Temporal Brittleness Analysis of Task Networks for Planetary Rovers

Tiago Vaquero, Steve Chien, Jagriti Agrawal, Wayne Chi, Terrance Huntsberger
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

Abstract

We propose a new method to analyze the temporal brittleness of task networks, which allows the detection and enumeration of activities that, with modest task execution duration variation make the execution of the task network dynamically uncontrollable. In this method, we introduce a metric for measuring an activity brittleness – defined as the degree of acceptable deviation of its nominal duration – and describe how that measurement is mapped to task network structure. Complementary to existing work on plan robustness analysis which informs how likely a task network is to succeed or not, the proposed analysis and metric go deeper to pinpoint the sources of potential brittleness due to temporal constraints and to focus either human designers and/or automated task network generators (e.g. scheduler/planners) to address sources of undesirable brittleness. We apply the approach to a set of task networks (called sol types) in development for NASA's next planetary rover and present common patterns that are sources of brittleness. These techniques are currently under evaluation for potential use supporting operations of the Mars 2020 rover.

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 environment 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 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 the natural unpredictability of the environment (Rabideau and Benowitz 2017; Chi et al. 2018).

The traditional approach used in planetary rover missions is to add significant temporal margin for execution robustness; however, this can hamper rover efficiency (Gaines et al. 2016). 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 constraints that cause brittleness to temporal unpredictability.

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Temporal constraint networks formalism have been used for representing temporal plans and for evaluating their consistency and robustness. They provide a useful framework for modeling the various temporal requirements for dynamic scheduling and execution in autonomous agents. Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991) are largely used for representing task networks when no uncertainty is present. Simple Temporal Constraints with Uncertainty (STNU) (Vidal and Ghallab 1996) covers the case in which temporal uncertainty exist and some events (time points) are contingents. The Probabilistic Simple Temporal Networks (PSTN) (Fang, Yu, and Williams 2014) handles the cases in which certain contingent constraints can be modeled as probability functions. In planetary rover missions (Gaines et al. 2016), activity duration is hard to accurately predict making STNU or PSTN quite applicable.

Existing work explores the aforementioned temporal formalism to assess and measure a task network robustness. Traditional work evaluates *flexibility* as the aggregate slack in temporal plans to measure how robust a STN is to scheduling disturbance (deviation from nominal case) (Wilson et al. 2014). In principle, the more schedules the task network permits due to time slack, the more adaptive an agent would be to disturbance. More recently, *robustness* is related to 1) the probability of a plan execution to succeed (Tsamardinos 2002; Brooks et al. 2015; Saint-Guillain 2019), 2) the greatest level of disturbance at which the task network is still successfully executed (Cui et al. 2015), or 3) the ability of a plan to withstand unforeseen disturbances (Lee, Ojha, and Boerkoel 2019). These robustness and flexibility metrics are key to providing a sense of whether a given task network would allow adaptation to disturbance. However, the state-of-the-art methods somewhat fall short on providing the intuition for why a task network is robust or not, and the causes for such. For example, if a task network is below a threshold of robustness, the natural question would be: what specific activities and temporal constraints in the task network are highly brittle to unpredictability and causing it to be temporally fragile. Deeply understanding how brittle certain activities are to unpredictability and uncertainty is fundamental to anticipating and addressing fragile execution points that might jeopardize an entire mission.

This paper addresses the challenge of identifying what are the activities that are most sensitive to temporal unpredictability in a dynamic scheduling setting. We propose a method to detect and enumerate activities that, with low levels of uncertainty or delays, make the execution of the task network dynamically uncontrollable, i.e. we cannot guarantee that temporal constraints will not be violated throughout execution. The goal is to provide a more informative analysis that is complementary to existing robustness metrics, as well as fundamental for guiding designers and planners to increase robustness and address undesirable levels of brittleness. More specifically, our contributions are the following:

- A novel method to analyze and measure temporal brittleness of task networks, while detecting and ranking the most brittle activities;
- A tool for mapping brittle activities to task network structure that aims to help designers and autonomous agents to address undesirable levels of brittleness.

The paper is organized as follows. We first provide a background on the target application of the proposed analysis, temporal constraint networks formalism, and how robustness is currently measured in the literature. We then present our brittleness analysis approach with key algorithms, as well as experimental results. Finally, we conclude with some insights and future directions.

Background

In this work we study brittleness analysis in the context of NASA's next planetary rover, the Mars 2020 (M2020) rover. We derive task networks from M2020 *sol types* (Jet Propulsion Laboratory 2018), the current best available data on expected rover operations during a Martian sol. Based on (Chi et al. 2018), we define a task network as a set of activities $A = \{a_i\langle TC, D\rangle ... a_n\langle TC, D\rangle\}$, where: TC is the set of temporal constraints referring to the nominal duration $TC_{i,dur}$, earliest start time $TC_{i,est}$, latest start time $TC_{i,lst}$, and latest end time $TC_{i,let}$; and D is the set of the activity's dependency constraints in the form of $a_j \rightarrow a_k$, meaning a_j depends on a_k, i.e. the end time of a_k must be before the start time of a_j. All activities are mandatory and, therefore, no disjunction is considered. An execution horizon H is also defined for a task network, constraining when all the activities have to finish by. Herein, we focus on temporal constraints only, leaving resource constraints for future work.

The duration of planetary rovers activities can be quite unpredictable (see (Gaines et al. 2016) for analysis of MSL variability) and it is expected that M2020 will be similar with uncertainty associated with $TC_{i,dur}$. Knowledge about activity duration is hard to get a priori, but partial knowledge might exist in the form of lower and upper bounds (set bounded) or a probability density function (pdf). Under those assumptions and constraints, temporal constraint networks are a suitable formalism to represent and analyze the aforementioned M2020 task networks.

Temporal Constraint Networks

In this section we focus on simple temporal networks formalism (STN) and its main generalizations that covers the scope of this work. A short review of the existing generalizations of STNs can be found in (Frank 2018).

STN is one of the most popular formalisms for temporal constraint reasoning (Dechter, Meiri, and Pearl 1991), which is framed as a constraint satisfaction problem over time point variables. An STN is a tuple $\langle T, C\rangle$, where T is a set of time points ($t_i \in T = \mathbb{R}$) and C is a set of constraints $c(t_i, t_j)$ that encode bounds on the differences between pairs of time points in the form $l_{i,j} \leq (t_j - t_i) \leq u_{i,j}$, i.e. $(t_j - t_i) \in [l_{i,j}, u_{i,j}]$. A *schedule* is a specific assignment to all time points variables T. In our M2020 task network each activity $a_i \in A$ would have two corresponding time points representing its start and end time. A special time point $t_0 = 0$ would represent the start time of the plan execution. An earliest and latest start time constraint of an activity a_i would be encoded as a temporal constraint of the form $(t_{i\_start} - t_0) \in [T_{i,est}, T_{i,lst}]$, while the constraint $(t_{i\_end} - t_{i\_start}) \in [T_{i,dur}, T_{i,dur}]$ would represent the duration of the activity being exactly $T_{i,dur}$.

STNU covers the cases in which certain time points are chosen by nature (Vidal and Ghallab 1996). An STNU is also a tuple $\langle T, C\rangle$, but the formalism partitions both sets T and C as follows. $T = T_E \cup T_C$, representing *executable* time points (T_E) determined by the agent and *contingent* time points T_C which are assigned by nature. The constraints set is $C = C_R \cup C_C$, in which C_R represents *requirement* edges, controlled by the agent, and C_C represents *contingent* edges that are not directly controlled. Such contingent edges link an executable time point to a contingent time point, representing uncertainty with respect to the time difference between the points, i.e., the exact value of $(t_j - t_i) \in [l_{i,j}, u_{i,j}]$ is unknown prior to execution. In the M2020 task network, all or some of activities' duration would be encoded as contingent constraints in C_C, also meaning that their end time points are in set T_C. A schedule in this case is an assignment to all time points in T_E. Ideally, that schedule would work for any situation imposed by nature, but in practice that is very restrictive. Instead, we refer to a *strategy* that maps observations to schedules during execution.

PSTN is a natural extension of STNUs in which a probability density functions are associated to contingent constraints, such as those representing uncertain activity duration (Tsamardinos 2002; Fang, Yu, and Williams 2014; Brooks et al. 2015; Santana et al. 2016). In this case, there exists some knowledge about the uncertain duration of the activity as opposed to no knowledge as in STNU. Herein, the contingent constraints C_C are represented as $(t_j - t_i) = X_{i,j}$ where $X_{i,j}$ is a random variable determined by a distribution, pdf (e.g., Normal distribution). Note that an STNU can be generally represented as a PSTN by assuming that all $X_{i,j}$ are uniform distributions. In the M2020 task network case, the duration of certain rover activities (e.g., drive) could be represented as a random variable. We consider those particular cases in our analysis.

Consistency and Controllability Check

An STN is said to be *consistent* iff there exists a schedule (an assignment to the time points) that satisfies all the temporal constraints (Dechter, Meiri, and Pearl 1991). In order to test for consistency, a traditional approach is to run a shortest path algorithm such as Floyd-Warshall (Floyd 1962) which, in the process of finding the tightest constraints implied by C, is able to detect negative cycles. The existence of a negative cycle proves that the given temporal constraints are inconsistent and no schedule can be generated.

In the STNU realm, consistency is not directly applied due to the unpredictable assignment of contingent time points and edges. An STNU relies instead on checking *controllability* (Vidal and Ghallab 1996), which verifies whether an agent can generate a valid/consistent schedule to any situation that may arise in the external world. Controllability theory specifies different levels. An STN is said to be *strong controllable* (Vidal and Ghallab 1996; Vidal and Fargier 1999) iff there exists at least one "universal" schedule that fits any situation: an assignment to the executable time points is guaranteed to satisfy all temporal constraints regardless of the nature's assignments to contingent time points. That is applicable in cases where agents have to compute a schedule offline before making any observations and don't have the opportunity to adapt online. Nevertheless, these cases are not usually practical in dynamic and unpredictable environments. A more practical level would be *dynamically controllable* (Morris, Muscettola, and Vidal 2001; Morris 2014), in which we check whether there is an execution strategy that at any time during execution, the partial sequence executed so far is ensured to extend to a complete solution whatever durations remain to be observed from the contingent activities. In this case, an agent would be able to choose online a valid assignment of executable time points based on observed past contingent time points, without violating any future temporal constraints. Controllability analysis can also be applied in PSTN when bounds can be established from the pdfs, which in turn incurs some risk of contingent constraints falling out of those bounds (Fang, Yu, and Williams 2014). We heavily used the controllability checks in this work to support the analysis of how brittle activities in the task network are wrt dynamic controllability when varying those contingent constraint bounds.

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 (Cesta, Oddi, and Smith 1998; Wilson et al. 2014), robustness is computed by measuring an STN flexibility: a metric that quantifies the aggregate slack in the simple temporal network. In the naive approach to compute a STN flexibility ($flex_N$), Floyd-Warshall's all-pairs-shortest-path algorithm is first used to infer the tightest bounds over the STN's temporal constraints C and, consequently, the earliest time ($est(t)$) and latest time ($lst(t)$) for all time points $t \in T$. The flexibility for scheduling time point t would be defined as $flex_N(t) = lst(t) - est(t)$. The flexibility of an STN S would be then $flex_N(S) = \Sigma_{t \in T} flex_N(t)$ (e.g., $flex_N(S) = 100$ secs). Such a naive approach ignores possible correlations that might exist between starting times of activities which might result in over estimating flexibility (Lund et al. 2017). In order to avoid double counting, Wilson et al. 2014 propose a more suitable approach in which a linear program formulation is used to maximize the sum of the time intervals each time point t can be scheduled, under temporal constraints C. More recently, STN problem is represented as a polyhedron and the flexibility linked to its volume (Huang et al. 2018). 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 (Cui et al. 2015). Moreover, the above methods do not inform where more flexibility is needed if the metric value is below the acceptable range.

Cui et al. 2015 define robustness as the greatest level of disturbance (delay) on an STNU at which it is still successfully executed. In this method, uncertainty is explicitly modelled and considered to compute robustness. Computing robustness is framed as a linear constraint optimization problem to compute the maximum deviation from the nominal case (i.e., width of the contingent bound) on any activity at which the STNU is dynamically controllable. A particular application of this approach is the maximum delay problem, in which the goal is to maximize the minimum width of contingent constraint intervals. The lower bound on the contingent activity duration is set to the nominal duration and the delay is the varying upper bound of the contingent interval. That particular approach would result in a minimum allowed delay in all contingent activities (e.g., all contingent activities can delay at least 10 minutes). If a 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 contingent activity duration fall inside the chosen bounds. The robustness metric in this case would be a probability of execution success (e.g., a given PSTN is 97% robust).

In (Tsamardinos 2002; Brooks et al. 2015), robustness is also quantified as the likelihood that a PSTN will be executed successfully. Herein, a naive robustness measure, $Robustness_N$, is computed by first using a shortest path algorithm to determine the tightest bounds for each contingent constraint $c_{i,j} \in C_C$; then it uses those bounds to compute the area under the corresponding pdf f between those bounds. Applying it to all contingent constraints, the resulting naive metric would be $Robustness_N = \prod_{c \in C_C} \int_{l_{i,j}}^{u_{i,j}} f_{i,j}(x)dx$. The naive approach assumes no correlation between the temporal constraints, and as before, suffers over estimation. To account for interrelation, Brooks et al. (2015) propose a Monte Carlo approach in which the execution of a PSTN is simulated multiple times and the duration of contingent activities is sampled as they are executed. Herein, the simulated execution uses dynamic controllability technique to determine how it should adapt to the sampled values of duration. 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 (Fang, Yu, and Williams 2014) guide their search for a temporal plan based on a user-specified risk bound (e.g., 5% risk). In (Santana et al. 2016), a risk-minimization approach is used to generate a schedule under strong controllability constraints, in which the risk is one of the components of the optimization function. The minimal risk value would indicate how robust the PSTN is.

It is worth mentioning the body of work on sensitivity analysis for scheduling (Stoskov 1991; Ali et al. 2003; Hall and Posner 2004) that studies problem parameters variation (including task duration) and its impact on scheduling performance (e.g., makespan, solvability). In (Stoskov 1991) for instance, authors measure the maximum amount of parameters change that maintains schedule optimality. In this paper we focus on the impact of temporal disturbance wrt *execution* as opposed to scheduling performance.

The aforementioned metrics, success probabilities and risk bounds, provide an intuitive way to evaluate if an STNU or PSTN execution is likely to succeed and whether the entire network is brittle to disturbance or not. However, it is hard to determine potential sources of brittleness when those metrics do not meet a desirable threshold. What activities in particular are culminating in low robustness, and why is that the case? What activities and temporal constraints are most brittle to delays so that one might try to address specific issues in the STNU or PSTN? In the next section we propose a method to address these questions.

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 activity a in the task network. We called the set of contingent activities A_C and the non-contingent/controllable activities A_R, where $A = A_C \cup A_R$ and $A_C = \{a_i\langle TC, D, \mu, \sigma\rangle ... a_n\langle TC, D, \mu, \sigma\rangle\}$.

We propose a brittleness analysis approach for task networks with uncertainty in which 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"), to analyze the temporal brittleness of each contingent activity individually by exploring different disturbance scenarios and evaluating how interrelated an activity brittleness is to other activities disturbances. We then overlay that information onto the task network to analyze why certain activities are more brittle than others. We first introduce the core analysis algorithm.

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 $a \in A_C$ makes the task network become dynamically uncontrollable (or strong controllable if that is the case) under the STNU formalism. 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*).

Deviation here is related to a *sigma multiplier* that is used to determine the lower and upper bounds of an activity's uncertain duration (i.e., a contingent edge in the STNU) around the mean. The maximum sigma multiplier of a contingent activity a_i that makes the network dynamically uncontrollable (u) is referred to z_i^u, where the breaking uncertain duration bounds for that particular activity would be $(t_{i\_end} - t_{i\_start}) = [\mu_i - z_i^u \times \sigma_i, \mu_i + z_i^u \times \sigma_i]$ in the STNU 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. Note that if the mean and sigma are associated to a normal distribution, the sigma multiplier z_i^u is linked to the probability mass under the pdf within $[\mu_i - z_i^u \times \sigma_i, \mu_i + z_i^u \times \sigma_i]$: the smaller that probability, the more brittle is the activity. Herein, we use the sigma multiplier z_i^u as our central *measure of brittleness* and the means to enumerate brittle activities. In what follows we describe the proposed analysis algorithm to compute z_i^u for each target contingent activity.

Algorithm 1: Ceteris Paribus Brittleness Analysis

```
Input : A Task Network $TN$, i.e. a set of activities
        $A = A_C \cup A_R$ and a plan time horizon $H$; and $F_x$
        used to set the uncertainty scenario during the
        individual activity analysis
Output: A list of contingent activities ordered by their
        corresponding $z_i^u$
1 $multipliers \leftarrow \{\}$;
  //Step 1
2 $x^u \leftarrow computeOverallSigmaMultiplier(TN)$;
  //Step 2
3 $y = x^u \times F_x$ ;        //Sets fixed $y$ multiplier
4 for $a_i \in A_C$ do
5     $z_i^u \leftarrow$
        $computeActivitySigmaMultiplier(a_i, y, TN)$;
6     $multipliers.append((a_i, z_i^u))$;
7 end
8 $ordered\_multiplier \leftarrow order(multipliers)$ ;
9 return $ordered\_multiplier$
```

Algorithm 1 is designed to order activities by their degree of brittleness. It is composed of two main steps. In **Step 1**, we detect the single overall sigma multiplier (applied to all the contingent activities edges at once) that makes the network TN dynamically uncontrollable. That overall sigma multiplier will be used in the second step to created specific *ceteris paribus* scenarios when analyzing each individual activity. In line 2, the function $computeOverallMultiplier$ translates our task network TN to an STNU representation and incrementally adds uncertainty around the mean μ of all activities duration by applying a single increasing overall sigma multiplier x ($0 \leq x \leq \infty$) to all contingent activities. At each increment of x we check if the resulting STNU is dynamically controllable (Morris 2014). The (breaking) overall sigma multiplier that makes TN uncontrollable is set to be x^u. An alternative to the aforementioned incremental approach is to use an exponential/doubling search (with unbounded target value) to find x^u (currently implemented in this work). Figure 1 (a) illustrates the process of (a.1) increasing/searching x and (a.2) hitting an uncontrollable state in which one or many contingent activity duration bounds violate the dynamic controllability constraint.

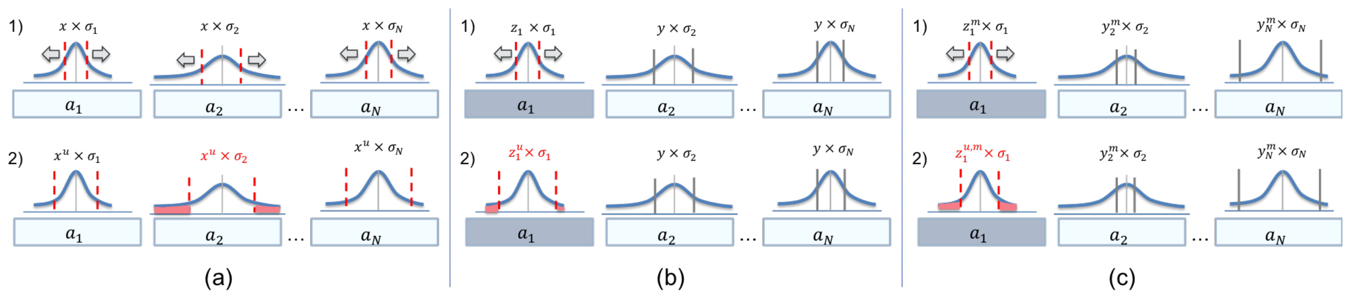

Figure 1: (a) Algorithm 1, Step 1 - Process to determine the overall sigma multiplier x^u that makes the network uncontrollable: a.1) represents the incremental/search process of applying sigma multiplier x to define duration lower and upper bounds to all contingent activities; a.2) represents to point in which the task network becomes uncontrollable and therefore determining the overall sigma multiplier x^u. **(b)** Algorithm 1, Step 2 - Determines the sigma multiplier z_i^u for each target contingent activity $a_i \in A_C$ while applying a fixed sigma multiplier y to all other contingent activities a_j ($j \neq i$): b.1) represents the search process for the maximum sigma multiplier z_i^u for activity a_i; b.2) represents the uncontrollability point in which z_i^u is found. **(c)** Algorithm 2's Monte Carlo simulation process for each contingent activity a_i: c.1) represents the search process for the maximum sigma multiplier $z_i^{u,m}$ in simulation m; c.2) represents the uncontrollability point in which $z_i^{u,m}$ is found.

The most important process of the Algorithm 1 occurs in **Step 2**. Here we start by setting a fixed sigma multiplier y ($0 \leq y < x^u$), line 3) that will be used to specify the *ceteris paribus scenario* (i.e., 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 ($0 \leq F_x < 1$). As we will see later, $y = 0$ (by $F_x = 0$) is a special case of the analysis.

For each activity $a_i \in A_C$ (line 4), we compute its sigma multiplier z_i^u and append the pair (a_i, z_i^u). The function $computeActivitySigmaMultiplier$ (line 5) starts by first creating a STNU in which all other contingent activities $a_j \in A_C (j \neq i)$ has its contingent duration 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 the dynamic controllability constraint, as illustrated in Figure 1 (b). The search for z_i^u can be done in different ways. Similar to Step 1, one can incrementally add uncertainty to activity a_i's duration by applying a sigma multiplier z_i until it hits the breaking point with z_i^u. Another approach would be to use a doubling search to find the breaking point. A more elaborated approach would be to use a relaxation approach, specifically the Conflict-Directed Relaxation with Uncertainty (CDRU) algorithm (Cui et al. 2015) to solve a maximum delay problem over the STNU in a specific way, in which the duration of activity a_i is the only relaxable constraint. The algorithm sets the upper bound of the relaxable contingent duration to a large number and relax that upper bound so that the network is dynamically controllable. For more details on the relaxation approach see (Cui et al. 2015). Finally, every pair (a_i, z_i^u) is stored and later ordered by the value z_i^u (line 8). The ordered list of contingent activity is then returned.

A special case of the analysis in Algorithm 1 is when the fixed sigma multiplier $y = 0$ in Step 2, meaning that 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, i.e. increased values of y can only decrease our brittleness metric for each activity. Figure 2 (a) shows an example of ranking of activity brittleness from one of the studied M2020 task networks. Here we 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. The ranking can inform designers or planners what activities are most brittle and which of them might need further inspection if they do not meet a desirable brittleness level. Note that the chart in Figure 2 (a) can be mapped to maximum delays ($z_i^u \times \sigma_i$) providing actual timing information.

It is worth noting that varying the fixed sigma multiplier y can provide useful information about interrelated contingent activities. If iterate y from 0 to z_i^u and apply Step 2 of our algorithm, we can evaluate the gradual impact of other activities uncertainty to each activity a_i. 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 do changes with varying y, then a_i is interrelated to other activities uncertainty. For cases in which x^u is significantly small, such y varying analysis could be nevertheless uninformative. We describe a complementary analysis in what follows that explores the domain space of y to better analyze interrelation and brittleness in more diverse *ceteris paribus* scenarios.

Monte Carlo Simulation Analysis

Here we explore the impact of an activity's brittleness level given different random duration bounds applied to all others contingent activities in the task network. Specifically, as opposed to using fixed sigma multiplier y across all other activities, we randomly select sigma multipliers y_j for each contingent activity a_j when analysis activity a_i ($j \neq i$) to create random *ceteris paribus* scenarios and explore the domain space of the uncertainty level in the whole task network. In this work, we explore the range $0 \leq y_j \leq W_j$ were W_j is an arbitrary upper bound on the sigma multiplier

Algorithm 2: Ceteris Paribus Brittleness Analysis with Monte Carlo Simulation

```
Input  : A Task Network $TN$, i.e. a set of activities
         $A = A_C \cup A_R$ and a plan time horizon $H$; a number
         $M$ of Monte Carlo simulations per contingent
         activity; and a list $W$ used to bound the sigma
         multipliers when creating a random ceteris paribus
         scenario
Output: A mapping between contingent activities and a list of
         values of $z_i^u$ for each simulation
1  $Z^u \leftarrow \{\}$ ; //initializes the mapping
2  for $a_i \in A_C$ do
3      $Z_i^u \leftarrow \{\}$ ; //initializes the list of $\sigma$
        multipliers for $a_i$
       //Perform Monte Carlo simulations
4      for $m$ from $1$ to $M$ do
5          $Y^m \leftarrow randomSelectYMultipliers(a_i, W, TN)$;
6          $z_i^{u,m} \leftarrow$
             $computeZSigmaMultiplier(a_i, Y^m, TN)$;
7          $Z_i^u.append(z_i^{u,m})$;
8      end
9      $Z^u.append((a_i, (Z_i^u))$;
10 end
11 return $Z^u$
```

assigned to define activity a_j's lower and upper bound contingent duration in the STNU. A straightforward value for W would be x_j^u (which requires first running the aforementioned *ceteris paribus* algorithm with the special case $y = 0$. If normal distributions are associated to the contingent activities in the task network, then one could use values such as $W = 3$, covering 99.7% of the possible values of duration for activity a_j, or $W = 5$ covering 99.9% of the cases.

We propose a Monte Carlo approach to the augment our analysis and explore different *ceteris paribus* scenarios, represented in Algorithm 2. For each activity a_i, we run M Monte Carlo simulations in which we first randomly select a set of sigma multipliers Y^m (line 5), one per contingent activity a_j ($j \neq i$), i.e. y_j^m where $0 \leq y_j^m \leq W_j$. Note that one could run Algorithm 1 with $y = 0$ and use all of some z_i^u for set list W. In addition to selecting Y^m, function $randomSelectYMultipliers$ maps TN to an STNU by applying $y_j \in Y^m$ to each activity a_j's duration bounds and setting a_i duration bounds to $[\mu_i, \mu_i]$ (no deviation). We select Y^m such that the resulting STNU is dynamically controllable (for example using a generate and test approach).

Given a scenario established by Y^m, the algorithm then computes the specific $z_i^{u,m}$ (line 6) for that scenario and simulation m. Figure 1 (c) illustrates that process. The resulting set of sigma multipliers Z_i^u represents activity a_i's variation of the brittleness metric due to varying degrees of uncertainty across the task network. Those sets are then aggregated (line 9) to provide the result of the algorithm Z^u: a mapping between contingent activity and the distribution of its corresponding sigma multiplier.

The data points in Z^u can be plotted to a box-whisker chart, Figure 2 (b) which illustrates how brittle each activity is with respect to the variation of the uncertainty level of others activities. Herein, we can more clearly visualize how dependent (large variation) or independent (no or insignificant variation) the activities are to others.

Mapping Brittleness to Task Network Structure

We implement a tool for representing the outputs of the proposed analysis along side a graphical representation of the temporal network (e.g., as a STNU or a PSTN). The objective is to help a designer to map brittleness to elements in the network structure in order to identify brittleness sources. In addition to the temporal constraints (edges), time points representing the start and end times of activities, and means and sigma, we provide the following information for each activity a_i in the task network:

1. **Brittleness rank**: activity's rank in the brittleness scale, where 1 is the most brittle. Here we use color code to visually represent activity from most to least brittle;
2. **Sigma multiplier**: the z_i^u from Algorithm 1 (with $y = 0$) and the mean, min, max values of z_i^u from Algorithm 2.
3. **Delay**^{u}: refers to $\sigma_i \times z_i^u$ representing the max delay of the activity (using z_i^u from Algorithm 1 with $y = 0$) in which the network becomes dynamically uncontrollable;
4. **Specified execution time window**: the time window specified in the input TN based on the earliest start time and cutoff time. This is to contrast with the inferred time window below.
5. **Inferred execution time window** (Tw_{Exec}^{Infer}): here we run Floyd-Warshall's all-pairs shortest path algorithm to infer the actual available time window for execution due to other temporal constraints. This relates to the flexibility of the activity.
6. **Inferred start time window**: the length of time window in which we can start the activity. This item also relates to the flexibility of the activity start $flex_N(t_{i\_start})$
7. **Dependencies**: we present how many activities need to be scheduled/executed *before* activity a_i, as well as how many activities need to be scheduled/executed *after* the activity. This item provides some some notion on where this activity is on the dependency chain, if any.

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. In the next section we describe how the above added information support identifying the causes on brittleness and the common reasons to why an activity is highly brittle in the set of studied M2020 task networks.

Experimental Results

In this study, we analyze a set of task networks from Mars 2020 sol types and run the proposed *ceteris paribus* analysis and the Monte Carlo approach. By using the analysis, mapping and inspection tool we draw observations on the typical characteristics found with respect to variations on activity's uncertain duration and the corresponding brittleness levels under the dynamic controllability constraint.

Task Network	# Act	# Dep	# Nodes	# Edges	x_i^u	$z_{i,max}^{u,y=0}$
TN 0	33	27	69	193	0.095	1336.77
TN 1	40	43	83	237	0.045	1116.22
TN 2	28	25	59	162	0.060	1767.85
TN 3	18	21	39	104	0.030	3075.725
TN 4	42	42	87	251	0.150	1075.25
TN 5	42	42	87	251	0.135	738.78
TN 6	35	42	73	200	0.030	5239.57
TN 7	46	55	95	279	0.030	980.01
TN 8	20	18	43	115	0.135	896.61

Table 1: Studied M2020 sol types task networks with their respective number of activities (# Act), total number of activity dependencies (# Dep), the number of nodes and edges in the corresponding STNU, the resulting overall sigma multiplier (x_i^u) and the metric for the least brittle activity ($z_{i,max}^{u,y=0}$) in the *ceteris paribus* analysis.

Setup

We run the brittleness analysis using a set of nine M2020 sol type task networks. These networks are representative of what is currently being investigated and applied to develop an onboard scheduler for the M2020 rover (Chi et al. 2018). Table 1 shows some of the main properties of each task network, including the number of activities and dependencies, as well as the resulting number of nodes and edges when translating it to a STNU for controllability check. Due to the fact that the M2020 rover is not yet in operations, accurate models of activity duration variance are not yet available. For the purpose of this study and to illustrate the benefits of the analysis, we use estimate of activities duration as their mean value and select a sigma for each activity randomly as a fraction of the mean, i.e. $\sigma_i = r_i \times \mu_i$ where $0 < r_i < 1$.

For each task network (*TN 1* through *8*), we perform the following: 1) run Algorithm 1 in the special case $y = 0$ to identify the upper bound of $z_i^{u,y=0}$ and plot the brittleness ranking; 2) run Algorithm 2 to analyze the brittleness metric value variation in the Monte Carlo approach, where we set 20 simulation per activity a_i (i.e., $M = 20$); and 3) we use the mapping tool to generate graphical representation (STNU-like) of the task network with the added layer from the brittleness analysis data. For STNU controllability checks in both Algorithms 1 and 2 we used the open-source implementation derived from (Cui et al. 2015).[1]

Given that the analysis is meant to be an offline process, runtime analysis of the algorithms is not the focus on this paper. As a reference, Algorithm 1 usually runs in about 2 minutes in a Ubuntu 16.04 Virtual Machine, 12GB Memory, 3.1 GHz Intel Core i7. Algorithm 2 varies significantly with the number of Monte Carlo simulations and the selected method to chose dynamically controllable scenarios Y^m (line 5).

Results

Table 1 shows the overall sigma multiplier x_i^u for each task network from the *ceteris paribus* analysis, specifically from Step 1 of Algorithm 1. The table also presents the maximum values of the target sigma multiplier $z_i^{u,y=0}$ in the $y = 0$ case, i.e., the least brittle activity value, to illustrate the large range of brittleness in the set. The minimum values of $z_i^{u,y=0}$ (most brittle activities) matches the value of x_i^u provided in the table, except for *TN 7* where $z_{i,min}^{u,y=0} = 0.095$ in showing that in this case the most brittle activity is sensitive to other activities uncertainty with modest variations.

These results shows a brittle set of task networks, given the low sigma multipliers x_i^u, that is: with a small variation of one key activity's duration the task network becomes dynamically uncontrollable. Figure 2 (a) and (c) for example illustrate exactly which activity is the fragile execution element in *TN 2* and *TN 7*, respectively. In *TN 2*, *Activity 9* is the most brittle, and largely more brittle than the other activities. *TN 7* shows an interesting case in which there is a clear highly brittle activity, *Activity 8*, but several other activities are significantly brittle due to a more temporally constrained network with less room for variation in several points.

The results from the Monte Carlo analysis and the mapping approach show two main sources of temporal brittleness in the studied set of M2020 task networks, which falls into two cases: ***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 (or insignificant) variation in the Monte Carlo analysis. That can be also be inspected in the mapping tool where it would 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 variations to the brittleness value in the Monte Carlo approach and 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.

The brittleness of *Activity 9* – a Long Drive activity – in Figure 2 (a) is primarily attributed to its tight execution window 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 (3 hr, 37 min) while the difference between its earliest allowed start time and its cutoff time is only slightly longer at 13563 sec (3 hr, 46 min). 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. Figure 2 (b) shows the long drive activity with no brittleness value variation in the Monte Carlo analysis, while Figure 3 (a) shows the portion of the STNU-like graph with the added analysis results and the key temporal constraints that represent the tight execution window case.

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. Another case would be cutoff times for activities (at cutoff time an activity would be aborted to

[1]Conflict-Directed Relaxation under Uncertainty (CDRU). https://github.com/yu-peng/cdru

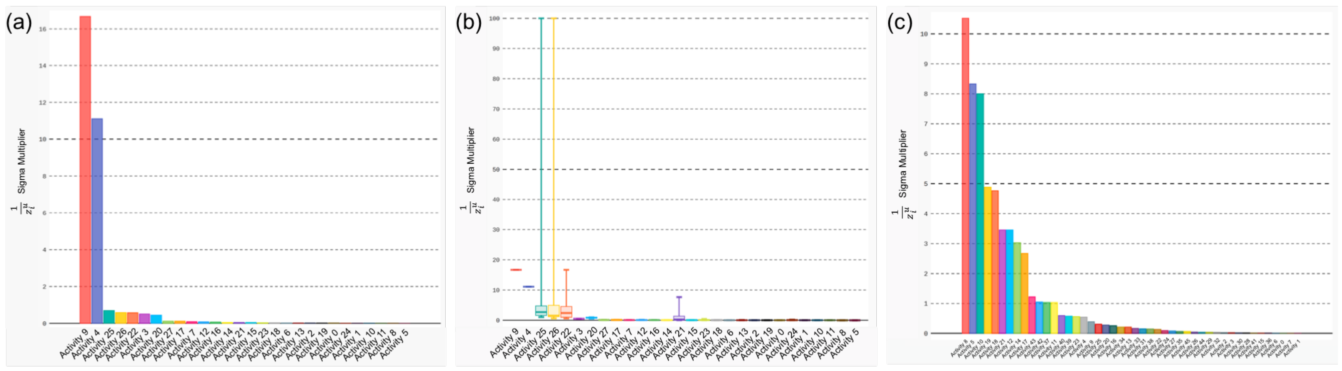
Figure 2: (a) Ranking of brittle activities from *TN 2*. Each activity a_i has a corresponding sigma multiplier z_i^u that represents the temporal deviation bounds in which the network becomes dynamically uncontrollable. A fixed sigma multiplier $y = 0$ is applied to all other contingent activities. The chart shows value of $1/z_i^u$ for each activity. (b) Variation of sigma multipliers Z_i^u of each contingent activity a_i in the Monte Carlo approach from *TN 2*. (c) Ranking of brittle activities for *TN 7*.

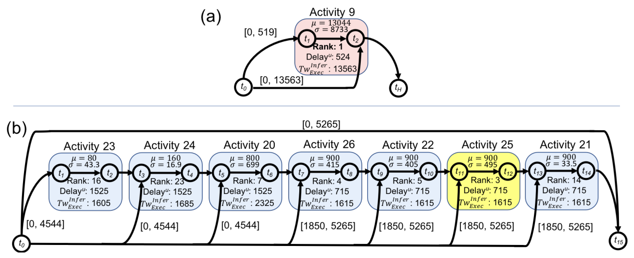
Figure 3: Brittleness due to (a) tight user-specified execution window and (b) to dependencies in *TN 2*.

prevent interference with a later, high priority activity). An example of this would be a science activity cutoff to ensure non interference with a UHF pass – an activity which allows critical data to be sent to and from Earth and is constrained to start only at one particular time when an orbiter is overhead. We expect more illumination requirements from scientists in future sol types which would further constrain the execution windows, resulting in more cases such as this.

We also examine 7 activities in the sol type *TN 2* (Figure 3 (b)) that cannot occur concurrently due to hardware resource contention (all use the Remote Sensing Mast (RSM)), which causes brittleness. 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 next 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 (*Case 2*).

Although the four activities have the same nominal duration, *Activity 25* has a larger sigma (uncertainty) than the others and, therefore, is the most brittle of the four. *Activity 21* on the other hand has less uncertainty and is lower in the rank, but the variations in the Monte Carlo simulations as significant as the other three activities in that chain.

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, whether they be temporal constraints or dependencies. It could also be the case that certain activity duration estimates and/or standard deviations have to be revisited or redesigned (e.g., to reduce uncertainty, sigma). If a more brittle activity was not expected to have high risk of failure, then such analysis can help scientists understand how to reconstruct the plan so that that it is not the case.

Conclusion

In this paper we presented a new approach for measuring and analyzing the temporal brittleness of a temporal plan and application to planetary rovers. We introduced a new metric to measure brittleness of activities in the plan and algorithms to help designers identify the most critical activities that violated the dynamic controllability 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 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 such key brittle activities is paramount in planetary rover where large uncertainty exists in the environment.

Future work includes exploring constraint relaxation mechanisms (Yu, Fang, and Williams 2015; Michel and Hentenryck 2004) to guide designers through the process of reducing a task network brittleness to an acceptable level, as well as studying the brittleness metric in the PSTN realm.

Acknowledgments

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The project was funded through the NSTA Advanced Concept program. The authors would like to thank Dr. Peng Yu for the discussions on the CDRU system, and the reviewers for the valuable feedback and suggestions.

References

Ali, S.; Maciejewski, A. A.; Siegel, H. J.; and Kim, J.-K. 2003. Definition of robustness metric for resource allocation. In *Proc. of the 17th International Parallel and Distributed Processing Symposium (IPDPS)*. France: IEEE.

Brooks, J.; Reed, E.; Gruver, A.; and Boerkoel, J. C. 2015. Robustness in probabilistic temporal planning. In *Proc. of the 29th AAAI Conf. on Artificial Intelligence*, 3239–3246.

Cesta, A.; Oddi, A.; and Smith, S. F. 1998. Profile-based algorithms to solve multiple capacitated metric scheduling problems. In *Proc. of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS)*, 214–223.

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 *Proc. of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*.

Cui, J.; Yu, P.; Fang, C.; Haslum, P.; and Williams, B. C. 2015. Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *Proc. of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 52–60.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Fang, C.; Yu, P.; and Williams, B. C. 2014. Chance-constrained probabilistic simple temporal problems. In *Proc. of the 28th AAAI Conf. on Artificial Intelligence*, 2264–2270.

Floyd, R. W. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5(6):345.

Frank, J. 2018. On controllability of temporal networks: A survey and roadmap. In *International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Integrated Planning, Acting and Execution (IntEx)*, 19–23.

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

Hall, N. G., and Posner, M. E. 2004. Sensitivity analysis for scheduling problems. *Journal of Scheduling* 7(1):49–83.

Huang, A.; Lloyd, L.; Omar, M.; and Boerkoel, J. C. 2018. New perspectives on flexibility in simple temporal planning. In *Proc. of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*, 123–131.

Jet Propulsion Laboratory. 2018. Mars 2020 rover mission. https://mars.nasa.gov/mars2020. Accessed: 2019-03-04.

Lee, J. Y.; Ojha, V.; and Boerkoel, J. C. 2019. Measuring and optimizing durability against scheduling disturbances. In *Proc. of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*.

Lund, K.; Dietrich, S.; Chow, S.; and Boerkoel, J. C. 2017. Robust execution of probabilistic temporal plans. In *Proc. of the 21st AAAI Conf. on Artificial Intelligence*, 3597–3604.

Michel, L., and Hentenryck, P. V. 2004. Iterative relaxations for iterative flattening in cumulative scheduling. In *Proc. of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, 200–208.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI) - Volume 1*, 494–499. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Morris, P. 2014. Dynamic controllability and dispatchability relationships. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems CPAIOR*, volume 8451 of *Lecture Notes in Computer Science*, 464–479. Springer.

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)*.

Saint-Guillain, M. 2019. Robust operations management on mars. In *Proc. of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*.

Santana, P.; Vaquero, T.; Toledo, C.; Wang, A.; Fang, C.; and Williams, B. 2016. Paris: A polynomial-time, risk-sensitive scheduling algorithm for probabilistic simple temporal networks with uncertainty. In *Proc. of the 26th International Conference on Automated Planning and Scheduling*.

Stoskov, Y. 1991. Stability of an optimal schedule. *European Journal of Operational Research* 55(1):91–102.

Tsamardinos, I. 2002. A probabilistic approach to robust execution of temporal plans with uncertainty. In Vlahavas, I. P., and Spyropoulos, C. D., eds., *Methods and Applications of Artificial Intelligence*, 97–108. Berlin, Heidelberg: Springer Berlin Heidelberg.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence* 11:23–45.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In Wahlster, W., ed., *European Conference on Artificial Intelligence (ECAI)*, 48–54.

Wilson, M.; Klos, T.; Witteveen, C.; and Huisman, B. 2014. Flexibility and decoupling in simple temporal networks. *Artificial Intelligence* 214:26–44.

Yu, P.; Fang, C.; and Williams, B. 2015. Resolving overconstrained probabilistic temporal problems through chance constraint relaxation. In *Proc. of the 29th AAAI Conference on Artificial Intelligence*, 3425–3431.