Optimizing Parameters for Uncertain Execution and Rescheduling Robustness

Wayne Chi, Jagriti Agrawal, Steve Chien, Elyse Fosse, Usha Guduri
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

We describe use of Monte Carlo simulation to optimize schedule parameters for execution and rescheduling robustness in the face of execution uncertainties. We search in the activity input parameter space where a) the onboard scheduler is a one shot non-backtracking scheduler and b) the activity input priority determines the order in which activities are considered for placement in the schedule. We show that simulation driven search for activity parameters outperforms static priority assignment. Our approach can be viewed as using simulation feedback to determine problem specific heuristics e.g. Squeaky Wheel Optimization. These techniques are currently baselined for use in the ground operations of NASA's next planetary rover, the Mars 2020 rover.

Introduction

Embedded schedulers often must perform within very limited computational resources. We describe an approach for automatically deriving problem specific control knowledge for a one-shot (non-backtracking) scheduler. In this application, the onboard scheduler allows the rover to take advantage of run-time variations (e.g., execution durations) by rescheduling. Because the general structure of the schedule is known a priori on the ground before uplink, we use both analysis of the schedule dependencies and simulation feedback to derive problem specific control knowledge to improve the onboard scheduler performance.

We study this problem in the context of setting activity parameters—specifically activity priorities and preferred times—as part of the ground operations process for a scheduler (Rabideau and Benowitz 2017) designed to run onboard NASA's next planetary rover, the Mars 2020 (M2020) rover (Jet Propulsion Laboratory 2018). This scheduler is extremely computationally limited; even without backtracking a single scheduling run is estimated to take up to 60 seconds, making ground analysis to improve scheduler performance critical. Also, during operations onboard the rover, the scheduler will be re invoked in response to execution feedback - causing rescheduling approximately 15 times per sol. Thus, any ground analysis must take into account execution uncertainty and multiple scheduler re-invocations. For our problem, aside from the activity parameters, the onboard scheduler is treated as a "black box"; we can only set input activity parameters.

Since the scheduler does not backtrack across activity placements, both the order in which it considers activities and each activity's preferred start time heavily influence the resulting schedule quality. We therefore search the space of both a) activity priorities (which determine the order in which the scheduler considers activities) and b) preferred start times (which influence the start times for activity placement). At each step in the search, a Monte Carlo simulation is conducted to assess the likelihood of each activity being executed. Using an approach analogous to Squeaky Wheel Optimization (Joslin and Clements 1999), these Monte Carlo runs are automatically analyzed and provide feedback into activity priority and preferred time adjustment, affecting both the initial schedule generation and rescheduling. This search in the activity parameter space continues until all requested activities are expected to be included or a resource bound is exceeded. We call this approach *Parameter Search* and we present empirical results that show that *Parameter Search* outperforms several static priority assignment algorithms (those that do not use Monte Carlo feedback) including *manual expert derived priority setting*.

The remainder of the paper is organized as follows. First we describe our formulation of the scheduling problem, metrics for schedule quality, and the onboard scheduling algorithm. Second, we describe several static approaches to priority assignment as well as our *Parameter Search* approach that leverages Monte Carlo simulation feedback. Third, we describe empirical results demonstrating the efficacy of *Parameter Search* over static algorithms by evaluating on constrained variations of *sol types*, the best available anticipated operations plans for the M2020 planetary rover mission. Finally, we describe related and future work and conclusions.

Problem Definition

Rabideau and Benowitz 2017 define the M2020 scheduling problem as follows; the scheduler is given

- a list of activities

Copyright © 2019 California Institute of Technology. Government sponsorship acknowledged. All rights reserved.

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

$A_1\langle p_1, d_1, R_1, e_1, dv_1, \Gamma_1, T_1, D_1\rangle \ldots$
$A_n\langle p_n, d_n, R_n, e_n, dv_n, \Gamma_n, T_n, D_n\rangle$,

- where p_i is the scheduling priority of activity A_i,
- d_i is the nominal, or predicted, duration of A_i,
- R_i is the set of unit resources $R_{i_1} \ldots R_{i_m}$ that A_i will use,
- e_i and dv_i are the rates at which the consumable resources energy and data volume respectively are consumed by A_i,
- Γ_i is the set of non-depletable resources (e.g. sequence engines available, peak power) $\Gamma_{i_1} \ldots \Gamma_{i_l}$ that A_i will use,
- T_i is a set of start time windows $[T_{i_{1_start}}, T_{i_{1_preferred}}, T_{i_{1_end}}] \ldots [T_{i_{k_start}}, T_{i_{k_preferred}}, T_{i_{k_end}}]$ for A_i [1], and
- D_i is a set of activity dependency constraints for A_i where $A_p \rightarrow A_q$ means A_q must execute successfully before A_p starts.

We model execution uncertainty as variation in activity durations. The specific probabilistic model used is based on M2020 knowledge and described in our Empirical Analysis. We do not explicitly change other activity resources such as energy and data volume since they are generally modeled as rates and changing activity durations implicitly changes energy and data volume as well.

Scheduler Design The M2020 onboard scheduler is a non-backtracking scheduler that considers activities in priority-first order and never removes or moves an activity after it is placed during a single scheduler run. It does not search except when considering valid intervals for a single activity placement . Due to the greedy, non-backtracking nature of the onboard scheduler, both the order in which activities are scheduled and an activity's preferred start time can greatly impact the quality of the schedule.

Goal The goal of the scheduler is to schedule and execute all activities while respecting individual and plan-wide constraints. The goal of *Parameter Search* is to derive an input plan on the ground such that the scheduler will best be able to achieve its goal. We must derive the input plan in a time-sensitive manner to satisfy daily mission time constraints.

Evaluating a Schedule

The utility of a particular input parameter assignment is based on a) how *many* activities were successfully executed, and b) how *well* the activities were executed.

How *well* activities were executed is defined by a utility function that prefers leaving more energy for the next sol's plan (handover state of charge). As the most important aspect is still whether or not an activity is executed, the utility function follows an ordering of:

$$S_{exec}(A_i) \gg S_{SOC}(A) \tag{1}$$

where A is the set of all activities, $S_{exec}(A_i)$ is the utility if A_i was successfully executed, and $S_{SOC}(A)$ is the value of the plan's handover state of charge (SOC).

[1] If $T_{i_{k_preferred}}$ is not specified by the user then it is set to $T_{i_{k_start}}$

Handover State of Charge

The state of charge at the handover of every plan, is equivalent to the SOC that the next input plan will begin with. A higher SOC increases the utility of the plan. Where an activity is scheduled in a plan can affect how much preheating is needed for the activity (Figure 1), thereby affecting the amount of energy that is needed. Thus, in addition to being an overall indicator, the handover SOC acts as an indicator of how well each activity is scheduled.

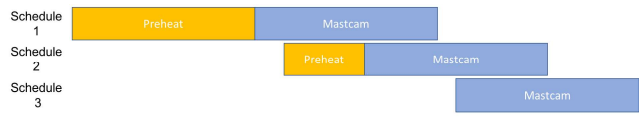


Figure 1: Preheat duration varies depending on an activity's scheduled start time. The Mastcam activity requires less preheat as the sol grows warmer near the middle of the day.

Static Algorithms for Activity Priority Assignment

Several static algorithms were developed to set activity priorities based on various activity ordering criteria. [2] Unlike Parameter Search, these algorithms do not consider Monte Carlo simulations of plan execution where activities may end early or late while determining priorities. These static algorithms will be compared to Parameter Search to gain a better understanding of how well Parameter Search performs. Activities which must begin at a particular time (e.g. data downlink) are always given the highest priority and thus are not affected by the static algorithms described.

The following three algorithms are used to initialize activity priorities:

- *Random Assignment* Each activity is given a random priority.
- *Latest Start Time* Activities with earlier latest start times are given higher priority.
- *Human Expert* Each activity is assigned a priority based on the start time of the activity in a schedule constructed by a human expert. The activity with the earliest start time in this schedule has the highest priority.

The following two methods are applied to activity priorities after they have been initialized by one of the static algorithms:

- *Tie Breaker* If activities have the same priority, the activity with the earliest latest allowed start time is higher priority. If they also have the same latest allowed start time then the longer activity has the higher priority. If all of these attributes are equal then the higher priority activity is chosen lexicographically based on each activity's unique identifier.
- *Dependencies* In the priority-first scheduling scheme, if $A_p \rightarrow A_q$ then A_q must be considered for scheduling

[2] The focus of the static algorithms was priority setting; modifying other schedule parameters is an area of future work.

before A_p; otherwise, A_p will never be scheduled. We ensure that A_q always has a higher priority than A_p through a simple recursive sort [3].

Parameter Search

To determine a set of parameters that will allow the scheduler to generate a schedule better than the static algorithms, we attempt to search the parameter space in an approach similar to Squeaky Wheel Optimization (SWO) as described in Joslin and Clements 1999. Squeaky Wheel Optimization usually involves a constructor, an analyzer, and a prioritizer. The constructor generates a schedule, the analyzer determines problem areas and assigns "blame" to certain elements in the schedule, and the prioritizer modifies the order in which the elements are considered. This process repeats until a satisfactory result is reached or allotted time runs out. However, there are two main differences between our algorithm and traditional SWO. First, we do not just consider the ordering of activities but also other parameters such as preferred time. Thus, our "prioritizer" should be described as a "parametizer" instead. Second, our scheduling problem is intrinsically tied to execution; analyzing the initial schedule generated by itself is not satisfactory. Our approach (Figure 2) builds upon the usual SWO approach by incorporating a simulation of execution and Monte Carlo to build an execution sensitive result. We call our approach Parameter Search as it searches the activity parameter space using Monte Carlo simulation feedback to find a good set of priorities, unlike the static algorithms.

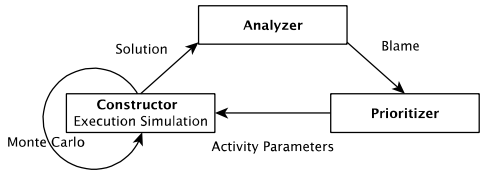


Figure 2: SWO accounting for execution.

Constructor

Typically, the constructor generates a schedule as the solution, which is then fed into the analyzer. However, our scheduling problem must be taken in context with execution. Activities may finish early or late, affecting how many and which activities can be scheduled. In order to take this into account, we generate the final schedule following a (lightweight) simulation of the entire plan execution. This is simulated by letting activities finish early or late by a variable amount based on a probabilistic model of plan execution; however, the probabilistic model may promote misleading results if only sampled once.

Our constructor runs a Monte Carlo and simulates multiple plan executions, passing on all of the executed plans as the solution to the analyzer. Each column in Figure 3 is a different simulation of execution and each row represents a reinvocation of the scheduler. The Monte Carlo simulation consists of multiple execution traces, each with different activity duration variants. The score is aggregated across all execution traces, and is fed to the analyzer.

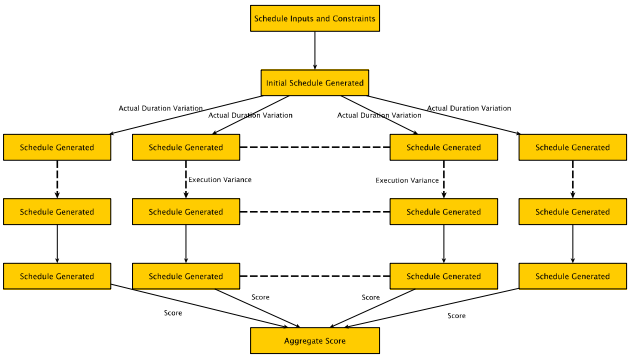


Figure 3: Monte Carlo simulation of multiple plan executions.

Runtime Analysis

Traditionally, the runtime of SWO is $\mathcal{O}(S \cdot (C + A + P))$, where $S = total_search_steps$, $C = constructor$, $A = analyzer$, and $P = prioritizer$. The constructor is typically a greedy algorithm (Joslin and Clements 1999); as such, $\mathcal{O}(C)$ is relatively small and on the order of $\mathcal{O}(N)$ where N is the number of activities. The runtimes of the analyzer and prioritizer are usually comparable to that of the constructor. However, Parameter Search's constructor is a Monte Carlo of multiple scheduler re-invocations.

In Parameter Search, $C = M \cdot s \cdot R \cdot c$ where M is the number of Monte Carlos per step, s is the runtime of the scheduler, R is the number of scheduler re-invocations per execution simulation, and c is a constant cost of simulating execution. It is possible to minimize the effects of M by parallelizing the Monte Carlos, but this is realistically limited by cost. Since the onboard scheduler's average case complexity is $\Theta(TN^2)$ (Rabideau and Benowitz 2017) [4], the constructor's runtime dominates the analyzer and prioritizer (parametizer in our case). The resulting runtime is

$$\mathcal{O}(S \cdot C) \tag{2}$$

The constructor's significantly increased runtime poses a notable challenge as our constructor can take minutes to generate a solution, in comparison to the seconds or milliseconds a traditional constructor would take. This limits the number of steps that Parameter Search can take before running out of time.

Analyzer

The analyzer (Algorithm 1) takes the solution and assigns blame to problem areas. The objective is to execute all activities, thus all activities that are not executed are blamed. Since the solution is composed of multiple schedules, there may be multiple Monte Carlo runs where activities fail to be

[3] Runtime is trivial due to a limited number of input activities.

[4] T is the number of timelines required by the scheduler.

Algorithm 1 Monte Carlo Analyzer

Input:
$A\langle p\rangle$: List of activities with priorities
S: List of all final schedules after simulating execution
Output:
U: List of all unscheduled activities
$score$: Score (objective function)
for each $S_i \in S$ **do**
  $U \leftarrow U \bigcup \{\forall a \in A | a \notin S_i\}$
  $score \leftarrow score +$ get_score(S_i)
end for

executed. For simplicity, we generally assign equal blame to any activity that was not executed in any of the simulations. However, further on we describe an approach that adjusts the blame factor according to how many times an activity failed to schedule and execute.

Parametizer

There are three basic heuristics for changing activity priorities. **Constant Step** increases the priorities of all blamed activities by a static amount x. **Stochastic Step** increases the priorities of all blamed activities by a random amount between 1 and N. This helps avoid local maxima. **Max Step** promotes any blamed activity to the highest scheduling priority. Each of these heuristics restarts to a random priority set if a cycle is detected.

Constant Step performs poorly due to cycles, eventually devolving to random walk. Although Max Step is often too coarse in its search and Stochastic Step too random to reliably converge in the limited steps available, Max Step and Stochastic Step still perform well in empirical analysis. However, these heuristics are quite naive and leave room for improvement. Specifically, these heuristics do not attempt to analyze the structure of the plan, do not fully utilize the multiple re-invocations of the scheduler (focusing instead solely on the final executed schedule), ignore preferred time as an adjustable input parameter, and do not utilize information from past iterations. The following heuristics attempt to address those shortcomings.

Intersect The Max Step heuristic does not analyze the plan when setting activity priorities, instead setting any blamed activity with the highest priority. As a result, it often results in a higher priority than necessary and causes other lower priority activities to fail to be scheduled. The *Intersect* heuristic attempts to promote a blamed activity's priority to its optimal position by analyzing each activity's execution time constraints and unit resources.

Most of the time, the reason an activity cannot be scheduled within its execution time constraints is due to either a) insufficient consumable resources (e.g. energy) or b) unit resource claims being blocked by higher priority activities. The order in which activities are scheduled is unlikely to fix any issues caused by consumable resources. However the order in which activities are scheduled will affect where activities are scheduled (and unit resource usage) and guide activities towards their optimal priority.

Priority of each blamed activity A_i is derived as follows:

$$\begin{aligned} p_i &= \max(p_i, \max_{j \in K} p_j) + 1 \\ K &= \{j \mid A_j \in A \wedge \ intersect(T_i, T_j) \\ &\qquad \wedge \ \{R_i \cap R_j\} \neq \emptyset\} \end{aligned} \tag{3}$$

where T is the set of start time windows for each activity and R is the set of unit resources for each activity. By promoting a blamed activity above only the activities that a) share a unit resource and b) intersect with its execution time constraints, the intersect heuristic can utilize knowledge of activity interactions to adjust its priority towards its optimal priority setting.

Intermediate Schedules The scheduler is re-invoked multiple times throughout execution and generates multiple intermediate schedules along with the final executed schedule. It is possible for an activity to be executed successfully even though it was not successfully scheduled in all of the intermediate plans. This could occur if activities finish early allowing the scheduler to reschedule with more time and energy available, thus scheduling and executing activities that could not fit using the original, conservative durations. Even if an activity executes, being absent from one or more intermediate schedules is an indication that the activity is at risk of not executing compared to an activity that was scheduled in all intermediate schedules. Thus, the *Intermediate Schedules* heuristic takes into account whether or not each activity appeared in all intermediate schedules, not just if it executed. An activity's scheduling priority is adjusted as follows:

- If an activity was executed but not scheduled in any intermediate schedule, then its priority is increased by $w_1 \cdot s$.
- If an activity failed to execute then its priority is incremented by $w_2 \cdot e$.

where s is the number of times the activity failed to schedule, e is the number of times the activity failed to execute, and w_1 and w_2 are derived constant weights. We derived w_1 and w_2 through trial and error testing and we recognize that there may be other heuristics that prove more effective.

Lastly, since ensuring that an activity is executed is significantly more important than ensuring that an activity is scheduled, a final sort enforces that any activity that failed to execute has a higher priority than any activity not scheduled in an intermediate schedule, but successfully executed.

Preferred Time One of the primary goals of manipulating the order in which activities are considered for scheduling is affecting each activity's scheduled start time. Changing an activity's preferred time is another avenue towards changing an activity's scheduled start time. In fact, for any deterministic schedule (durations do not change) where a solution (all activities are scheduled) exists, it is clear that if the preferred times of each activity were set to their scheduled start time in the solution, all activities would be able to be scheduled regardless of their scheduling order. Despite that, the reason we cannot solely look at preferred time is that a) the combinatorics of all possible start times is significantly higher than that of all possible scheduling orders and b) execution can present situations where scheduling order is important.

Another possible way of guiding where an activity should be scheduled is by changing its start time window. Given that a) start time windows can only be shrunk, not grown [5] and b) the goal is to schedule all activities, not as many as possible, we can prove that manipulating an activity's preferred time will result in no worse than manipulating start time windows. To prove this, let $A_{i'} = A_i$. We can manipulate activity $A_{i'}$'s start time window by:

$$\begin{aligned} T_{i'_{k_end}} &= T_{i'_{k_end}} - x : x > 0 \\ \text{or } T_{i'_{k_start}} &= T_{i'_{k_start}} + y : y > 0 \end{aligned} \tag{4}$$

k can be any arbitrary start time window $k \in T_{i'}$ since every start time window is disjoint and a preferred time can be specified for any start time window.

There are two situations.

1. *$A_{i'}$ cannot be scheduled in the window.* This is the worst result; therefore A_i cannot be worse than $A_{i'}$.
2. *$A_{i'}$ can be scheduled in the window.*

$$\text{Let } schedulable(A_{i'}, x) \text{ at any start time } x \in X$$
$$\Rightarrow \forall\, x\; A_{i_{scheduled_start}} = x, \text{ if } T_{i_{k_preferred}} = x$$

Therefore, A_i can achieve the same results as $A_{i'}$ by manipulating $A_{i_{k_preferred}}$.

The only way shrinking an activity's start time window would be better than manipulating preferred time would be if we were to give up on one activity in order to schedule another. However, our goal is to schedule all activities so we do not consider that situation.

If an activity cannot be scheduled then there does not exist a preferred time that would allow it to be scheduled since preferred time is a soft constraint. Thus changing the preferred time of blamed activities, B, would be fruitless. Instead, the preferred times of non-blamed activities are shifted through two approaches, described as follows:

1. *Start Time Window* For each start time window in each non-blamed activity, determine a range, r, that does not intersect with the start time windows of any blamed activities. If no such range exists, use the existing start time window. Then, stochastically change the preferred time to the earliest, latest, midpoint, or random time within r. This heuristic utilizes the same principles as the Intersect heuristic to prevent the activity from blocking any blamed activity. The stochastic nature prevents it from getting stuck in local maxima and cycles.
2. *Past Start Time* It is rarely the case that a blamed activity A_i, fails in all iterations of input parameters and Monte Carlos. Instead, A_i will schedule with some set of input parameters, but cause another activity to fail to schedule instead. The times where A_i was successfully scheduled signify potential regions where A_i could be scheduled again. Using preferred time, we can shift other activities' (that share unit resources and intersect start time windows) preferred times away from times where activity A_i was successfully scheduled, potentially giving it room to be scheduled in a future iteration. If the blamed activity A_i has not been successfully scheduled yet, then the algorithm behaves the same as the Start Time Window preferred time approach.

[5] growing the window enables solutions that are not a subset of the original solution set

Portfolio Each of the aforementioned heuristics attempts to analyze a different part of the plan structure. Thus, certain heuristics may prove more effective for certain types of schedules, activities, and constraints. For example, the Intersect heuristic might prove effective when the start time windows are small and when unit resources are shared rarely, but would eventually devolve into Max Step if every activity shared the same unit resources and start time windows.

Past knowledge indicates that there are potentially a wide variety of plan types to consider. Not only that, but since operations knowledge is based on Mars missions (e.g. MSL, MER) without an onboard scheduler, we cannot be sure how different Mars 2020 plans will be structured in the future. Thus, we must be robust to multiple possibilities. A portfolio scheduler is a common technique that allows a scheduler to remain robust to a variety of situations by leveraging the strengths of each individual heuristic and widening the search scope (Gomes and Selman 2001; Huberman, Lukose, and Hogg 1997).

Portfolio in Parameter Search stochastically chooses a Parameter Search heuristic at each step of the search. The probabilities are currently equally weighted, but can be adjusted. The goal is that, while each heuristic may outperform the portfolio approach for certain input schedule structures, the variety the portfolio provides will allow it to outperform any individual heuristic over multiple schedule types. Note that it is possible to derive an improved portfolio probability distribution (Xu et al. 2008; Engelhardt and Chien 2001a; 2001b). We discuss this further in the Future Works section.

Empirical Analysis

In order to evaluate how well our approaches are able to generate a set of input parameters that result in an optimal schedule, each approach is applied to various sets of input constraints and compared against each other as well as various static algorithms. Parameter Search is initialized with the latest start static algorithm as its runtime is trivial and it performs the best among the static algorithms.

We use *Copilot* to construct a schedule and simulate plan execution. *Copilot* includes the *M2020 surrogate scheduler*—a Linux workstation environment implementation of the same algorithm as the M2020 onboard scheduler (Rabideau and Benowitz 2017). The surrogate scheduler is expected to produce the same schedules as the operational scheduler, but runs much faster in a workstation environment. Copilot is expected to assist in validating the flight scheduler implementation and also in ground operations for the mission.

The plan inputs are derived from *sol types*—the current best available data on expected M2020 rover operations (Jet Propulsion Laboratory 2018). Each sol type contains between 20 and 40 activities. Data from the Mars Science Laboratory (MSL) Mission (Jet Propulsion Laboratory 2017; Gaines et al. 2016) indicates that activities were conserva-

tive and completed approximately 30% early. There is a desire by the M2020 mission to operate with less margin to increase productivity. We use normal distributions to determine the activity execution durations; the mean is 90% of the given duration and the standard deviation is based on the percent of activities expected to run late (values falling to the right of the given duration in the normal distribution). This value is set to 10% by default, but is varied to further constrain our plans. Activity durations are independent of each other, and 10 Monte Carlo simulations are run per step of Parameter Search.

While the sol types are accurate representations of plan intents (e.g. drilling vs imaging), there has been strong concern about how accurate the individualized constraints are in relation to actual M2020 operations since past rover operations knowledge (MSL, MER) was without an onboard scheduler. As a result, the accuracy of the assumptions would be higher for the types of input activities (plan intent), but lower for the specific input constraints (science preferred time, start time windows). Thus, we must be robust to a variety of constraint possibilities. To simulate those possibilities, we take the existing *sol types*, constrain them, and then analyze the effectiveness of our algorithms on the constrained plans. They are constrained as follows:

1. *Shrink Start Time Windows* Each activity's earliest start time is increased by the same amount its latest start time is decreased. Start time windows are constrained from 10% to 50% in multiples of 10%.
2. *Vary Incoming SOC* Incoming SOC (energy) is decreased from 100% to 60% of the maximum SOC in multiples of 10%.
3. *Actual Duration Variation* Currently, our actual activity durations are based on a probabilistic distribution. Increasing the portion of the distribution that falls to the right of the given, conservative duration will increase both the number of activities that finish late and by how much they finish late. The distribution is varied such that 0% to 40% of activities finish late in multiples of 10%.

It may not be possible to execute all activities in every variation due to plan constraints. However, it is difficult to determine a more accurate upper bound as that would require a tractable, optimal scheduler. As such, we compare against a perfect score that assumes all activities given can be executed.

Results

In all of the sol type variations, Parameter Search outperforms the static algorithms. This can be seen in both the overall results (Figure 4d) and across the individual variations (Figures 4a, 4b, 4c). The differences are statistically significant ($p < 0.01$) for every Parameter Search approach against every static algorithm.

In individual plan variations, the portfolio approach does not statistically outperform and sometimes under-performs certain Parameter Search heuristics. This is to be expected as each heuristic has its specific strengths, making it specialized but not versatile. In Figure 4a, the three priority setting heuristics—Max Step, Stochastic, and Intersect—perform better than the preferred time approaches. This is because the tighter start time windows limit the adjustment of preferred times, thereby limiting their effectiveness as well. Figure 4b shows a similar trend; the Max Step and Stochastic heuristics even outperform the portfolio approach as the incoming SOC is decreased. In theory, the preferred time heuristics could have performed well in energy constrained schedules by targeting regions where less preheat is necessary, but this was not implemented in the current approach. This will be developed and tested in a future implementation. In contrast, Figure 4c shows both preferred time approaches as the top performers after the portfolio approach. As variance increases, there will be situations where an activity finishes late and forces the execution system to handle the next back-to-back activity. The existing execution system is based on fixed start times and allows limited delays of activity start times (Chi et al. 2018). Delays cannot always be handled and may result in an inconsistent schedule and unscheduled activities. If the two activity durations are similar, changing their relative priorities will not be effective as the activities will remain back-to-back and the likelihood that either activity finishes late would be equal given their equal probabilistic distribution. However, if preferred times were set to prevent back-to-back activities, we can avoid the need to rely on the execution system.

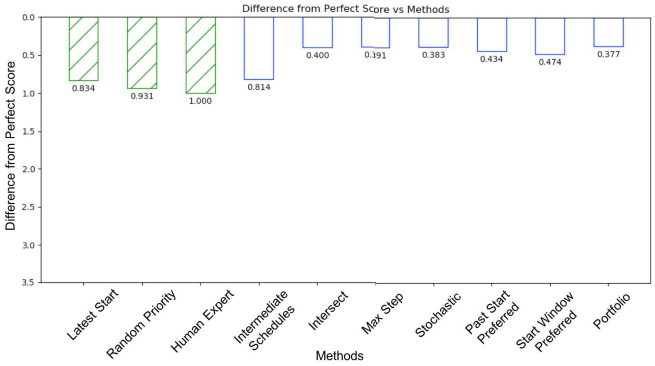
Method	Mean	Std. Dev	t	df	Sig. (2-tailed)
Max Step	0.087	0.997	2.816	1049.000	0.005
Stochastic	0.072	0.835	2.807	1049.000	0.005
IntermediateSchedules	0.335	1.577	6.884	1049.000	0.000
Intersect	0.118	1.027	3.725	1049.000	0.000
Start WindowPreferred	0.067	0.821	2.631	1049.000	0.009
Past StartPreferred	0.081	0.846	3.098	1049.000	0.002

Table 1: 1 sample t-test of the difference from the portfolio approach. Lower means are closer to the portfolio approach.

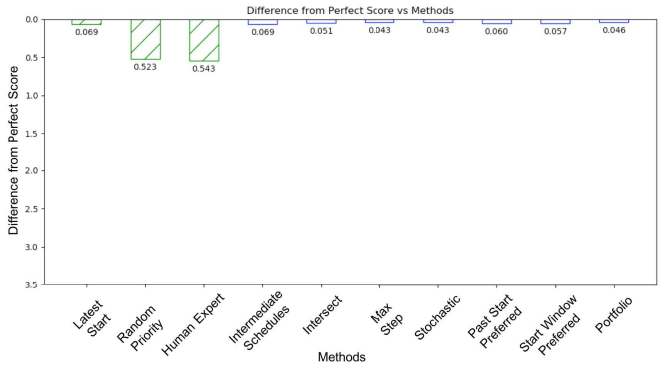
For each sol type variation, it would be better for the portfolio to focus on the type of approach best suited for the variation type. Training the portfolio to adjust its probability distribution based on plan structure analysis would help the portfolio overcome this shortcoming. Despite that, in Figure 4d it can be seen that the portfolio approach performs better overall than any individual algorithm, showcasing its versatility across a wider range of problem sets. A difference of 0.05 mandatory activities is noteworthy because that translates to an average of 1 mandatory activity per 20 Sols; for comparison, MSL dropped an average of 1 activity every 90 sols (Gaines et al. 2016). In addition, the differences are statistically significant ($p < 0.01$) as shown in Table 1.

While the portfolio approach performs the best, a concern is that the use of multiple heuristics may result in a slower convergence on the solution. However, in Figure 5, we can see that the portfolio based approach converges to its solution in a number of steps comparable to the other heuristics. The time spent on each Parameter Search approach is approximately equal. Each approach only considers one child for its next iteration, and the constructor, which is the same for every approach, dominates the parametizer and analyzer as described in Equation (2).

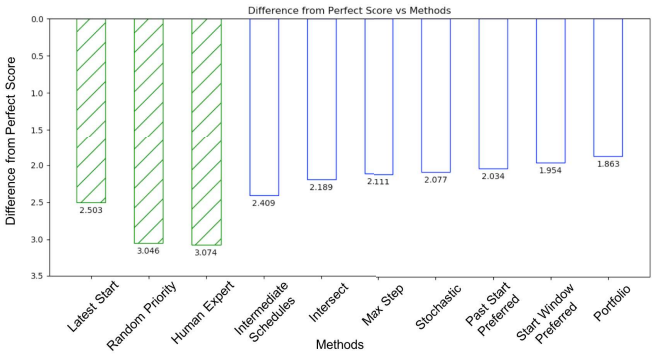
Figure 6 shows how Parameter Search improves upon



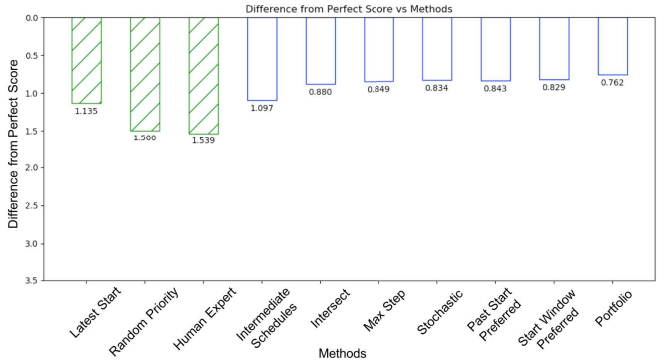
(a) Shrink Start Time Windows



(b) Vary Incoming SOC



(c) Activity Duration Variance



(d) Overall Average

Figure 4: Difference from the perfect activity score vs. each method per sol type variation. Static algorithms are green and hatched and Parameter Search heuristics are blue and not hatched. Lower values (Shorter bars) indicate less activities dropped and is better. One point equals one activity. Methods from left to right are Latest Start, Random Priority, Human Expert, Intermediate Schedules, Intersect, Max Step, Stochastic, Past Start Preferred, Start Window Preferred, and Portfolio.

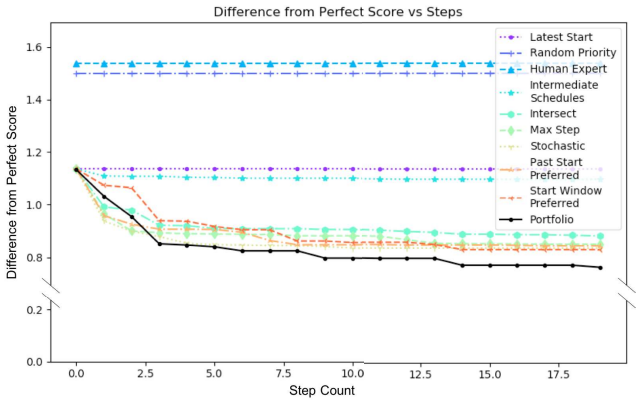


Figure 5: Difference from the perfect activity score vs. number of steps. Lower values are better. Portfolio is black.

handover SOC. Despite being a secondary score, the handover SOC score is improved upon by Parameter Search alongside the number of activities scheduled. This is likely due to Parameter Search improving the structure of the plan in ways such as shifting activities to regions where less preheat is necessary or reducing unnecessary rover awake time.

Figure 6 also provides additional evidence that the portfolio approach outperforms individual heuristics.

The Intermediate Schedules heuristic consistently underperforms the other heuristics. This is mainly because this heuristic effectively uses a different utility function; one where being scheduled is important, not just being executed. However, to compare all of the methods equally, we used Equation 1 as our utility function where only successful execution matters. Therefore, the strengths of the Intermediate Schedules heuristic were not highlighted in our analysis.

Related Work

Parameter Search is inspired by Squeaky Wheel Optimization (SWO). Typically, SWO uses a constructor, analyzer, and prioritizer for the next iteration of schedule generation (Joslin and Clements 1999). Parameter Search differs in that the intent is not to generate a good schedule but rather to set parameters that perform well in execution and rescheduling. Therefore the Parameter Search constructor must use the scheduler through multiple runs of execution (where each run of execution incurs multiple scheduler invocations) to assess parameter assignment performance.

Portfolio algorithms (Gomes and Selman 2001; Leyton-

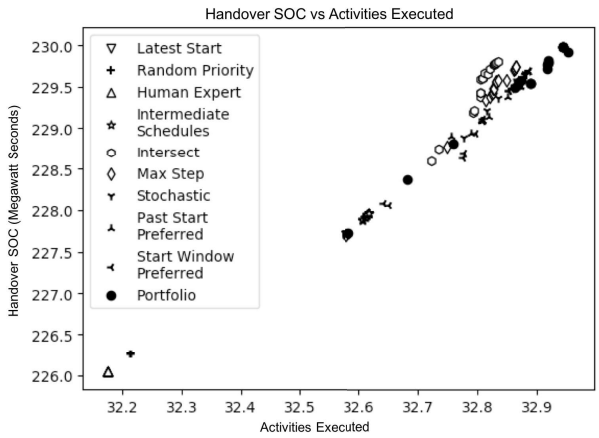


Figure 6: Handover SOC vs. activities executed. Higher and to the right is better.

Brown et al. 2003) have proven their effectiveness when dealing with a variety of problem sets. These works differ in their focus on runtime improvement since each individual algorithm is able to solve the problem (or prove no solution exists). Our heuristics and algorithms may not be able to solve the problems individually (e.g. some problems cannot be solved with just priority setting and require preferred time manipulation) and thus the focus is on the overall utility improvement from running a portfolio.

Generating schedules that are robust to execution run time variations (Leon, Wu, and Storer 1994) is a mature area of work. However, the topic usually revolves around developing a scheduler that can generate robust schedules. In our case, the scheduler is a) a fixed "black box" that we have no control over and b) robust to execution run time variations mainly through rescheduling. As a result, rather than developing a scheduler itself, we are developing a methodology that is able to generate a set of parameters for a fixed scheduler that enables it to be robust to rescheduling due to runtime variations.

Other approaches (Drummond, Bresina, and Swanson 1994; Washington, Golden, and Bresina 2000) use branching to increase robustness - these differ from our work that adjusts parameters and allows rescheduling.

A number of other spacecraft (Muscettola et al. 1998; Pell et al. 1997; Chien et al. 2005; 2016) and rover (Woods et al. 2009; Gregory et al. 2002) autonomy systems have included planning, but these differ in that we are deriving control information specific to scheduling for a limited context - e.g. one rover sol temporal schedule.

Other approaches that utilize repeated sampling to search the solution space include: a) Monte Carlo Tree Search which builds a search tree according to random samples (Browne et al. 2012), b) Hindsight Optimization which generates scenarios to be solved by a deterministic solver (Chong, Givan, and Chang 2000), and c) Reinforcement Learning which translates differing actions and their change to environmental state (Kaelbling, Littman, and Moore 1996). These differ in that they search each node in the tree, while we search leaf nodes only.

Future Work

As described in Equation (2), the constructor dominates the runtime of the algorithm. While it is theoretically possible to parallelize all of the Monte Carlos, realistically this is limited by cost. Therefore, decreasing the number of Monte Carlos required would decrease overall runtime. One reason a Monte Carlo requires so many evaluations is because it repeatedly samples the mean and only rarely samples edge cases. An alternative sampling method to consider is *Importance Sampling*. Importance sampling chooses samples from a distribution which applies higher weight to the important regions (Rubinstein and Kroese 2016). It then accounts for overweighting the important regions by considering the likelihood of the importance distribution to the normal distribution. By using importance sampling (or other sampling methods) we can achieve the same accuracy as a Monte Carlo with fewer samples, allowing the spare runtime to be put into more steps of the algorithm or even branching.

Our current portfolio approach is incredibly naive, choosing the heuristic for each step stochastically without analysis. There are multiple ways to improve on this. One simple way is to run all child paths in parallel for a fixed number of steps and backtrack to the best performing one (Gomes and Selman 2001). However, this requires parallel computing which is already limited by the Monte Carlos. Another approach is to randomly restart after the current algorithm slows in its improvement of the utility function. This would shorten the search depth, but can be suprisingly effective (Gomes and Selman 2001). Lastly, we can apply learning to adjust the portfolio probability distribution based on plan structure analysis (Xu et al. 2008; Engelhardt and Chien 2001a; 2001b).

The M2020 Onboard Scheduler must deal with two tiers of activities: mandatory and optional. In this paper we focused only on and assumed that all activities are mandatory activities. Optional activities add another layer of challenge as 1) not all optional activities must be scheduled and 2) they may need to adhere to certain constraints (e.g. handover minimum SOC) that mandatory activities can ignore. Mixed-Criticality Schedulers deal with systems where there are two or more levels of criticality (Burns and Davis 2013) and may be an area of future work.

Other approaches or tools such as paramILS (Hutter et al. 2009) and potential theoretical guarantees are also topics of future interest.

Conclusion

Deriving parameters for a limited embedded scheduler is challenging due to the need to optimize for rescheduling and uncertain execution. To account for execution, Parameter Search adjusts parameters based on feedback from simulated execution and rescheduling using Monte Carlo simulations. We derive a range of potential sol type variations based on best available planetary rover operations knowledge and data, and present an empirical evaluation showing Parameter Search outperforms static parameter assignment methods including manual expert derived priorities.

References

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1):1–43.

Burns, A., and Davis, R. 2013. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep* 1–69.

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 *ICAPS*.

Chien, S. A.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Mandl, D.; Trout, B.; Shulman, S.; et al. 2005. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing Information and Communication* 2(4):196–216.

Chien, S.; Doubleday, J.; Thompson, D. R.; Wagstaff, K. L.; Bellardo, J.; Francis, C.; Baumgarten, E.; Williams, A.; Yee, E.; Stanton, E.; et al. 2016. Onboard autonomy on the intelligent payload experiment cubesat mission. *Journal of Aerospace Information Systems*.

Chong, E. K.; Givan, R. L.; and Chang, H. S. 2000. A framework for simulation-based network control via hindsight optimization. In *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, volume 2, 1433–1438. IEEE.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *AAAI*, volume 94, 1098–1104.

Engelhardt, B., and Chien, S. 2001a. Empirical evaluation of local search methods for adapting planning policies in a stochastic environment. In *Proceedings of the Workshop on Local Search for Planning and Scheduling-Revised Papers*, ECAI '00, 108–119. Berlin, Heidelberg: Springer-Verlag.

Engelhardt, B., and Chien, S. 2001b. Stochastic optimization for adapting behaviors of exploration agents. In *International Symposium on Artificial Intelligence, Robotics, and Automation for Space (ISAIRAS 2001)*.

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

Gomes, C. P., and Selman, B. 2001. Algorithm portfolios. *Artificial Intelligence* 126(1-2):43–62.

Gregory, N. M.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International Workshop on Planning and Scheduling for Space*.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economics approach to hard computational problems. *Science* 275(5296):51–54.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36:267–306.

Jet Propulsion Laboratory. 2017. Mars science laboratory mission https://mars.nasa.gov/msl/ 2017-11-13.

Jet Propulsion Laboratory. 2018. Mars 2020 rover mission https://mars.nasa.gov/mars2020/ 2018-11-02.

Joslin, D. E., and Clements, D. P. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4:237–285.

Leon, V. J.; Wu, S. D.; and Storer, R. H. 1994. Robustness measures and robust scheduling for job shops. *IIE transactions* 26(5):32–43.

Leyton-Brown, K.; Nudelman, E.; Andrew, G.; McFadden, J.; and Shoham, Y. 2003. A portfolio approach to algorithm selection. In *IJCAI*, volume 3, 1542–1543.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103(1-2):5–47.

Pell, B.; Gat, E.; Keesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *International Joint Conference on Artificial Intelligence*, 1234–1239.

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

Rubinstein, R. Y., and Kroese, D. P. 2016. *Simulation and the Monte Carlo method*, volume 10. John Wiley & Sons.

Washington, R.; Golden, K.; and Bresina, J. 2000. Plan execution, monitoring, and adaptation for planetary rovers. *Electron. Trans. Artif. Intell.*

Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an exomars rover–like mission. *Journal of Field Robotics* 26(4):358–390.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research* 32:565–606.