundant preheats while it creates schedules.

Data Provided from the Ground

Much of the data OBP needs to generate schedules is provided from the ground. This comes in the form of an uplinked file, which we call the plan file. At the top level, the plan file contains a number of plan-wide constraints and a number of activities. An activity is the item that is scheduled. Each activity has a type, as well as set of attributes.

Plan Attributes and Constraints

Recall that at handover, we switch from one plan file to the next. We are required to do this switch autonomously if requested, so the plan file contains the handover time, and the name of the next plan file to start at handover. The plan has a constraint on the state of battery charge at handover. This expresses the desire to have the OBP make a given amount of energy available towards the end of its schedule for use by activities in the next plan file. Additionally, there is a constraint on the maximum allowable data volume to collect between the start of the plan execution and the handover time. Handover constraints ensure optional activities in the current plan don't consume too many resources potentially needed by the subsequent plan.

Activity Types

There are several types of activities. The most common is the generic activity. The generic activity runs a sequence and has a fixed duration.

The expanding activity has a variable duration. The minimum duration of an expanding activity is specified by the ground. The OBP may grow the duration of an expanding activity if more resources are available. The typical use case for an expanding activity is a variable-length drive; the rover may drive longer if more energy is available.

"Single Selection Groups" are a disjunctive group of activities designated in the plan file. At most one of the activities in the single selection group is allowed to be scheduled. The purpose for this grouping is as follows: If the most resource intensive activity in a group does not fit in the schedule, run a less intensive activity instead.

Several activities are related to the rover's sleep cycle. Recall that the OBP can autonomously generate CPU wakeup and shutdown activities. The ground can override this behavior by inserting "stay awake" activities, which prevent autonomous shutdown activities in a given time interval. Similarly, the ground can force a sleep to occur at a given time with a manual sleep activity.

Activity Attributes and Constraints

Each activity has a number of attributes that OBP uses to estimate the activity's use of onboard resources. Each activity has a duration. Each activity can have multiple start time ranges. Associated with each start time range is an absolute cutoff time. If an activity runs past its cutoff time, it will be aborted.

The energy consumed by an activity is given as a rate. To obtain the energy consumed by an activity, we multiply the energy rate by the activity's duration. Similarly, the rate of data volume generation is provided as well. We can multiply this value by the activity duration to get the data volume generated by an activity.

Additional attributes include the maximum number of sequence engines an activity could use, as well as the peak power an activity could use. Activities may also indicate that they depend on certain thermal zones being within their allowable flight temperatures. This attribute is an indication to the OBP that it must autonomously generate preheating and maintenance heating for this activity.

Additionally, each activity contains a number of constraints that restrict when it can be scheduled. Each activity has a bit array, which we call "resource bits". If activities have the same resource bit set, they are not permitted to run in parallel. We allow the ground to define the meaning of each resource bit for their own purpose. This gives the ground a way to explicitly prevent given activities from running in parallel. For example, resource bits could be used to prevent a robot arm activity from being scheduled to run in parallel with a drive activity, or to prevent two high-CPU using activities from being scheduled to run at the same time.

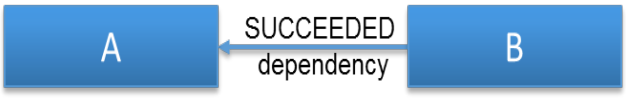

Figure 1: Dependency Constraint

In addition, the ground can create dependencies between activities (see Figure 1). Flight software knows if an activity succeeded or failed. We call this the activity's status. We can express that an activity cannot be scheduled unless a prior dependent activity has completed with an acceptable status. In the following example, activity B can only be scheduled to start after activity A has completed with a status indicating success. We have two options for the dependency constraint, selected with a Boolean "meets" attribute. If the "meets" attribute is true, B must be schedule to start at A's end time. If the "meets" attribute is false, B may be scheduled at any time after A's end time.

In addition, the ground can express a constraint on an activity that prevents the activity from running based on the value of a flight software variable. This is an area of design that we are in the process of maturing.

Timeline Library

To support activity scheduling, we have continued developed on a library for projecting values on timelines that was originally prototyped for the Europa flyby mission [Verma et al., 2017]. These timelines are used to predict the combined impact of activities on shared states and resources. Predicted values at future times are then used to identify potential constraint violations, or conflicts. With the ability to detect conflicts, we can then calculate the valid time intervals that are required to schedule activities. First, we describe the general types of timelines supported. Then, we provide more detail on our methods for computing valid start times. Finally, we give examples of timelines being used for M2020.

For simplicity, all types of timelines are implemented using the same data structures. All consist of a dynamic set of numeric constraints, impacts, and results. Timeline impacts and constraints are derived from the activity attributes and constraints, while results are computed from the impacts. A timeline impact is a change in the timeline value at a specific time. This can be an assignment to a value, an incremental change to the result value, or an incremental change to the rate. A timeline constraint is a set of minimum and maximum limits on the result value over a period of time. A timeline result is the value at a specific time, computed from the accumulation of impacts prior to and including that time.

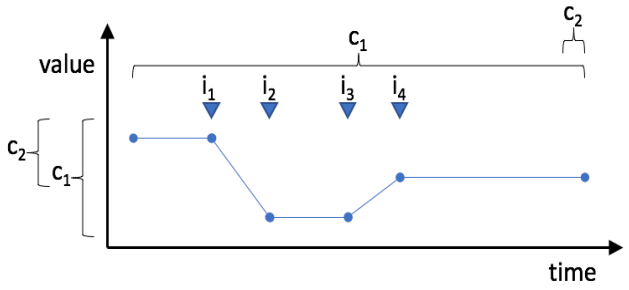

Figure 2: Timeline Representation

Figure 2 shows an example generic timeline with four rate impacts and two constraints. The timeline results are shown as dots falling under the impacts.

All timelines support the following functions:

- Add/remove impact
- Add/remove constraint
- Find constraint violations
- Find valid start times

When an impact is added to a timeline, a new result is created at that time, and results after it are updated. When a constraint is added, it is stored for use when finding constraint violations and valid start times. Constraint violations are found by simply looking for a result with a time that is within the constraint time range, but with a value that is not within the constraint limits.

For calculating valid start times of an activity, we use a simple method that involves temporarily adding the activity and checking for conflicts. This is done systematically across the timeline, and the conflict check is used to either include or exclude a time range from the valid start times. It is important to note that this does not need to be done for every possible time, but only where the timeline changes (i.e. at each result).

While all timelines are implemented the same, certain use cases arise that make it beneficial to define types and verify the use of each type. The timeline library supports five types similar to those found in [Rabideau et al, 1999, Knight et al., 2000, Chien et al. 2012]:

- State
- Atomic
- Claimable
- Cumulative
- Cumulative rate

For state timelines, values are integers, and a global constraint is used to restrict the values to a discrete set of states. All state changers are assignment impacts, and state requirements are local constraints. Atomic timelines are used for non-parallel constraints. Values are integers, and all impacts are provided in pairs: an increase of one at the activity start, and a decrease of one at the activity end. A global constraint is used to restrict all timeline values to either one or zero, enforcing the non-parallelism. Claimable timelines are similar, but are initialized with a user-specified maximum limit, and allow activities to "claim" more than one value at a time. This allows for a limited amount of activity overlap, depending on the available resource. Cumulative and cumulative rate timelines are the most generic types, allowing incremental changes in either value or rate. A global constraint may be provided, but constraints can be added over any time period.

Mars 2020 Timelines

Mars 2020 uses the timeline library to represent a specific set of rover states and resources. They are:

- Activity status
- Asleep and awake states
- Number of sequence engines
- Peak power
- Battery state-of-charge (SOC)
- Delta data volume
- Resource bits

A state timeline is used to represent the status of each activity in the plan. An impact is added at the end of the activity to change the state to "SUCCEEDED". If another activity has a dependency on this state, it will add a constraint to the timeline at the start of the activity.

Another state timeline is to represent the awake and asleep periods for the rover CPU. Most activities will add constraints to this timeline that require the rover to be awake. Special activities will be used to power down the CPU when needed to recharge the rover battery. These will add impacts that change the awake state.

Each activity uses a sequence, and the number of sequence engines on the rover is limited. A claimable timeline is used to represent the number of engines in use. Each activity adds a pair of impacts, incrementing the timeline value by one for only the duration of the activity. A global constraint keeps engine use below the maximum.

Peak power is also simulated and restricted using a claimable timeline. In this case, timeline values are floating-point numbers calculated from the power impacts of the activities.

Battery state-of-charge (SOC) is an approximation of the amount of energy available in the rover battery, specified as a percentage. Predicted SOC is represented using a cumulative rate timeline. Each power consumer activity increases the rate of discharge by adding a timeline impact at the start, and another impact at the end to negate the rate change. Two constraints are added to the SOC timelines: one to enforce a minimum SOC, covering the entire planning period; another to enforce a different SOC at handover.

The data volume (DV) production limit is enforced using a global constraint on a cumulative rate timeline that simulates data volume produced by the activities. Any activity

that produces data will add a pair of impacts to change the DV rate over the time range of the activity.

Finally, any non-parallelism indicated with the "resource bits" in the activity is enforced with an atomic timeline. For efficiency, we use a single timeline with a special implementation that employs a bit array, but otherwise behaves the same as multiple atomic timelines.

Algorithm Approach

As one can see from our flight processor, we have limited computing resources both in terms of CPU and RAM. With this in mind, we chose a greedy algorithm for OBP. Each activity has a ground-assigned priority associated with it. Activities are scheduled in priority order, with all optional activities having priority lower than all mandatory activities. This ensure an optional activity cannot consume resources that may be needed for a mandatory activity.

We consciously kept requirements loose to allow a greedy algorithm. There are no requirements to produce an optimal schedule or to optimize the schedule in any way. We are not required to schedule a maximal number of activities, nor are we required to schedule every mandatory activity. OBP must place activities on the schedule in priority order, and must only place it on the schedule if doing so does not violate any constraint. Given this flexibility, OBP is free to place an activity anywhere as long as constraints are not violated.

The greedy algorithm first sorts activities in priority order. Each activity is examined in that order. A running set of valid intervals is initialized with the activity's valid start time ranges. We then examine each timeline. Using the timeline library, we determine the valid start times available for an activity on a given timeline. We repeat this process of determining valid intervals for each timeline, intersecting the valid intervals as we go. At the end, we have a final set of valid intervals in which the activity can start. Next, we chose the earliest time in the valid interval and place the activity on the schedule, solidifying its impacts on all the timelines. In pseudocode, this is approximately:

```
For each activity
  For each timeline
    Find valid start times
    Intersect valid start times
  If a valid start time exists
    Add activity to plan
    Add impacts and constraints
```

If N is the number of activities and T is the number of timelines, the worst-case complexity for the overall algorithm is $O(TN^3)$. The biggest contributor factor is our simple calculation of valid start times for cumulative timelines. When scheduling the last activity, the timeline could have up to 2(N-1) results from the N-1 previously placed activities. Adding the activity could change 2(N-1) results, at the first time, 2(N-2) at the second, etc. The resulting complexity for cumulative timelines is N*(N-1)/2 or $O(N^2)$. The other timeline types have impacts that only change results that fall under the activity. For these types, valid start times can be computed in linear time. And because most timelines on M2020 are not cumulative, we would expect the average-case complexity for the full algorithm to be closer to $\Theta(TN^2)$.

There is no backtracking in a given OBP cycle; once an activity is added to the schedule, it does not move. However, on the next OBP cycle, scheduling can start from scratch. Any activity that was not dispatched for execution can be reevaluated and scheduled at a new location that is better suited for the latest spacecraft state.

Recall that expanding activities have a minimum duration, but can expand in duration as onboard resources allow. We schedule an expanding activity by doing a binary search on its duration.

If our initial OBP prototype is not meeting performance requirements, we may consider more sophisticated algorithms for computing valid start times [Knight et al., 2000] or limited forms of rescheduling. We may also consider implementing the greedy scheduling algorithm as an anytime algorithm, sacrificing only the lowest priority activities when time expires. Performance requirements are still in development.

Auto-generating Awake Activities

Most rover activities will use some power, and most require the CPU to be powered-on which also uses power. Similar to MSL, M2020 rover power will be provided by a battery that is continuously recharged by a radioisotope thermoelectric generator (RTG). When the CPU is on, the rover drains energy from the battery faster than the RTG can recharge it. Putting the rover in a "sleep" mode with the CPU off will allow the battery to recharge. Activities will be scheduled as needed to power-up and shutdown the CPU and keep the battery state-of-charge (SOC) within the required limits. In this section, we present a rough sketch of the awake/asleep scheduling algorithm that will be prototyped in the OBP.

We start with the assumption that the CPU will be powered off until an "awake" activity is scheduled to perform a wakeup and shutdown. During scheduling, this assures us that the maximum amount of energy is being preserved for activities that have not been scheduled yet. Then, ignoring awake time and SOC constraints, we find the earliest time that the next activity can be scheduled. If an awake activity can be added or extended to accommodate the new generic activity without violating the SOC limits, the activity is scheduled at that time. Otherwise, we must either find an

existing awake time, or create a new awake activity along with the generic activity being scheduled. If an existing awake time cannot be found, we create a new awake activity that is just large enough to accommodate the new generic activity. While considering the energy requirements of both activities, we find the valid time intervals with respect to the SOC limits. These intervals are then intersected with the valid intervals found for all other constraints on the activity. If the resulting set of intervals is empty, the activity is rejected. Otherwise, the activity is scheduled at the earliest possible time.

Scheduling an Activity with Preheats

We do not model temperature directly as a timeline. This information comes from tables uploaded by the ground. These thermal relation tables tell us, given a time of day, estimates of temperature, preheat duration, and preheat energy consumption.

Recall that the preheat durations vary with time of day. Given a sufficiently small interval of time, however, we are guaranteed that the preheat duration is constant. So, we first divide time into smaller intervals under which preheat durations are constant. For each of these intervals, we try to find valid start ranges. We can then combine each smaller set of valid time intervals into one larger one to obtain the set of all available start times.

Because of their coupling, both the main activity and its associated preheats are scheduled together. Queries to the timeline library for valid intervals combine constraints and impacts from the main activity and all of its preheats. It is as if we combine the preheats and the main activity into one big activity that is scheduled all at once.

Pitfall Cases

We acknowledge that by choosing a greedy algorithm, there are cases where OBP misses an opportunity to schedule additional activities. For example, suppose that we have two activities A and B that cannot be scheduled concurrently, where activity A is higher priority than B. And suppose that A has no start time constraints, but B is constrained to have a start time t = 0. OBP will greedily schedule A at t = 0, then be unable to schedule B. An optimal planner would have scheduled both, with A starting after the end of B.

Flight vs Ground Responsibilities

Because of the greedy algorithm, in essence some of the algorithmic responsibilities now fall to the ground side. It is the ground that decides the order of activity scheduling, because the ground decides activity priority.

In assigning priority, the ground tools must consider the chain of dependencies between activities. For example, suppose activity B has a dependency constraint, requiring that A has completed before B is allowed to run. OBP must be

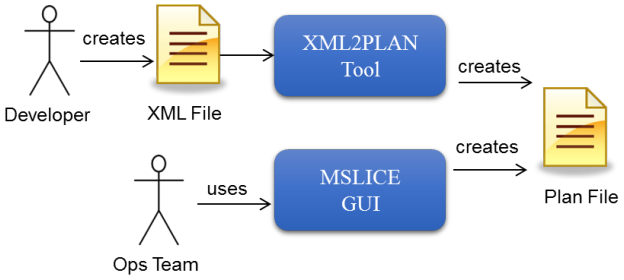

Figure 3: Plan File Creation

instructed to schedule B after A is scheduled. Otherwise, B would never be scheduled, as its dependency would never be met. Given this, the ground is required to assign priorities such that the priority of B is less than the priority of A. One approach is to perform a topological sort [Cormen, 2009] of the activities and their dependencies as an aid for assigning priorities.

The team debated whether to perform a topological sort onboard or on the ground. The issue with doing a topological sort onboard is that it can conflict with the ground's intent. By having the ground in complete control over the order of activity evaluation, we allow both topology and ground intent to be preserved and stay consistent with each other. The ground team is in the process of designing and prototyping their ideas for assigning priorities.

Prototype Tool Chain

We used a number of tools for use with the OBP prototype. As mentioned previously, the primary input used by OBP is the plan file. The plan file is a binary file created on the ground and uplinked to the spacecraft. We currently have two options for creating the plan file on the ground (see Figure 3). For the most precise control, the developer can create an XML file describing every detail of the plan file. A tool we call XML2PLAN converts the XML file into a binary plan file.

When the time for mission operations comes, we plan to have a user-friendly graphical application to assist with planning the Martian day. This application, called MSLICE, will graphically help the ground team describe activities and their relationships at a higher level. MSLICE also generates a binary plan file. Our prototype is able to ingest plan files from either XML2PLAN or MSLICE.

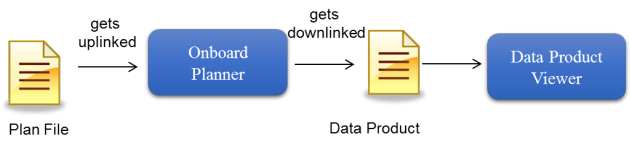

Figure 4: OBP File Inputs, File Outputs, and Data products

Once created, the plan file is then uplinked to the spacecraft (see Figure 4). A command is sent to the onboard planner to execute the plan described in the plan file. The planner can create files describing a history of its actions as it executes. We call files generated by the spacecraft Data Products. Data products can be downlinked from the spacecraft to the ground. Once a data product is down, we have several data product viewers. We have viewers that can either show the data product graphically or as text. Data products for the OBP can contain both predicted and actual start and end times for activities. A data product viewer displays this data in a Gantt chart style. We can also create a data product containing many internal data structures including timelines. We have a graphical data product viewer tool to examine this more detailed debugging data.

Related Work

Similar onboard scheduling software has been used on previous space missions. In the late 1990s, the first onboard planner was demonstrated for 48 hours on the Deep Space 1 spacecraft [Jónsson et al., 2000]. In the early 2000s, onboard planning was a central component in autonomy software that was used for 12 years as the primary control of the Earth Observing 1 spacecraft [Chien et al., 2005]. Starting in late 2013, onboard planning was used for one year of autonomous operations of the IPEX cubesat mission [Chien et al., 2016].

In addition, numerous systems have been developed that demonstrate the benefits of autonomy for Mars rovers using Earth-based analogues [Simmons and Apfelbaum, 1998] [Pedersen et al., 2003] [Estlin et al., 2007] [Woods et al., 2013] [Wettergreen et al., 2014]. Ongoing research continues to look at making future rovers more autonomous and more productive [Diaz et al., 2013] [Gaines et al., 2017]. For M2020, we are not developing a particularly sophisticated planning algorithm, or a new autonomy architecture. Instead, we are adding one new component to a heritage flight software system to address known productivity issues with rover operations, in a manner that is relatively predictable to the operators.

Mars rovers have seen an increase in autonomous capabilities over the years [Bajracharva et al., 2008]. Onboard path planning has been used to support autonomous navigation [Carsten et al., 2007]. The AEGIS system autonomously selects and sequences targeted science activities based on image analysis [Estlin et al., 2012]. The OBP is being prototyped to provide system-level coordination of M2020 rover activities, which would include activities to initiate existing autonomous behaviors.

Similar onboard planning capabilities were prototyped as the Multi-mission Executive (MEXEC) for the Europa flyby mission [Verma et al., 2017]. For Europa, the focus was on how these capabilities could be used to reestablish a science plan after a flight processor reset induced by the harsh radiation environment near Jupiter. For M2020, the focus is on utilizing opportunities created when resources are used more efficiently than conservative estimates. However, both scenarios require basic re-planning capabilities. In fact, much of the code for the M2020 timeline library was reused from the MEXEC prototype.

Finally, our problem of computing valid start time intervals is similar to the one presented in [Knight et al., 2000]. Our solution, however, uses a simpler, but more computationally expensive, test-and-check method. The overall approach for the OBP prototype is to start with simple implementations, evaluate performance, and consider more sophisticated algorithms only when required to fit within the limited M2020 computing resources.

Future Work

There are many areas we need to mature on the road from prototyping to the complete M2020 implementation. We are currently working towards a prototype implementation of the autonomous awake generation and autonomous preheat generation.

Much of this paper has dealt with discussion of the scheduling itself. Generating a schedule is only part of the problem. The schedule is generated periodically at a lower rate, but sequences within the schedule need to be started at the appropriate time. Just before we can start a sequence, we need to check its constraints. This is because the onboard state of the constraints may have changed between the time the schedule was generated and the time at which we want to start the sequence. Additionally, there is an onboard monitoring aspect that needs to run at a higher rate than the scheduler. We need to check for activities that overrun their cutoff time, and abort them. We expect that the timely constraint checks, sequence starts, monitoring and aborting will take place in a separate thread running at a higher rate. We have a preliminary design for this, but have not implemented it yet in the prototype.

Additionally, we have requirements to pause and resume activities. This is an area requiring further design refinement, and we have not yet prototyped this feature.

Conclusion

For a rover with limited communication, using conservative estimates of resource consumption can provide safe, but often inefficient operation. The Mars 2020 mission is prototyping one method for regaining efficiency while maintaining safe operations. We have shown how onboard scheduling software can be used to monitor rover activities, update predicted resource availability, and schedule more activities

when possible. As with previous rover missions, the M2020 ground operations team will be able to provide a baseline set of conservative activities that are expected to fit. We have shown how OBP can enable the team to also provide a set of optional activities that keep the rover busy when resources are available.

Acknowledgements

We had many fruitful discussions within the M2020 flight and ground teams and with the greater JPL community. We thank Steve Chien, Stephen Kuhn, Elyse Fosse, Glenn Reeves, Steve Scandore, Dan Gaines, Vandi Verma, Marcel Schoppers, Colette Lohr, Jim Kurien, and Corey Harmon. Thanks to Chet Joswig for creating some useful ground tools.

The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract from the National Aeronautics and Space Administration.

References

Bajracharya, M., Maimone, M.W. and Helmick, D., 2008. Autonomy for mars rovers: Past, present, and future. *Computer*, *41*(12).

Berger, R.W., Bayles, D., Brown, R., Doyle, S., Kazemzadeh, A., Knowles, K., Moser, D., Rodgers, J., Saari, B., Stanley, D. and Grant, B. 2001. The RAD750™ - a radiation hardened PowerPC™ processor for high performance spaceborne applications. In *Aerospace Conference, 2001, IEEE Proceedings*. (Vol. 5, pp. 2263-2272). IEEE.

Carsten, J., Rankin, A., Ferguson, D. and Stentz, A., 2007, March. Global path planning on board the mars exploration rovers. In *Aerospace Conference, 2007 IEEE* (pp. 1-11). IEEE.

Chien, S., Doubleday, J., Thompson, D.R., Wagstaff, K.L., Bellardo, J., Francis, C., Baumgarten, E., Williams, A., Yee, E., Stanton, E. and Piug-Suari, J., 2016. Onboard Autonomy on the Intelligent Payload EXperiment CubeSat Mission. *Journal of Aerospace Information Systems*, pp.1-9.

Chien, S.A., Knight, R., Stechert, A., Sherwood, R., and Rabideau, G., Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling, In *Proceedings of the Fifth Int'l Conference on AI Planning and Scheduling*, Breckenridge, CO, April 2000.

Chien, S.A., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davies, A., Mandl, D., Frye, S., Trout, B., Shulman, S., Boyer, D., Using Autonomy Flight Software to Improve Science Return on Earth Observing One, *Journal Aerospace Computing, Information, & Communication*, April 2005, AIAA.

Chien, S.A., Tran, D., Rabideau, G., Schaffer, S.R., Mandl, D. and Frye, S., 2010, May. Timeline-Based Space Operations Scheduling with External Constraints. In *ICAPS* (pp. 34-41).

Cormen, T. 2009. *Introduction to Algorithms*. MIT press.

Diaz, D., Cesta, A., Oddi, A., Rasconi, R. and R-Moreno, M.D., 2013. Efficient energy management for autonomous control in rover missions. IEEE Computational Intelligence Magazine, 8(4), pp.12-24.

Estlin, T.A., Bornstein, B.J., Gaines, D.M., Anderson, R.C., Thompson, D.R., Burl, M., Castaño, R. and Judd, M., 2012. Aegis automated science targeting for the mer opportunity rover. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *3*(3), p.50.

Estlin, T., Gaines, D., Chouinard, C., Castano, R., Bornstein, B., Judd, M., Nesnas, I. and Anderson, R., 2007, April. Increased Mars rover autonomy using AI planning, scheduling and execution. In *IEEE International Conference on Robotics and Automation,* (pp. 4911-4918). IEEE.

Gaines, D., Doran, G., Justice, H., Rabideau, G., Schaffer, S., Verma, V., Wagstaff, K., Vasavada, V., Huffman, W., Anderson, R., Mackey, R., Estlin, T., Productivity challenges for Mars rover operations: A case study of Mars Science Laboratory operations, Technical Report D-97908, Jet Propulsion Laboratory, January 2016.

Gaines, D., Rabideau, G., Doran, G., Schaffer, S., Wong, V., Vasavada, A., Anderson, R., Expressing Campaign Intent to Increase Productivity of Planetary Exploration Rovers, ICAPS Workshop on Planning and Robotics, 2017 (under review).

Grotzinger, J. et al. 2012. Mars Science Laboratory mission and science investigation. *Space science reviews* 170, no. 1-4, pp. 5-56.

Jónsson, A.K., Morris, P.H., Muscettola, N., Rajan, K. and Smith, B.D., 2000, April. Planning in Interplanetary Space: Theory and Practice. In *AIPS* (pp. 177-186).

Knight, R., Rabideau, G. and Chien, S.A., 2000, April. Computing Valid Intervals for Collections of Activities with Shared States and Resources. In *AIPS* (pp. 339-346).

Pedersen, L., Bualat, M., Lees, D., Smith, D.E., Korsmeyer, D. and Washington, R., 2003. Integrated demonstration of instrument placement, robust execution and contingent planning.

Rabideau, G., Knight, R., Chien, S., Fukunaga, A. and Govindjee, A., 1999, August. Iterative repair planning for spacecraft operations using the ASPEN system. In *Artificial Intelligence, Robotics and Automation in Space* (Vol. 440, p. 99).

R. Simmons and D. Apfelbaum, "A Task Description Language for Robot Control," Proceedings of the Intelligent Robots and Systems Conference, Vancouver, CA, October 1998.

Verma, V., Gaines, D., Rabideau, G., Schaffer, S., Joshi, R., Autonomous Science Restart for the Europa Mission with Lightweight Planning and Execution, *International Workshop on Planning and Scheduling for Space*, 2017 (under review).

Wettergreen, D., Foil, G., Furlong, M. and Thompson, D.R., 2014. Science autonomy for rover subsurface exploration of the Atacama Desert. *AI Magazine*, *35*(4), pp.47-60.

Woods, M., Shaw, A., Tidey, E., Van Pham, B., Simon, L., Mukherji, R., Maddison, B., Cross, G., Kisdi, A., Tubby, W. and Visentin, G., 2014. Seeker—Autonomous Long-range Rover Navigation for Remote Exploration. *Journal of Field Robotics*, *31*(6), pp.940-968.