CONSTRAINT-BASED BRITTLENESS ANALYSIS OF TASK NETWORKS FOR PLANETARY ROVERS

Virtual Conference 19–23 October 2020

Tiago Stegun Vaquero, Steve Chien, Jagriti Agrawal

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive Pasadena, CA 91109 USA
E-mail: tiago.stegun.vaquero@jpl.nasa.gov , steve.a.chien@jpl.nasa.gov, jagriti.agrawal@jpl.nasa.gov

ABSTRACT

We propose a new method to analyze brittleness of task networks with respect to not only temporal but any arbitrary set of constraints. The method allows the detection and enumeration of activities that, with modest duration variation, violate the target constraints.

1 INTRODUCTION

Temporal plans for agents that must deal with execution uncertainty must be designed for execution robustness. Planetary rovers, due to their interaction with a hard-to-predict environments match this type of problem. Planetary rover plans must be carefully designed and generated on the ground to allow successful execution of tasks while meeting all the required timing constraints, energy constraints, and being safe at all times. Accurately determining some of those timing constraints a priori, specially activity duration, is quite challenging though due to the natural unpredictability of the environment [1][2].

The traditional approach used in planetary rover missions is to add significant temporal margin for execution robustness; however, this can hamper rover efficiency [3]. Ideally we would use task networks that not only are consistent and maximize the vehicle's productivity, but that are also robust to unexpected events and delays. As a corollary, it is therefore critical to evaluate robustness and identify activities and (temporal and resource) constraints that cause brittleness to temporal unpredictability.

This paper addresses the challenge of identifying the activities that are most sensitive to temporal unpredictability, in a dynamic scheduling setting, given a set of user-specified constraints. We propose a new method to analyze the brittleness of task networks with respect to not only temporal but any arbitrary set of constraints (including resources). It allows the detection and enumeration of activities that, with modest task execution duration variation, violate the target set of constraints and make the success of execution no longer guaranteed. In this method, we introduce a metric for measuring an activity's brittleness - defined as the degree of acceptable deviation from its nominal duration - and describe how that measurement is mapped to task network structure. Complementary to existing work on temporal robustness analysis which informs how likely a task network is to succeed or not under controllability constraints, the proposed analysis and metric not only generalized robustness analysis to any arbitrary set of constraints (herein temporal controllability and energy resource constraints), but it also goes deeper to pinpoint the sources of potential brittleness. We develop an analyzer that helps human designers and/or automated task network generators (e.g. schedulers/planners) focus on and address sources of undesirable brittleness. For the latter, the analyzer proposes temporal constraint relaxations (e.g. changing start time or execution time windows of certain activities) when the brittleness metric values are not satisfactory. We apply the approach to a set of task networks in development for NASA's next planetary rover and present common patterns that are sources of brittleness. The proposed technique and tools are currently under evaluation for potential use supporting operations of the Mars 2020 rover.

2 BACKGROUND

We study brittleness analysis in the context of NASA's next planetary rover, the Mars 2020 (M2020) rover operations. We derive task networks from M2020 sol types [4]. In what follows we describe the main elements of a rover task network and existing robustness metrics.

2.1 2020 Rover's Task Networks

Based on [2, 5], we define a M2020 sol type task network as a tuple $TN = \langle A, H, SOC^{init}, SOC^{min}, SOC^{max}, SOC^{handover} \rangle$. The main element of our task network refers to a set of rover activities $A=\{a_i\langle TC_i,D_i,e_i\rangle \ldots a_n\langle TC_n,D_n,e_n\rangle\}$, where: a) TC_i is the temporal constraints tuple $\langle TC_{i,dur}, TC_{i,est}, TC_{i,lst}, TC_{i,let}\rangle$ referring to the nominal duration, earliest start time, latest start time and latest end time respectively (the duration of planetary rover activities can be quite unpredictable [3] and $TC_{i,dur}$ is expected be similar); b) D_i is the set of the activity's dependency constraints in the form of $a_j \rightarrow a_k$, i.e. a_j depends on a_k;

c) e_i is the rate at which the consumable resource energy is consumed by activity a_i.

All activities in A must be scheduled (no disjunction is considered). In addition to set A, our task network is defined by: d) H is the execution horizon (all activities have to finish by then); e) SOC^{init} is the initial state of charge (SOC) of the rover's battery at the start of task network execution; f) SOC^{min} is the global minimum state of charge that is required at all times during execution; g) SOC^{max} is the global maximum state of charge during execution (it never exceeds it during charging (sleep mode)); h) $SOC^{handover}$ is the handover minimum state of charge that is required at the end of the execution.

2.2 Robustness Analysis

State-of-the-art analysis has focused on quantifying how robust a temporal constraint network is with respect to disturbance (e.g., unforeseen delays). In [6, 7], robustness is computed by measuring a Simple Task Network (STN) flexibility: a metric that quantifies the aggregate slack in the simple temporal network. In [8], an STN is represented as a polyhedron and the flexibility linked to its volume. Although useful, it is hard to judge from the above metrics if the amount of flexibility is enough in certain unpredictable environments – with no uncertainty being consider it might not imply robustness [9]. Moreover, the above methods do not inform where more flexibility is needed if the metric value is below the acceptable range.

In [9] robustness is defined as the greatest level of disturbance (delay) on an STN with Uncertainty (STNU) at which it is still successfully executed. In this method, computing robustness is framed as a linear constraint optimization problem to compute the maximum deviation from the nominal case on any activity at which the STNU is dynamically controllable. If a Probabilistic STN (PSTN) is used instead, the level of disturbance at which the schedule becomes no longer dynamically controllable equates to the probability of it breaking during execution. The same constraint optimization problem framework applies in the probabilistic case, in which the objective is to maximize the probability that all uncertain activity durations fall inside the chosen bounds. The robustness metric in this case would be a probability of execution success.

In [10], the degree of dynamic controllability (DDC) of a STNU is used a robustness metric. DDC is defined as the proportion of STNU contingent edges realizations in which the temporal network remains dynamically controllable. A parallel could be done wrt PSTN, in which the DDC would rather be defined in terms of probability mass of the respective controllable realizations.

In [11, 12], robustness is also quantified as the likelihood that a PSTN will be executed successfully. The approach assumes no correlation between the temporal constraints, and therefore, suffers overestimation. To account for interrelation, [12] propose a Monte Carlo approach in which the execution of a PSTN is simulated multiple times and the duration of contingent/uncertain activities is sampled as they are executed. The ratio of successful executions corresponds to the robustness value.

Similar to the likelihood of success, the risk of violating any temporal constraints in a PSTN can also be interpreted as a robustness metric. Approaches like risk-bounded scheduling [13] guide the search for a temporal plan based on a user-specified risk bound (e.g., 5% risk). In [14], a risk-minimization approach is used to generate a schedule under strong controllability constraints, in which the risk value would indicate how robust the PSTN is.

The aforementioned metrics, success probabilities, degree of dynamic controllability and risk bounds, provide an intuitive way to evaluate if an task network (in the form or a STNU or PSTN) execution is likely to succeed and whether the entire network is brittle to disturbance or not (note that only temporal constraints are considered in the aforementioned robustness metrics). However, it is hard to determine potential sources of brittleness when those metrics (which boils down to a single output number) do not meet a desirable threshold. Those cases bring interesting and important questions: (a) What activities in particular are culminating in low robustness? (b) What activities and constraints (temporal and/or resource) are most brittle to delays so that one might try to address specific issues in task network? (c) What constraints are dominantly more brittle to delays? (d) What are the possible ways we can change or relax some of the temporal constraints to bring activities' and the task network's brittleness values to an acceptable level? In this paper we propose methods to address these questions.

3 TASK NETWORK BRITTLENESS ANALYSIS

In this work, we represent uncertainty by associating a mean (μ) and a standard deviation, sigma (σ) (e.g. from a normal distribution), to every contingent/uncertain activity's duration in the task network. We call the set of contingent activities A_C and the non-contingent/controllable activities A_R,

where $A=A_C \cup A_R$ and herein $TC_i = \langle \mu_i, \sigma_i, TC_{i,dur}, TC_{i,est}, TC_{i,lst}, TC_{i,let} \rangle$.

We propose a constraint-based brittleness analysis approach for task networks with uncertainty in which, given a set of user-specified brittleness constraints $BC=\{bc_1...bc_k\}$, we enumerate contingent activities, from most brittle to least brittle, and inform the causes of potentially undesirable brittleness levels. In a nutshell, we use a *ceteris paribus* approach (i.e. "all else unchanged"), leveraging [15], to analyze the brittleness of each contingent activity individually against *BC* by exploring different disturbance scenarios and evaluating which brittleness constraints are violated. We then overlay that information onto the task network structure to analyze why certain activities are more brittle than others.

3.1. Ceteris Paribus Analysis

In the core of the brittleness analysis, we introduce a ceteris paribus approach to measure the degree of disturbance (deviation to the mean) at which each target contingent activity $a_i \in A_C$ makes the task network violate one or more constraints in *BC*. For each activity, we measure its maximum duration deviation from the mean, while fixing the uncertainty (deviation) of all other remaining contingent activities to a certain level (called a ceteris paribus scenario).

Maximum deviation is related to a *maximum sigma multiplier*, z_i^u, that makes the task network violate one or more brittleness constraints *BC*, where the max duration bounds for that particular activity would be $(t_{i\_end} - t_{i\_start}) = [\mu_i - z_i^u \times \sigma_i\ ,\ \mu_i + z_i^u \times \sigma_i]$ and the set of violated constraints would be $BC_i^u \subseteq BC$ for a particular ceteris paribus scenario. The intuition here is: the smaller the z_i^u, the more brittle the activity is in the network. In this work, we use the sigma multiplier z_i^u as our central *measure of brittleness* and the means to enumerate brittle activities. We use the terms 'degree of brittleness' or 'brittleness level' interchangeably to refer to that measure. In what follows we describe the proposed analysis algorithm to compute z_i^u.

Algorithm 1 is designed to order activities by their degree of brittleness. It is composed of two main steps: **Step 1** detects the single overall sigma multiplier (applied to all the contingent activities edges at once) that makes the network *TN* violate the conjunction of constraints in *BC* (i.e., the *BC* check). That overall sigma multiplier is used in the second step to created specific *ceteris paribus* scenarios when analyzing each individual activity. In *line 2*, the called function adds uncertainty around the mean μ of all activities duration by applying a single increasing overall

Algorithm 1: Ceteris Paribus Brittleness Analysis
Input: *TN, BC, F_x*
Output: a list of contingent activities ordered by z_i^u along with their respective violated brittleness constraints.
1 *multipliers* ← {};
// **Step 1**
2 x^u ← *computeOverallSigmaMultiplier*(*TN,BC*);
// **Step 2**
3 $Y = x^u \times Fx$; // Sets fixed *y* multiplier
4 **for** $a_i \in A_C$ **do**
5 z_i^u, BC_i^u ← *computeActivitySigmaMultiplier*(a_i,*y,TN,BC*);
6 *multipliers.append*((a_i, z_i^u, BC_i^u));
7 **end**
8 *ordered_multiplier* ← *order*(*multipliers*);
9 **return** *ordered_multiplier*

sigma multiplier x ($0 \le x \le \infty$) to all contingent activities, generating a task network *TN* with new (larger) activities duration bounds. At each increment of x we check if the resulting task network satisfy all the brittleness constraints in *BC*. The (breaking) overall sigma multiplier that makes *TN* violate *BC* is set to be x^u.

The most important process of the Algorithm 1 occurs in **Step 2**. Here we start by setting a fixed sigma multiplier y ($0 \le y < x^u$) in *line 3* that will be used to specify the *ceteris paribus* scenario (i.e., the uncertainty level of all the other contingent activities) when analyzing a particular contingent activity a_i. The algorithm input float F_x is used to specify y as a fraction of x^u ($0 \le F_x < 1$). As we will see later, $y = 0$ (by $F^x = 0$) is a special case of the analysis. In lines 4-7, for each activity $a_i \in A_C$ we compute its sigma multiplier z_i^u and BC_i^u, and append the tuple (a_i, z_i^u, BC_i^u) to the multipliers results. The function called in line 5 computes starts by first creating an new TN in which all other contingent activities $a_j \in A_C$ ($j \neq i$) have their contingent durations set to $(t_{j\_end} - t_{j\_start}) = [\mu_j - y \times \sigma_j,\ \mu_j + y \times \sigma_j]$. We then search for the maximum sigma multiplier z_i^u applied to a_i's duration bounds that violates one or more elements of *BC*. Finally, every tuple (a_i, z_i^u, BC_i^u) is stored and later ordered by the value z_i^u (line8). The ordered list of contingent activity is then returned, with each activities brittleness value and the corresponding violated constraints.

A special case of the analysis in *Algorithm 1* is when the fixed sigma multiplier $y = 0$ in Step 2 (i.e., all other contingent activities a_j have duration set to be exactly the nominal case, $(t_{j\_end} - t_{j\_start}) = [\mu_j, \mu_j]$). In that case, we can determine the maximum temporal disturbance allowed for each activity, which sets the upper bound of each $z_i^u (z_i^{u,y=0})$. Figure 1 shows an example of ranking of activity brittleness based on z_i^u from one of the studied M2020 task networks with 20 rover activities, in which we consider three brittleness constraints in *BC:* bc_{DC} is the dynamic controllability constraint; bc_{SOCmin} is the global minimum SOC constraint; and $bc_{SOChandover}$ is the handover minimum SOC constraint. Here we

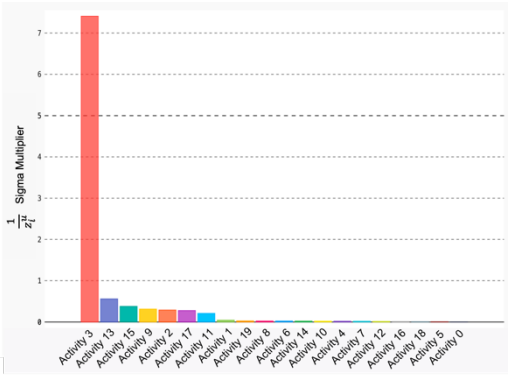

Figure 1: Example of brittle activities ranking.

use $1/z_i^u$ values to make the brittleness representation more intuitive: the higher $1/z_i^u$, the more brittle the activity is. Note that the chart in Figure 1 can be mapped to maximum delays ($z_i^u \times \sigma_i$) providing actual timing information. The proposed algorithm can inform designers or planners what activities are most brittle, which brittleness constraints are dominantly more brittle, and which of the activities might need further inspection if they do not meet a desirable brittleness level. It is worth noting that varying the fixed sigma multiplier *y* can provide useful information about interrelated contingent activities. If the resulting values of z_i^u do not change with *y* variation, then it shows that activity a^i is quite independent of other activities duration and uncertainty (or their effect is insignificant). If z_i^u value does change with varying *y*, then a_i is interrelated to other activities uncertainty.

3.2 Mapping Brittleness to Task Network Structure

We implement a visualization tool for representing the outputs of the aforementioned analysis over a temporal network graphical representation (e.g., as a STNU or a PSTN). The objective is to help a designer to identify brittleness sources. In addition to the temporal constraints and means and sigma, we provide the following information for each activity a_i in the task network visual representation: brittleness rank; z_i^u; BC_i^u, maximum delay ($z_i^u \times \sigma_i$); specified execution time window; inferred execution time window and start time window (using Floyd-Warshall's all-pair shortest path algorithm); and dependencies. By visual inspection or structure analysis, designers and planners can study the causes of brittleness given the parameters above while tracing the associated temporal constraints on the task network structure.

4 ADDRESSING UNDESIRABLE BRITTLENESS LEVELS

In this section, we propose an advisory system that suggests temporal constraint relaxations (e.g., decrease earliest start time of activity a_{drive} by 300 time units) that meets user-specified brittleness measure thresholds for a set of target activities in the *ceteris paribus* analysis - usually those activities that were found to be problematically brittle. If the suggestion is not satisfactory from the user/planner's perspective, the system would generate the next best relaxation solution. Herein, we focus on temporal constraint relaxation suggestions only. Moreover, the proposed system handles only the brittleness constraint bc_{DC}. Handling multiple constraints, specifically resources, is left for future work.

Our task network relaxation problem addressed by the proposed advisory system refers to $RP = \langle TN, A_{Brittle}, RA^{TC}, BM^{TH}, bm^{TH}_{default} \rangle$, where: $TN=\langle A, H \rangle$ is our task network, without resource constraints entirely; $A_{Brittle}$ is the set of target activities, $A_{Brittle} \subseteq A_C$, that brittle values do not meet an acceptable value and one wants to make sure they are in the brittleness measure acceptable range; RA^{TC} is a list of relaxable activities' temporal constraints (e.g. earliest/latest start times); BM^{TH} is a mapping between target activities $a_i \in A_{Brittle}$ to a brittleness measure threshold bm_i^{TH} - herein bm_i^{TH} refers to either an acceptable maximum sigma multiplier z_i^{TH} or an acceptable confidence level p_i^{TH} (probability mass between lower and upper bounds around the mean) in case of normal distribution duration models; and $bm^{TH}_{default}$ is the default brittleness measure threshold for all non-target activities. The output of our system is a set of relaxations, R^{TC}, that maps a subset of RA^{TC} to increments or decrements of time. In the aforementioned example on activity a_{drive} the solution would look like $R^{TC} = \{ra^{TC}_{drive,est} \rightarrow -300\}$ (subscript *drive,est* refers to earliest start time constraint of activity a_{drive}).

Algorithm 2 provides the pseudo code for the proposed advisory system. The first step (line 2) corresponds to the translation of the input TN to a STNU, generating TN_{STNU}. The resulting STNU has to be dynamic uncontrollable otherwise no relaxation is needed. Given TN_{STNU}, we then compute a solution for the relaxation problem under the dynamic controllability constraint (line 3). Here we use the Conflict-Directed Relaxation with Uncertainty (CDRU) algorithm [9] to solve a linear program optimization problem over the uncontrollable STNU. CDRU finds a least-cost relaxation of the temporal constraints in RA^{TC} that makes TN_{STNU} dynamic controllable. Herein, relaxing refers to tightening or loosening temporal constraints edges. Finally, Algorithm 2 allows an interactive process of generating the next best solution (lines 4-6), as long as one exists. This is possible thanks to *search_pointer* that stores the CDRU's search state and learned conflicts

Algorithm 2: Brittleness Level Guided Task Network Relaxation

Input: A task network temporal relaxation problem, *RP*
Output: a set of relaxations, RA^{TC}

1 $R^{TC} \leftarrow \{\}$;
// translate task network to its corresponding STNU
2 $TN_{STNU} \leftarrow$ *generateSTNU(TN,* $A_{Brittle}$, RA^{TC}, BM^{TH}, $bm^{TH}_{default}$*);*
// compute relaxation
3 *search_pointer,*$R^{TC} \leftarrow$ *computesRelaxationSolution(*TN_{STNU},RA^{TC}*);*
// provide next best solution if required
4 **while** *User needs next solution and one exists* **do**
5 $R^{TC} \leftarrow$ *search_pointer.nextSolution();*
6 **end**
7 **return** R^{TC}

to keep computing solutions. We believe this provides a powerful tool for designers and planners to explore different options of temporal constraints relaxations and therefore ways to bring the task network to the acceptable brittleness level.

5 EXPERIMENTAL RESULTS

5.1 Setup

We run the constraint-based brittleness analysis using a set of nine M2020 sol type task networks. These networks are representative of what is currently being investigated to develop an onboard scheduler for the M2020 rover [2]. Sol types generally contain between 20 and 50 activities, such as driving, conducting remote science, and taking images. Table 1 show some of the main properties of each task (i.e. number of activities and dependencies, as well as the resulting number of nodes and edges when translating it to an STNU).

Since the M2020 rover is not yet in operations, accurate models of activity duration variance are not yet available. For the purpose of this study, we use an estimate of nominal activities duration as their mean value (based on conservative estimates from scientists) and select a sigma for each activity randomly as a fraction of the mean. With respect to energy constraints, we use conservative estimate values for activity's energy consumption rates (e_i), as well as estimate of SOC^{init}, SOC^{min}, SOC^{max}, and $SOC^{handover}$ based on target operation requirements. In this work, we focus on the three brittleness constraints *BC*: bc_1=bc_{DC}, bc_2=bc_{SOCmin} and bc_3=$bc_{SOChandover}$.

For each task network, we perform the following: 1) run Algorithm 1 in the special case $y = 0$ to identify the upper bound $z_i^{u,y=0}$ and plot the brittleness ranking; and 2) we use the mapping tool to generate graphical representation (STNU-like) of the task network with the added layer from the brittleness analysis data. Finally, we select a representative task network from the analysis results to run Algorithm 2 and generate a temporal relaxation solution to improve temporal brittleness, considering constraint bc_{DC} only.

Task Network	# Act	# Dep	# Nodes	# Edges
TN 0	32	27	69	193
TN 1	40	43	83	237
TN 2	28	25	59	162
TN 3	18	21	39	104
TN 4	42	42	87	251
TN 5	42	42	87	251
TN 6	35	42	73	200
TN 7	46	55	95	279
TN 8	20	18	43	115

Table 1: Studied M2020 sol types task networks with their respective properties

Task Network	x_i^u	$z_{i,max}^{u,y=0}$	$1-p_i^{u,y=0}$	# bc_1 viol.	# bc_2 viol.	# bc_3 viol.
TN 0	0.095	1336.77	0.46	29	3	0
TN 1	0.045	1117.31	0.48	37	3	0
TN 2	0.060	389.71	0.47	25	3	0
TN 3	0.030	3076.92	0.49	16	2	0
TN 4	0.150	681.20	0.44	39	3	0
TN 5	0.140	595.60	0.44	39	3	0
TN 6	0.030	5241.09	0.49	32	2	1
TN 7	0.030	980.40	0.46	43	3	0
TN 8	0.135	896.86	0.44	19	1	0

Table 2: Summary of brittleness analysis results

5.2 Results

Table 2 shows the overall sigma multiplier x_i^u for each task network and the maximum values of the target sigma multiplier $z_i^{u,y=0}$ to illustrate the large range of brittleness in the set. Minimum values of $z_i^{u,y=0}$ (most brittle activities) matches the value of x_i^u, except *TN 7* where $z_{i,min}^{u,y=0} = 0.095$. These results show a brittle set of task networks, given the low x_i^u, that is: with a small variation of one key activity's duration the task network violates at least one of the brittleness constraints. The low sigma multipliers reflects on the high value of the probability of falling beyond the $z_i^{u,y=0}$ values in the activity delay case only ($1-p_i^{u,y=0}$), showing a high risk of constraint violation. Table 2 also shows dynamic controllability (bc_1) as the dominant brittleness constraint.

Plotting brittleness ranking, like in Figure 1, provides an effective tool for highlighting the fragile activities. Figure 1 shows the case of *TN 8* where activity 3 is the most brittle, and largely more brittle than the other activities. The results from the ranking and the mapping approach show two main sources/cases of temporal brittleness in the studied set of M2020 task networks: ***Case 1*** represents brittleness due to a tight user-specified execution window; and ***Case 2*** represents brittleness due to a dependency chain. Activities in Case 1 would usually have no temporal dependency to other activities, but a tight window between the earliest start time and the latest end time. Case 2 is manifested by clear temporal/ordering dependencies in the graph representation. In what follows we use task network *TN 2* as our representative example since it covers the two aforementioned cases.

Figure 2 represents a portion of the results from the mapping tool in the TN 2 case. The brittleness of Activity 9 – a Long Drive activity – in Figure 2 (a) is primarily attributed to its tight execution window

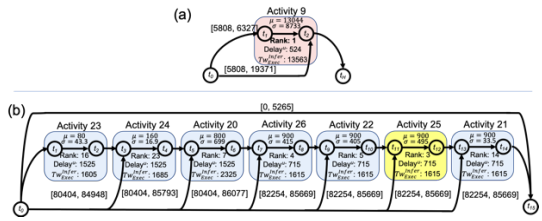

Figure 2: Brittleness due to (a) tight user-specified execution window (Case 1) and (b) to dependencies (Case 2) in TN 2.

compared to its large nominal duration (Case 1). Each activity may have a cutoff time to ensure that the activity does not interfere with another higher priority activity. If the activity runs past its cutoff time, then it is aborted and the activity fails to be scheduled. Even though the long drive has no dependencies or resource contention with any other activity, its nominal duration is 13044 sec (3hr, 37min) while the difference between its earliest allowed start time and its cutoff time is only slightly longer at 13563 sec (3hr, 46min). Then, assuming the activity starts as early as it can, if it runs longer than expected by even 9 minutes, it will violate its cutoff time and fail to be scheduled. Due to the lack of flexibility given by its execution window, even a small fluctuation from its nominal duration will result in an inconsistent schedule. The most common example of an activity that is tied to a tight execution window in our study is one which has a lighting requirement from science which translates into a tight execution window.

The 7 activities in Figure 2 (b), that cannot occur concurrently due to hardware resource contention (all use the Remote Sensing Mast (RSM)), shows a case of Case 2. The first three MastcamZ activities (Activities 23, 24 and 20) are imaging activities. These activities have execution windows that overlap with those of the next four activities, which all use the Navcam (cameras) in addition to the RSM. Of these four activities, the first two create atmospheric monitoring movies (Activities 26 and 22) while the second two (Activities 25 and 21) generate movies of dust devil activity. These four activities have the same execution window and must all end by the same time and also share part of their execution windows with the previous three activities. Because all of these activities use the same hardware and have highly overlapping execution windows, they must be placed sequentially. As a result, they become fairly constrained in where they can be scheduled, making the risk of failure if these activities run long quite high.

Understanding which activities are more brittle and why has important use cases. For example, such analysis can assist scientists in pinpointing activities that have a high risk of failure and help them understand which constraints might need to be reassessed.

In order to illustrate the proposed advisory system, we use task network TN 2 which contains both common brittleness cases. The constraint-based brittleness analysis highlights two highly brittle activities under Case 1: Activities 9 and 4. It also highlights three activities in Case 2: Activities 25, 26 and 22 (Figure 2 (b)). Let us focus on these five most brittle activities. We can extract the probability mass p_i^u that refers to a symmetric bound around the mean (confidence interval): p_9^u=0.04, p_4^u=0.06, p_{25}^u=0.84, p_{26}^u=0.90 and p_{22}^u=0.92. Let's use these values as our undesirable levels of brittleness, and the starting point of our relaxation approach. We then set the following task network temporal relaxation problem *RN* as the input to Algorithm 2: *TN 2* is the target input *TN;* $A_{Brittle}$={a_9, a_4, a_{25}, a_{26}, a_{22}}; RA^{TC} is set to be the earliest and latest start time of all activities in *TN 2*, including the five most brittle activities; BM^{TH} is set to be the following target brittleness measure, i.e. the target confidence levels for the five target activities, BM^{TH} = {bm_9^{TH}=0.50, bm_4^{TH}=0.50, bm_{25}^{TH}=0.95, bm_{26}^{TH}=0.95, bm_{22}^{TH}=0.95}; and finally, no requirements are set for $bm_{default}^{TH}$.

With that input, Algorithm 2 will suggest the following solution R^{TC}: $ra_9^{TC} \rightarrow -5371$; $ra_4^{TC} \rightarrow -381$; $ra_{26}^{TC} \rightarrow -1866$; $ra_{25}^{TC} \rightarrow -976$; $ra_{22}^{TC} \rightarrow -151$; $ra_{20}^{TC} \rightarrow -816$; $ra_{23}^{TC} \rightarrow -1056$. This solution indicates that a designer should decrease the earliest start time of activities a_9, a_4, a_{26}, a_{24}, a_{22}, a_{20} and a_{23} by the indicated amount. It is interesting to see that the system is relaxing not only the temporal constraints associated with the five most brittle activities, but also two additional activities that play a role in the dependency chain shown in Figure 2 (b). Activities 20 and 23 are the most brittle activities on the left hand side time window, showing the option of making room for activities in the right hand side time window of Figure 2 (b).

If we apply the above suggested relaxations to *TN 2*, we can run the constraint-based brittleness analysis again and analyze impact on the brittleness measures. In doing so, the target confidence levels are meet properly and the brittleness of both the target five activities and the whole task network is reduced to the acceptable values. We believe that this capability can be quite powerful and useful not only for human planners but also for (offline) automated planners in a generated and test/analyze setting.

5 CONCLUSION

In this paper we presented a new approach for measuring and analyzing the brittleness of a temporal plan based on a set of user-specified constraints, as well its application to planetary rovers. We introduced a new metric to measure brittleness of activities in the plan and algorithms to help designers/planners identify the most critical activities that violated the set of user-specified brittleness constraints (herein dynamic controllability constraint, global minimum state of charge constraint, and handover minimum state of charge constraint) with small variations to their duration. This analysis can be overlapped onto a STNU representation of the plan to augment the understanding of the sources of brittleness and the temporal constraints associated. We also present a new advisory system that is capable of suggesting temporal constraint relaxation to adjust the brittleness measure to an acceptable level. We showed results from a selected set of Mars 2020 planetary rover task networks in which two general cases for activity brittleness emerged, mapping to different ways to address them. We advocate that identifying and fixing such key brittle activities is paramount in planetary rover where large uncertainty exists in the environment.

Acknowledgement

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1] G. Rabideau, E. Benowitz (2017) Prototyping an onboard scheduler for the Mars 2020 rover. International Workshop on Planning and Scheduling for Space (IWPSS), Pittsburgh, PA.

[2] W. Chi, S. Chien, J. Agrawal, G. Rabideau, E. Benowitz, D. Gaines, E. Fosse, S. Kuhn, J. Biehl (2018) Embedding a scheduler in execution fora planetary rover. International Conference on Automated Planning and Scheduling (ICAPS), Delft, Netherlands.

[3] D. Gaines, G. Doran, H. Justice, G. Rabideau, S. Schaffer, V. Verma, K. Wagstaff, V. Vasavada, W. Huffman, R. Anderson, R. Mackey, T. Estlin (2016) Productivity challenges for mars rover operations: A case study of mars science laboratory operations, Technical Report D97908, Jet Propulsion Laboratory.

[4] Jet Propulsion Laboratory (2018) Mars 2020 rover mission, https://mars.nasa.gov/mars2020.

[5] J. Agrawal, W. Chi, S. Chien, G. Rabideau, S. Kuhn, D. Gaines (2019) Enabling limited resource-bounded disjunction in scheduling. International Workshop on Planning and Scheduling for Space (IWPSS) Berkeley, California, pp. 7–15.

[6] A. Cesta, A. Oddi, S. F. Smith (1998) Profile-based algorithms to solve multiple capacitated metric scheduling problems, International Conference on Artificial Intelligence Planning Systems (AIPS), pp. 214–223.

[7] M. Wilson, T. Klos, C. Witteveen, B. Huisman (2014) Flexibility and de-coupling in simple temporal networks, Artificial Intelligence 214, 26–44.

[8] A. Huang, L. Lloyd, M. Omar, J. C. Boerkoel (2018) New perspectives on flexibility in simple temporal planning. Inter-national Conference on Automated Planning and Scheduling (ICAPS), Delft, Netherlands, pp. 123–131.

[9] J. Cui, P. Yu, C. Fang, P. Haslum, B. C. Williams (2015) Optimising bounds in simple temporal networks with uncertainty under dynamic control-lability constraints. International Conference on Automated Planning and Scheduling (ICAPS) pp. 52–60.

[10] S. Akmal, S. Ammons, H. Li, J. C. Boerkoel (2019) Quantifying Degrees of Controllability in Temporal Networks with Uncertainty, International Conference on Automated Planning and Scheduling (ICAPS).

[11] I. Tsamardinos (2002) A probabilistic approach to robust execution of temporal plans with uncertainty. Methods and Applications of Artificial Intelligence, pp. 97–108.

[12] J. Brooks, E. Reed, A. Gruver, J. C. Boerkoel (2015) Robustness in probabilistic temporal planning, AAAI Conf. on Artificial Intelligence, 3239–3246.

[13] C. Fang, P. Yu, B. C. Williams (2014) Chance-constrained probabilistic simple temporal problems. AAAI Conf. on Artificial Intelligence, 2264–227.

[14] P. Santana, T. Vaquero, C. Toledo, A. Wang, C. Fang, B. Williams (2016) PARIS: A polynomial-time, risk-sensitive scheduling algorithm for probabilistic simple temporal networks with uncertainty. International Conference on Automated Planning and Scheduling.

[15] T. Vaquero, S. Chien, J. Agrawal, W. Chi, T. Huntsberger (2019) Temporal Brittleness Analysis of Task Networks for Planetary Rovers. International Conference on Automated Planning and Scheduling (ICAPS'19).