Using Explainable Scheduling for the Mars 2020 Rover Mission

Abstract

Understanding the reasoning behind the behavior of an automated scheduling system is essential to ensure that it will be trusted and consequently used to its full capabilities in critical applications. In cases where a scheduler schedules activities in an invalid location, it is usually easy for the user to infer the missing constraint by inspecting the schedule with the invalid activity to determine the missing constraint. If a scheduler fails to schedule activities because constraints could not be satisfied, determining the cause can be more problematic. In such cases it is important to understand which constraints caused the activities to fail to be scheduled and how to alter constraints to achieve the desired schedule. In this paper, we describe such a scheduling system for NASA's next planetary mission, the Mars 2020 Perseverance Rover, as well as Crosscheck, an explainable scheduling tool that explains the scheduler behavior. The scheduling system and Crosscheck are the baseline for operational use to schedule activities for the Mars 2020 rover. As we describe, the scheduler generates a schedule given a set of activities and their constraints and Crosscheck (1) provides a visual representation of the generated schedule (2) analyzes and explains why activities failed to schedule given the constraints provided; and (3) provides guidance on potential constraint relaxations to enable the activities to schedule in future scheduler runs.

Introduction

In order for an automated scheduling system to be used for highly critical applications, such as space missions, the users must both trust the system and understand its behavior. One such application is the Mars 2020 (M2020) Perseverance Rover, which is scheduled for a launch no earlier than July 30, 2020, with an expected landing date on Mars of February 18, 2021 (Jet Propulsion Laboratory 2020b). The mission will use a ground-based scheduler, called Copilot, with plans to transition to using an onboard scheduler later on in the mission. Both the ground scheduler and the onboard scheduler use the same core scheduling algorithm (Rabideau and Benowitz 2017) (Chi et al. 2018) (Agrawal et al. 2019) (Chi, Chien, and Agrawal 2020).

Such automated scheduling systems have not been used for prior NASA planetary rover missions (with the notable exception of the Mars Exploration Rover Mission (Bresina et al. 2005)). Instead, science team members created the activities for the rover and science planners had to manually generate a schedule consisting of the activities and ensure that the schedule satisfied all given constraints.

This process resulted in a significant amount of time being spent to create the schedule. In addition, the activity durations provided were very conservative. Data from the Mars Science Laboratory Mission (Gaines et al. 2016a; 2016b) indicates that activities completed with durations on average 28% smaller than predicted. The ground scheduler primarily aims to significantly reduce the time spent to generate a correct schedule. Since the onboard planner will be able to account for activities taking more or less time than expected (Chi et al. 2018), the onboard planner will allow the science team to be less conservative when determining activity durations, resulting in higher mission productivity.

Using a scheduler that can generate a schedule which will satisfy the constraints provided by the science team will save much needed time in the process of creating a schedule for the rover, as well as increase mission productivity. In order to build a high level of confidence and trust in such a system, it is essential to have a tool that can easily explain why the final schedule looks the way it does, particularly when a critical activity was not able to be scheduled due to the constraints provided.

Crosscheck is an explainable scheduling tool that provides a visual representation of the generated schedule and explains exactly why certain activities failed to schedule. By using Crosscheck, users will not only gain confidence in the scheduling system but they will also understand how they can alter constraints to achieve the desired schedule.

The remainder of the paper is organized as follows. First, we describe the scheduling problem and what types of constraints the scheduler must satisfy. We also discuss how the scheduler auto-generates and schedules certain activities that are required by the activities given by the science planners. These auto-generated activities ensure the rover is sufficiently heated to perform activities, determines when the CPU must be on to perform activities, and when it can be turned off to preserve energy. Second, we describe the algorithms used in Crosscheck to identify why a given activity

failed to be schedule and how the constraints may be altered to achieve the desired schedule. Third we provide visual examples of what kinds of failure causes Crosscheck is able to identify. Finally, we discuss related work in this field as well as areas for future work.

Problem Definition and Scheduling Constraints

The target scheduler is a non-backtracking scheduler to be onboard the M2020 rover (Rabideau and Benowitz 2017) that schedules in priority first order and never removes or moves an activity after it is placed during a single run of the scheduler. The priorities of each activity are determined using a squeaky wheel approach described in (Chi et al. 2019). For the scheduling problem we adopt the definitions in (Rabideau and Benowitz 2017). The scheduler is given a list A of activities with their constraints, $A_1\langle p_1, d_1, R_1, e_1, dv_1, \Gamma_1, T_1, D_1 \rangle \ldots$
$A_n\langle p_n, d_n, R_n, e_n, dv_n, \Gamma_n, T_n, D_n \rangle$ where

- p_i is the scheduling priority of activity A_i. A higher numerical value for p_i indicates a lower scheduling priority and vice versa.

- d_i is the nominal, or predicted, duration of activity A_i.

- R_i is the set of unit resources $R_{i_1} \ldots R_{i_m}$ that activity A_i will use. Two activities cannot be scheduled in parallel if they use the same unit resource. For example, if two activities both claim the Arm resource, then they cannot be scheduled in parallel.

- e_i and dv_i are the average rates at which the consumable resources energy and data volume respectively are consumed by activity A_i.

- $peak_i$ is the maximum rate at which the activity will consume energy at any point.

- $\Gamma_{i_1} \ldots \Gamma_{i_r}$ are non-depletable resources used such as sequence engines available or peak power for activity A_i.

- T_i is a set of allowed start time windows $[T_{i_{j\_start}}, T_{i_{j\_preferred}}, T_{i_{j\_end}}] \ldots [T_{i_{k\_start}}, T_{i_{k\_preferred}}, T_{i_{k\_end}}]$ for activity A_i. Activity A_i must be scheduled within one of its allowed start time windows. [1].

- D_i is a set of activity dependency constraints for activity A_i where $A_p \rightarrow A_q$ means A_q must execute successfully before A_p starts.

- SR_i is a set of activity state constraints or requirements for activity A_i where a certain rover state must be satisfied before activity A_i can execute. The rover state may be satisfied either by a previous activity that sets the state to the necessary value or it may be satisfied if the incoming rover state is the required value.

- SE_i is a set of activity state effects for activity A_i where a certain rover state is set to a value by activity A_i. This rover state may be required by another activity in order for it to execute.

---

[1] If a preferred start time, $T_{i_{j\_preferred}}$ is not specified for window j then it is by default $T_{i_{j\_start}}$

- U_i is a set of UHF interactions for activity A_i. A UHF interaction determines which kinds of specific data uplink and downlink activities A_i are allowed to be scheduled in parallel with.

Each activity may also require the automatic generation of: 1) a set of preheat activities, $P_i = \{p_{i_1} \ldots p_{i_k}\}$, 2) a set of maintenance heating activities, $M_i = \{m_{i_1} \ldots m_{i_k}\}$, or 3) an awake activity, a_i. Preheats are setup activities (i.e. they occur before the activity), while maintenance heating and awakes are companion activities (i.e. they occur during or with the activity).

In addition to satisfying the constraints listed above for each individual activity, the scheduler must also ensure that scheduling an activity will not violate certain global constraints. These global constraints include the following.

- *Minimum State of Charge (SOC) (C_{soc}^{min})* The state of charge, or energy value, cannot dip below the Minimum SOC at any point. If scheduling an activity would cause the energy value to dip below the Minimum SOC, then that activity will not be scheduled.

- *Maximum Peak Power ($C_{p\_power}^{max}$)* If scheduling the activity would result in the total peak power at a given time point to exceed the Maximum Peak Power, then the activity will not be scheduled.

Scheduling Wake Sleep Activities

The M2020 rover's power source is a Multi-Mission Radioisotope Thermoelectric Generator (MMRTG) (Jet Propulsion Laboratory 2020a). While the MMRTG constantly generates energy, the CPU's awake and "idle" state (i.e. no other activities) consumes more energy than the MMRTG provides. Therefore, the rover's energy, or battery SOC, only increases when it is asleep. The rover, however, must be awake to execute certain activities. The scheduler is responsible for generating and scheduling these periods when the rover must wakeup, remain awake, shut down, and then sleep as shown in Figure 1. While scheduling these awake and asleep periods, the scheduler must consider the following types of constraints to prevent the rover from waking up and shutting down too frequently.

- *Minimum Sleep Time(C_{sleep}^{min})* Any given asleep period must be at least a certain duration.

- *Minimum Awake Time (C_{awake}^{min})* Any given awake period must be at least a certain duration.

More details about how these awake and asleep periods are generated can be found in (Chi, Chien, and Agrawal 2020).

Figure 1: The scheduler must schedule periods when the rover will be awake and asleep while satisfying the minimum awake/asleep requirements.

Scheduling Heating Activities

Certain activities may require an instrument on the rover to be sufficiently heated before the activity can execute and this heating must be maintained for the duration of the activity. For a given activity, A_i, the scheduler must autogenerate and schedule these heating activities called preheats, P_i which must complete prior to the start time of an activity, and maintenances, M_i, which occur during the activity, as shown in Figure 2. If the required preheats and maintenances are not able to be scheduled, then the activity will also not be able to be scheduled.

These autogenerated preheats and maintenances consume energy just as other activities do. The duration of the preheat is dependent on the time of day. During times when the temperature is colder, a longer preheat will be required than during times when the temperature is warmer. If the preheat is too long and consumes too much energy then it is possible it will not be able to scheduled due to violation of C^{min}_{soc}. Preheats also have defined *operability windows*. If the activity is scheduled such that its preheat would need to start outside of its operability window, then the preheat and activity will not be able to scheduled.
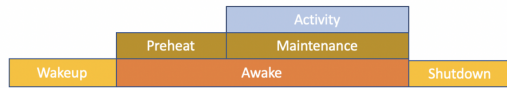

Figure 2: The scheduler must schedule the required heating for a given activity as well.

If any of the above constraints are violated while the scheduler is attempting to schedule the activity or any of its required autogenerated awake, asleep, or heating activities then the activity will not be scheduled. Since all input activities coming from the user are deemed *mandatory*, if science planners see that an activity failed to schedule they will want to know why.

In the following sections, we discuss how Crosscheck analyzes the scheduling behavior described above to identify why an activity failed to schedule and how it presents this information to users in a comprehensible way.

Determining First Scheduling Step when Constraints Failed to be Satisfied

After the scheduler generates a schedule with the given activities and constraints, users will use Crosscheck to view and understand the resulting schedule. Crosscheck analyzes the behavior of the scheduler to provide additional information while each activity is being scheduled in order to identify which constraints caused the activity to fail to schedule.

The onboard scheduler that will be used further into the mission has extremely limited CPU resources available[2] and

---

[2]The RAD750 processor used by the Mars 2020 rover has measured performance in the 200-300 MIPS range. In comparison a 2016 Intel Core i7 measured over 300,000 MIPS or over 1000 times faster. Furthermore, the onboard scheduler is only allocated a portion of the computing cycles onboard the RAD750 resulting com-

a conservative estimated runtime of 60 seconds. Backtracking would result in a significantly higher runtime. Thus it does not backtrack. Since it is crucial that the ground scheduler, which will be used immediately after landing, sufficiently prepares the team for use of the onboard scheduler it is important that both schedulers use the same scheduling algorithm. Therefore, the ground scheduler also does not backtrack.

Since the scheduler does not backtrack while scheduling an activity, the order in which activities are scheduled, given by the scheduling priority, can have a significant impact on the final schedule.

Each time the scheduler attempts to schedule an activity, it performs a new scheduling step. Consequently, the total number of scheduling steps is equal to the number of activities that must be scheduled. We define a *failure step* of a given activity as a step during which the activity is scheduled which results in the activity failing to be scheduled. The goal is to determine the earliest failure step of an activity that failed to schedule given the initial input plan.

Changing constraints of activities with a scheduling priority that would result in them being scheduled after the first failure step will not change whether the activity is able to schedule. However, it is possible that once constraints change for activities scheduled at or before the first failure step that is identified, activities scheduled after the first failure step will contribute to the failure or success of the activity in question, and a subsequent viewing of the updated schedule in Crosscheck would reveal these.

Crosscheck finds the earliest failure step of a given activity by using a binary search method. Algorithm 1 describes how Crosscheck finds the earliest failure step for an activity and the resulting partial schedule.

If an activity A_f with scheduling priority p_f is scheduled at step p_f in the initial plan, then it is first considered for scheduling by Crosscheck at step $p_f/2 = p'_f$. All activities with priorities 0 through $p'_f - 1$ are included in the plan. If the activity A_f successfully schedules in this first partial input plan, then the first failure step is due to an activity that with a lower scheduling priority (higher numerical value for p_i). The process repeats, with the steps under consideration being between p'_f and p_f. If the activity A_f is not successfully scheduled, the process is repeated with the steps under consideration being between 1 and p'_f. This process is repeated until the earliest failure step is found.

Some constraints that A_f has may be dependent on other activities being present in the plan. These include dependency constraints and state requirements. A constraint that falls in either of these two categories is only included in the current partial input plan if all activities that could satisfy the constraint and having a higher scheduling priority are also present in the input plan.

Dependency constraints are tied to a single other activity. If that activity is in the plan, then the dependency constraint is in the plan. If an activity A_f has a dependency on activity A_i, then A_i will have a higher scheduling priority than A_f.

---

putation *several thousand times* slower than a typical laptop.

Algorithm 1 Binary Search to Find Failure Step

Require:
    A: List of activities with their individual constraints
    P: Plan wide constraints (e.g. available cumulative resources)
    p_f: priority of activity that failed to schedule
Ensure:
    $Partial_{sc}$: Partial schedule consisting of activities scheduled in the failure step
1: $low \leftarrow 1$
2: $high \leftarrow len(A)$
3: **while** $low \neq high$ **do**
4:     $Partial_{sc} \leftarrow \{\}$
5:     $Partial_{in} \leftarrow \{\}$
6:     $p_{curr} \leftarrow low + (high - low \div 2)$
7:     **for all** $A_i \in A$ **do**
8:         **if** $p_i \leq p_{curr}$ **then**
9:             $Partial_{in}.insert(A_i)$
10:        **end if**
11:     **end for** $Partial_{sc} \leftarrow run\_scheduler(Partial_{in})$
12:     **if** $A_i \notin Partial_{sc}$ **then**
13:         $high \leftarrow p_{curr}$
14:     **else**
15:         $low \leftarrow p_{curr} + 1$
16:     **end if**
17: **end while**

State requirements may be satisfied by the effect of one of many activities. A state requirement constraint is not included in a plan unless all the activities with a higher scheduling priority than that of a A_f that have a state effect that could satisfy the requirement are included in the plan. Consider the following example. Activity A_3 with scheduling priority p_3 may have a state requirement SR_3 that could be satisfied by either state effect SE_1 given by activity A_1 with priority p_1, or by SE_2 given by activity A_2 with priority p_2. Activities A_1 and A_2 have a higher scheduling priority than activity A_3. If requirement SR_3 is added to the partial input plan consisting of the activities A_1 and A_3 then A_3 may fail to schedule if the activity A_1 fails to schedule. It would be premature to determine step 1 as the earliest failure step, as the activity may be able to successfully schedule after A_2 is scheduled. Thus, the requirement is not be added to the plan unless both the activities A_1 and A_2 are present in the plan.

Determining which Constraints Failed to be Satisfied

Once we find the failure step of an activity that failed to schedule, we can analyze why it failed to schedule. The scheduling algorithm consists of two primary phases, and an activity can fail to schedule for various reasons in either of these phases. The two phases are described as follows.

1. We find *valid intervals* for each activity constraint. Similar to the valid intervals used in scheduling for the Rosetta Orbiter (Chien et al. 2015), we define a *valid interval* for a given constraint as a continuous time interval within the plan bounds where the constraint will be satisfied for the duration of the activity. The valid intervals for each constraint are intersected with each other to generate the ac-

tivity's *final valid intervals*. If the set of final valid intervals for an activity is empty, then the activity fails to schedule. Valid constraint intervals were also calculated while scheduling for the Rosetta Orbiter.

2. We account for plan wide energy constraints, wake sleep activities, and heating activities in order to determine the activity's final start time. If the algorithm fails to find a time within the final valid intervals that satisfies these additional requirements, then the activity fails to schedule.

Crosscheck determines the cause of an activity failing to schedule in either of the two phases of the scheduling algorithm.

Failure due to Constraint Valid Intervals

First, Crosscheck checks if an activity failed to schedule due to an empty set of final valid intervals. While each activity A_i is being scheduled, valid intervals are computed for the following constraints.

- Allowed start time windows/Execution (T_i)
- Dependencies (D_i)
- Unit Resources (R_i)
- State Requirements (SR_i)
- State Effects (SE_i)
- Data Volume (dv_i)
- UHF interactions (U_i)

The valid intervals for each of the above constraints are intersected together to find the final valid intervals for the activity. If the resulting set of valid intervals is empty, then the activity cannot be scheduled and further analysis must be conducted in order to find out which constraints were unable to be satisfied. Algorithm 2 describes the recursive algorithm Crosscheck uses in order to identify which constraints caused the activity to fail to schedule.

First, Crosscheck checks if each individual constraint has valid intervals. If they all have at least one valid interval, then the algorithm will perform every possible pairwise intersection between the valid intervals. If each pair of constraints has valid intervals, then perform every three-way intersection, and so on. Ultimately through this process, Crosscheck will find the minimal set or sets of constraints that did not result in valid intervals after being intersected together, and thus, were the source of the activity failing to schedule.

In order to resolve the issue, one or more of the constraints that caused the activity to fail to schedule would need to be altered or a prior activity constraint would need to be altered in order to change the valid intervals for the activity that failed to schedule.

Failure due to Plan-wide Constraints or Wake Sleep/Heater Scheduling

An activity A_i may fail to schedule to schedule for the following reasons during the second phase of scheduling the activity. Crosscheck adds detailed descriptions to inform the user of exactly which of the below constraints caused the activity prevented the activity from scheduling.

Algorithm 2 Find Failure Cause from Constraint Intervals

1: **function** FINDINVALIDINTERSECTIONS(VI_{prev}, VI_{list}, $depth$, dur, $prevC$, $failedC$)
2:     **for all** $ind \in VI_{list}.size() - 1$ **do**
3:         $VI_{curr} \leftarrow VI_{list}[ind]$
4:         $intersections \leftarrow intersect(VI_{prev}, VI_{curr}, dur)$
5:         **if** $intersections.size() = 0 \land depth = 0$ **then**
6:             $failedC.insert(prevC + \{curr\})$
7:         **else if** $depth > 0$ **then**
8:             FINDINVALIDINTERSECTIONS($intersections$, $VI_{list}[ind+1{:}]$, $depth-1$, dur, $prevC + \{curr\}$, $failedC$)
9:         **end if**
10:     **end for**
11: **end function**
Require:
    VI_{list}: List of valid intervals for each constraint. VI_c corresponds to the valid intervals for constraint c
    dur: Duration of activity
    $planBounds$: start time and end time of the plan
Ensure:
    $failedC$: Minimal list of constraint types whose valid intervals intersected together result in activity failing to schedule.
12: $depth \leftarrow 0$
13: **do**
14:     $VI_{prev} \leftarrow \{planBounds\}$
15:     $failedC \leftarrow \{\}$
16:     $prevC \leftarrow \{\}$
17:     FINDINVALIDINTERSECTIONS(VI_{prev}, VI_{list}, $depth$, dur, $prevC$, $failedC$)
18:     $depth \leftarrow depth + 1$
19: **while** $failedC \neq \{\}$
20: **return** $failedC$

- Insufficient energy to schedule the activity along with its required preheat (P_i), maintenance (M_i), and awake activities (a_i) without violating the minimum state of charge constraint, ($C\ _{soc}^{\ min}$).

- Peak power of activity along with its required preheat (P_i), maintenance (M_i), and awake activities (a_i) would violate the maximum allowed peak power constraint ($C\ _{p\text{-}power}^{\ max}$).

- Scheduling the required wake sleep activities would cause violation of the minimum required sleep time ($C\ _{sleep}^{\ min}$) or the minimum required awake time ($C\ _{awake}^{\ min}$).

- One or more of the required preheat activities P_i would be scheduled outside of its operability window.

- One or more of the required preheat activities P_i would be scheduled outside of the bounds of the plan.

At each time point considered for scheduling during this phase, one of the above reasons can be the cause of the failure, and this reason is output by the scheduler. As multiple time points may be considered for scheduling during this phase, there may be multiple reasons for the failure, and Crosscheck will indicate all of them.

In the following sections, we discuss how users will use Crosscheck to visualize the schedule, understand why activities failed to be scheduled, and which constraints need to be changed to achieve the desired schedule.

Using Crosscheck

Crosscheck enables the user to visualize the current state of the schedule at each scheduling step as well as information on why a specific activity failed to schedule. By knowing which constraints caused the activity to fail to schedule, users can understand how the constraints may be modified to enable scheduling of the activity.

The schedule information at any given scheduling step is shown as a series of the following different types of timelines:

- *Output schedule.* The activities in the output schedule have already been scheduling by the current scheduling step. That is, they are of higher priority (lower numerical value) than the activity being scheduled at the current step.

- *Yet to be scheduled activities.* These activities are of lower priority than the activity being scheduled at the current step. Consequently they have not yet been scheduled.

- *Failed to schedule activities.* These activities have failed to schedule by the current scheduling step.

- *Energy profile.* The energy profile shows the total energy usage up until the current scheduling step.

- *Peak Power profile.* The peak power profile shows the total peak power usage up until the current scheduling step.

In addition, users are able to view the following activity specific information by clicking on any activity in the timelines containing activities that successfully scheduled or failed to schedule:

- The partial schedule at the scheduling step during which the scheduler attempted to schedule the failed activity.

- The valid intervals for each constraint as well as the final valid interval after intersecting the valid intervals for each constraint.

For activities in the failed to schedule timeline, users can view detailed information on why the activity failed to schedule and which constraints need to be altered in order to allow the activity to schedule.

Figure 3 shows an example of a schedule shown in Crosscheck along with the various kinds of information described above.

In Figure 3 an activity failed to schedule because there were no valid intervals after intersection the activity's valid intervals for its in progress dependency and execution constraints. This means that in order for the activity to schedule, on of two changes to the constraints must occur- 1) The activity's execution constraints need to be extended so that it can satisfy the dependency constraint as well or 2)the constraints of the activity that the failed activity was dependent on need to change so that it is scheduled during the execution intervals of the activity.

Resource Usage

Failures relating to depletable resources, such as energy, may not have one single activity causing the failure. Rather,
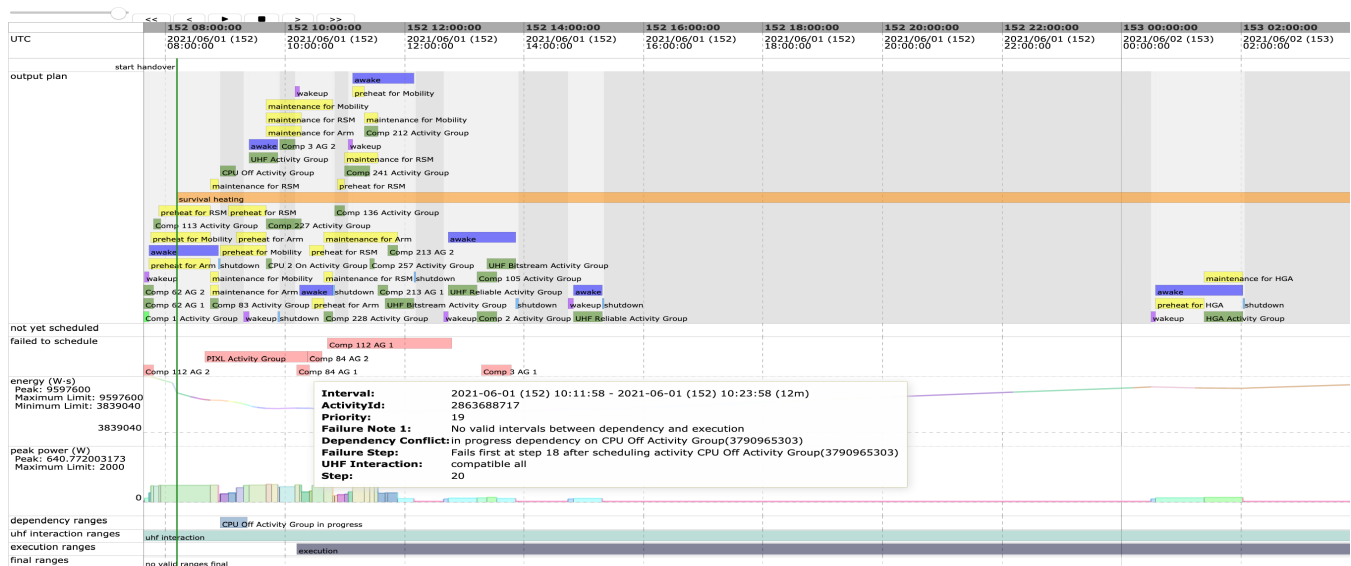
Figure 3: Crosscheck allows users to view the schedule and specific information on why an activity failed to schedule as well as which constraints may need to be adjusted.

it is the aggregate effects of multiple activities using the resource over separate times that would cause an activity to violate the minimum energy constraint.

To aid in science planners remedying failures related to energy violations, they are able to view each activity's energy use up to the current time, for any time. If they were expecting an activity to be scheduled at a particular time, they can view all the energy users up to that time, in decreasing order of their energy use. This will allow them to potentially change the constraints on prior activities to change their final start times, allowing more energy to be available for the failed activity to use at that time.

Similar information is available for peak power. As peak power is a non-depletable resource, it only matters which activities are using that resource at the current moment. Science planners can see which activities are using peak power at any time, as well as their peak power values, in decreasing order of the value.

Figure 4 shows how energy and peak power usage is visible in Crosscheck.

Crosscheck Examples of Identifying Scheduling Failures

Crosscheck will provide reasons for why an activity failed to schedule as well as guidance on how constraints need to be changed in order to schedule the activity.

If an activity fails to schedule in the first phase of the scheduling algorithm, (the set of final valid intervals is empty), then Crosscheck will provide detailed information about which constraints must be altered in order to allow the activity to schedule. This information can be used by the users to modify the input activity constraints so that the scheduler generates the desired schedule.

Figure 5 showcases several use cases of Crosscheck if an activity fails to schedule due to the activity having no final valid intervals. In Figure 5a, as the Failure Note in Crosscheck indicates, the activity fails to schedule because there is no valid intersection between the activity's execution valid intervals and UHF interaction valid intervals. Consequently, in order for the activity to schedule, either its UHF interaction must change so that it does not have to execute in parallel with a UHF activity (which is less realistically plausible for science planners to be able to change) or the allowed start time windows of the activity must change. This is more feasible. If the allowed start time window of the activity is widened so that the execution valid intervals will have more overlap with the UHF interaction valid intervals, then the activity may be successfully scheduled.

In Figure 5b, as indicated in Failure Notes 1 and 2, the activity fails to schedule because the activity's dependency valid intervals do not intersect with either the activity's execution valid intervals or its unit resource valid intervals. It is unrealistic for the activity's unit resources to be able to change, so either the dependency constraints or the execution constraints must be altered. While it is unlikely the science planner can remove the dependency from the activity that failed to schedule, it may be possible to alter the constraints of the parent activity that it has a dependency on. If the parent activity then is scheduled elsewhere in the plan, then the valid intervals of the child activity would also be altered. It may also be possible to alter the activity's execution constraints so that the activity has allowed final valid intervals.

In Figure 5c, as indicated in the note for the State Requirement Violation, the activity failed to schedule because the requirement that the rover must have its drill stowed was not satisfied by any prior activity. This means that either another activity must be added that is scheduled temporally
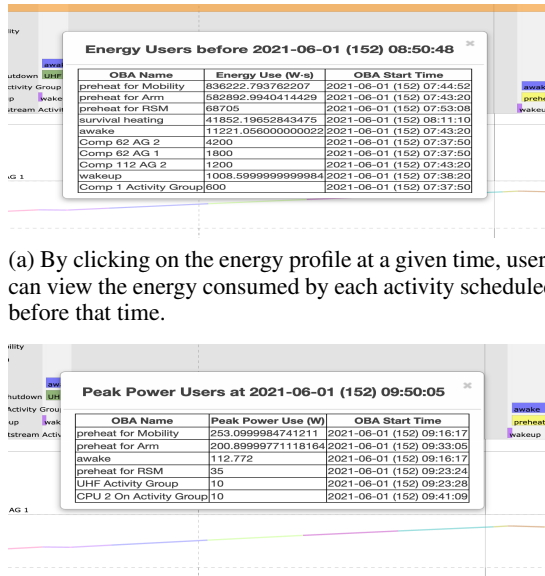
(a) By clicking on the energy profile at a given time, users can view the energy consumed by each activity scheduled before that time.

(b) By clicking on the peak power profile at a given time, users can view the peak power for each activity at that time.

Figure 4: Viewing energy and peak power usage in Crosscheck.

before this one so that the rover is in the required state or the incoming state of the rover must be the required state.

If the activity does have a nonempty set of final valid intervals, but fails to schedule during the second phase of the algorithm, then this means that the plan wide constraints (the minimum state of charge, the maximum peak power allowed, the minimum required sleep time, or the minimum required awake time) could not be satisfied while also accounting for the required heater and wake sleep activities.

In Figure 6a, the activity fails to schedule because scheduling it along with its required wake sleep and heater activities would result an a violation of the minimum state of charge constraint. After knowing this information by reading Failure Note 1 generated by Crosscheck, the user may want to know which activities scheduled temporally before the one that failed were the most energy consumptive. As shown in Figure 4a, the user can click on the energy timeline at a given time point and view the energy use of all activities occurring before that time point. For example, if the user clicked at a time on the energy timeline before the allowed execution valid interval of the activity, then they will be able to view the energy consumption of any activity that occurred before that time in order of most consumptive to least consumptive. Then, the user may be able to alter the constraints of one of the more consumptive activities so that it can be scheduled later on in the sol (a Martian day) when there is more energy available.

In Figure 6b, the activity fails to schedule because scheduling it along with its required wake sleep and heater activities would result an a violation of the maximum allowed peak power constraint, as indicated by Failure Note

(a) The activity cannot schedule because there is no valid time where the activity can satisfy both the dependency and the uhf interaction constraints.

(b) The activity cannot be scheduled because there is no valid intersection between the dependency valid intervals, unit resource valid intervals, and execution valid intervals.

(c) The activity cannot be scheduled because there is no valid time where the state requirement can be satisfied by a state effect of a prior activity.
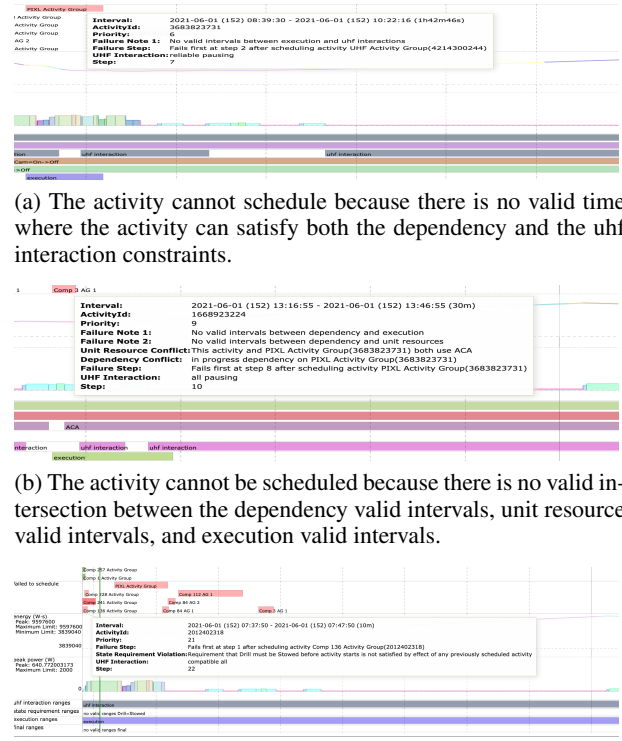
Figure 5: Activities Fail to Schedule due to No Final Valid Intervals

1. The user may want to know which activities were scheduled at the time the scheduler tried to schedule the activity and how much peak power those other activities used. As shown in 4b, the user can click on the peak power timeline at a given time point and view the peak power use of all activities occurring at that time point. If the user clicks at a time on the energy timeline during the allowed execution valid intervals, then they will be able to view the power consumption of any activity that occurs at that time in order of most consumptive to least consumptive. Then, the user may be able to alter the constraints of one of the more consumptive activities so that it can be scheduled later on in the sol when there is lower peak power usage.

In Figure 6c, the activity failed to schedule because the required preheat would be outside of its operability window. The science planner can clearly know this by viewing the failure note, and must alter the execution constraints of the activity so that the preheat would schedule within its operability window.

Related Work

Bresina et al. (Bresina et al. 2005) describe an automated planning and scheduling system called MAPGEN which helps users create plans for the Mars Exploration Rovers. MAPGEN was a constraint-posting planner, and it explic-
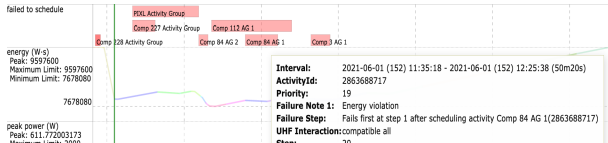
(a) An activity may fail to schedule because scheduling it would cause the energy to go below the minimum allowed energy level
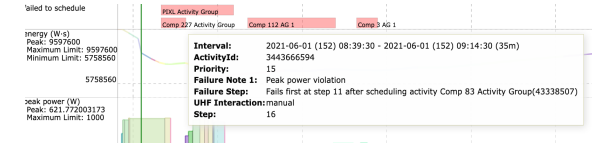
(b) An activity may fail to schedule because the maximum allowed peak power would be violated
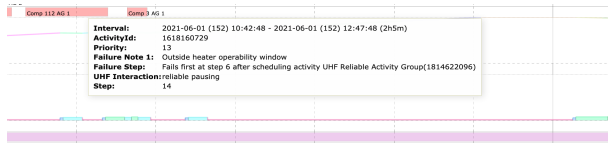
(c) An activity may fail to schedule because the required preheat would be outside of its operability window

Figure 6: Activities Fail to Schedule due to Plan Wide or Heater Specific Constraints

itly indicated temporal flexibility, allowing users to drag activities until they reached their earliest or latest allowed start times. Unlike Crosscheck, it does not explicitly indicate why the planner could not achieve the desired result.

Both (Chakraborti, Sreedharan, and Kambhampati) and (Fox, Long, and Magazzeni 2017) conduct a survey of important prior work in the field of explaining planning. They discuss the several important questions that arise when users interact with automated planning and scheduling systems as well as techniques that can help answer these questions. Crosscheck primarily focuses on answering why the planner did what it did given the activity constraints.

Ramaswamy et al. (Alper Ramaswamy et al. 2019) discuss a tool to visualize various execution runs of a given input plan using the Mars 2020 ground scheduler. This tool provides information on how often an activity failed to execute over all execution runs. It also indicates how often activities executed in parallel over all execution runs and whether they switched order temporally. Crosscheck focuses on visualizing a single schedule and provides specific information on why an activity failed to schedule given the constraints provided.

The Rosetta Orbiter (Chien et al. 2015) used automated scheduling to schedule science activities. A static display of the scheduling iterations is shown since ASPEN-RSSC also used a priority-based method to schedule activities, but it did not determine the higher priority iteration at which the activity would have been scheduled. The operators had to manually infer that information. Also the science campaigns only competed for the same resources; they did not set up

preconditions such as dependencies for each other, meaning that the problem came down to which campaign used the limiting resource.

Future Work

We aim to improve the efficiency of the current implementation that determines the set of constraint intervals that caused the activity to fail. In the current implementation, there are repeated intersection calculations as the depth increases. Storing intermediate interval calculations could aid in a faster runtime, but this has not been the significant bottleneck in the runtime of Crosscheck overall.

If the rover sleeps enough, the energy may reach the maximum allowed state of charge. If an activity fails to schedule due to violation of the minimum state of charge constraint after a time when the rover had maximum state of charge, any activities in the schedule prior to the time of maximum state of charge would not be at fault for the activity failing to schedule. When viewing resource users in Crosscheck, the energy users list contains all activities in the plan up to the time of interest, but not all of these activities are actually contributing to the energy value at that time. Future work includes showing only the relevant activities for an energy value at a given time.

Onboard the rover, activities may take more or less time than the predicted activity duration. It is important for the team to trust that the onboard scheduler will be able to handle these varying execution activity durations properly. Furthermore, if a certain set of execution durations results in activities failing to execute, then science planners may want to understand how they can change constraints so that their is a higher change the activities will execute even if prior activities take longer than expected. When the mission begins using the onboard planner, Crosscheck can be used to analyze a given execution trace, determine why activities failed to execute for that set of execution durations, and help science planners understand which constraints need to be altered to ensure the activity will execute despite uncertainty in execution.

Conclusion

In this paper we have described the automated scheduling system for the Mars 2020 Perseverance Rover mission as well as the explainable scheduling tool, Crosscheck, which will be used to explain the scheduling behavior to users in a comprehensible way that builds trust in the automated scheduling system. Crosscheck allows users to view the schedule at each scheduling iteration, gives detailed information on why an activity failed to schedule, and guides users on which constraints need to be altered in order for the activity to successfully schedule. This information will not only allow users to understand how they may need to change activity constraints to achieve the desired schedule, but it will build trust in the automated scheduling system as they become aware of why it behaves in the way that it does.

References

Agrawal, J.; Chi, W.; Chien, S.; Rabideau, G.; Kuhn, S.; and Gaines, D. 2019. Enabling limited resource-bounded disjunction in scheduling. In *11th International Workshop on Planning and Scheduling for Space (IWPSS 2019)*, 7–15. Also appears at the 29th International Conference on Automated Planning and Scheduling (ICAPS) Workshop PlanRob 2019 and ICAPS SPARK 2019 and ICAPS IntEx 2019.

Alper Ramaswamy, B. B.; Agrawal, J.; Chi, W.; Kim Castet, S. Y.; Davidoff, S.; and Chien, S. 2019. Supporting automation in spacecraft activity planning with simulation and visualization. In *AIAA Scitech 2019 Forum*, 2348.

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity planning for the mars exploration rovers. In *ICAPS*, 40–49.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. The emerging landscape of explainable automated planning & decision making.

Chi, W.; Chien, S.; Agrawal, J.; Rabideau, G.; Benowitz, E.; Gaines, D.; Fosse, E.; Kuhn, S.; and Biehl, J. 2018. Embedding a scheduler in execution for a planetary rover. In *International Conference on Automated Planning and Scheduling (ICAPS 2018)*. Also appears at International Symposium on Artificial Intelligence, Robotics, and Automation for Space (ISAIRAS 2018) as an abstract.

Chi, W.; Agrawal, J.; Chien, S.; Fosse, E.; and Guduri, U. 2019. Optimizing parameters for uncertain execution and rescheduling robustness. In *International Conference on Automated Planning and Scheduling (ICAPS 2019)*.

Chi, W.; Chien, S.; and Agrawal, J. 2020. Scheduling with complex consumptive resources for a planetary rover. In *International Conference on Automated Planning and Scheduling (ICAPS 2020)*.

Chien, S.; Rabideau, G.; Tran, D.; Troesch, M.; Doubleday, J.; Nespoli, F.; Ayucar, M. P.; Sitje, M. C.; Vallat, C.; Geiger, B.; Altobelli, N.; Fernandez, M.; Vallejo, F.; Andres, R.; and Kueppers, M. 2015. Activity-based scheduling of science campaigns for the rosetta orbiter. Invited Talk, in Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *arXiv preprint arXiv:1709.10256*.

Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; et al. 2016a. Productivity challenges for mars rover operations. In *Proceedings of 4th Workshop on Planning and Robotics (PlanRob)*, 115–125. London, UK.

Gaines, D.; Doran, G.; Justice, H.; Rabideau, G.; Schaffer, S.; Verma, V.; Wagstaff, K.; Vasavada, A.; Huffman, W.; Anderson, R.; et al. 2016b. Productivity challenges for mars rover operations: A case study of mars science laboratory operations. Technical report, Technical Report D-97908, Jet Propulsion Laboratory.

Jet Propulsion Laboratory. 2020a. Electrical power - nasa mars https://mars.nasa.gov/mars2020/spacecraft/rover/electrical-power/ retrieved 2020-07-24.

Jet Propulsion Laboratory. 2020b. Mars 2020 perseverance rover - nasa mars https://mars.nasa.gov/mars2020/ retrieved 2020-07-24.

Rabideau, G., and Benowitz, E. 2017. Prototyping an onboard scheduler for the mars 2020 rover. In *International Workshop on Planning and Scheduling for Space (IWPSS 2017)*.