Robust Execution of Probabilistic Temporal Plans

Kyle Lund, Sam Dietrich, Scott Chow, and James C. Boerkoel Jr.
Human Experience & Agent Teamwork Laboratory (http://cs.hmc.edu/HEAT/)
Harvey Mudd College, Claremont, CA
{klund, sdietrich, schow, boerkoel}@hmc.edu

Abstract

A critical challenge in temporal planning is robustly dealing with non-determinism, e.g., the durational uncertainty of a robot's activity due to slippage or other unexpected influences. Recent advances show that robustness is a better measure of solution quality than traditional metrics such as flexibility. This paper introduces the Robust Execution Problem for finding maximally robust dispatch strategies for general probabilistic temporal planning problems. While generally intractable, we introduce approximate solution techniques—one that can be computed statically prior to the start of execution with robustness guarantees and one that dynamically adjusts to opportunities and setbacks during execution. We show empirically that our dynamic approach outperforms all known approaches in terms of execution success rate.

Introduction

Successfully executing tasks in uncertain environments requires close coordination between agents. Consider an automated warehouse where two robots must deliver pallets to a pick station so that a box can be manually packed for shipment. Navigation can take an uncertain amount of time due to slippage and localization error. Further, due to starting in different locations, robot A takes around six minutes to navigate to the pickup, while robot B can arrive in around two minutes. To avoid congestion and ensure efficiency, the robots must arrive at the pick station within two minutes of each other. Clearly, the likelihood of successfully arriving at the pick station at the same time improves if Robot B waits to begin. However, the critical question is: given the durational uncertainty of tasks, how long should Robot B wait to maximize its chance of arriving within 2 minutes of A?

Probabilistic temporal planning provides a framework for representing systems of scheduling constraints, like our example, and captures how those constraints restrict the times at which activities can occur, including characterizing activities with uncertain durations. The challenge that this paper addresses is determining the *best* way to schedule events to account for scheduling uncertainty in probabilistic temporal plans. More specifically, this paper contributes:

- a formal definition of the Robust Execution Problem for finding maximally-robust dispatch strategies to general probabilistic temporal planning problems;
- a new LP-based approximation for determining a static dispatch strategy with strict robustness guarantees;
- the first dynamic probabilistic temporal plan dispatch strategy that opportunistically refines the dispatch strategy in response to environmental dynamism; and
- an empirical comparison that shows our dynamic approach outperforms all known approaches in terms of temporal plan execution success rate.

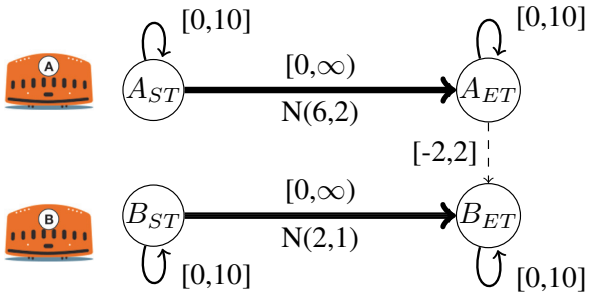


Figure 1: Temporal network for our example. Robot A and B's navigation tasks take ~ 6 and ~ 2 minutes respectively.

Background

Our example scheduling problem is displayed graphically in Figure 1. Each vertex represents an event such as the start and end times for robot A's navigation task, which are represented as A_{ST} and A_{ET} respectively. Directed edges represent temporal constraints and are labeled with the range of time that is allowed to elapse between the occurrence of the events of the source and target vertices. Note the two styles of directed edges: dashed edges highlight constraints between agents (e.g., both robots must end navigation within two minutes of each other), and thick edges convey that the duration is controlled by an uncertain process (e.g., the time to completion of each robot's task). In our example, the robots are given 10 minutes to complete all tasks. This constraint is represented by loops labeled $[0, 10]$ at each event node. The uncertain navigation durations for robots A (approximately 6 minutes) and B (approximately 2 minutes)

are captured by normal distributions ($\mu = 6; \sigma = 2$ and $\mu = 2; \sigma = 1$ respectively). This graphically represents a Probabilistic Simple Temporal Network, defined next.

Simple Temporal Network

A **Simple Temporal Network** (STN) is defined as the tuple $S = \langle T, C \rangle$, where the **timepoints** $t_0, t_1, ..., t_n \in T$ represent $n + 1$ distinct events in the schedule, and each **temporal difference constraint** $c_{ij} \in C$ represents a bound on the elapsed time from t_i to t_j of the form $t_j - t_i \leq b_{ij}$, where $b_{ij} \in \mathbb{R}$ (Dechter, Meiri, and Pearl 1991). The timepoint t_0 is the **zero timepoint**. This is defined to occur at time 0 and grounds the schedule against a clock time. Constraints between timepoints t_0 and t_i implicitly specify when t_i is allowed to occur (i.e., its domain of possible times). The mapping from an STN to a distance graph (e.g., Figure 1) is straightforward. Each timepoint in T maps to a vertex, and each constraint $c_{ij} \in C$ maps to a directed edge from vertex t_i to t_j with the label $[-b_{ji}, b_{ij}]$, i.e., $t_j - t_i \in [-b_{ji}, b_{ij}]$, where b_{ij} is set to ∞ if c_{ij} does not exist. Each timepoint t_i's domain, defined through its constraints with t_0, is represented with a self-loop labeled $[-b_{i0}, b_{0i}]$.

A substantial advantage of temporal networks is that they do not require the combinatorial overhead of planning approaches that discretize time. An STN is **consistent** if there is some assignment of values to the timepoints such that all constraints are satisfied. In a **minimal** STN, the ranges for constraints and timepoints are restricted to exactly those values that could result in a consistent schedule. A minimal STN exactly captures the set of all solutions (consistent schedules) so that a timepoint can be scheduled to occur at any of its allowed values and be guaranteed that there is *some* assignment of the remaining timepoints that satisfies all constraints. A minimal STN can be computed by applying a shortest-path algorithm to the distance graph of an STN. For instance, the constraints between the activities' start and end times in Figure 1 can be made minimal by restricting their intervals from $[0, \infty)$ to $[0, 10]$, the feasible range of time implied by the timepoint domains. A **decomposition** of an STN further restricts the domains of timepoints such that *all* timepoints can be scheduled *independently* to occur at any of their allowed values while satisfying all constraints. For instance, restricting both A_{ET} and B_{ET} to occur between 7 and 9 minutes would decompose the constraint between them, since any values between 7 and 9 would inherently satisfy the constraint that the robots must arrive within 2 minutes of each other.

Models of Temporal Uncertainty

The **Simple Temporal Network with Uncertainty (STNU)** (Vidal and Ghallab 1996) adds a set of **contingent** edges, C_C, to the typical **requirement** edges, C_R (i.e., temporal difference constraints) defined by an STN. A contingent edge, k_{ij}, is one that is not directly controlled by the agent. Instead, some uncontrollable process (e.g., nature) sets the time that elapses between t_i and t_j to a value $\beta_{ij} \in [-b_{ji}, b_{ij}]$ which is unknown prior to execution. Any timepoint with an incoming contingent edge is thus called a **contingent timepoint**, since when it occurs is decided by nature. A timepoint with no incoming contingent edge is an **executable timepoint**, because the agent can decide when to execute it. Contingent (T_C) and executable (T_X) timepoints partition the set of non-zero timepoints ($T = T_C \cup T_X$). In Figure 1, contingent edges are represented with thick lines. Here, the start time of each activity is executable by the robot, whereas the end time is contingent (e.g., due to an unexpected obstacle or low battery). Because it is theoretically impossible for two activities with uncertain, continuous durations to end at *exactly* the same time, the STNU formulation assumes that a timepoint cannot have more than one incoming contingent edge. This limitation can be remedied by constraining two or more contingent timepoints to occur within a synchronization window (e.g., in Figure 1, robots A and B must arrive within 2 minutes of each other).

A **Probabilistic STN (PSTN)** is an extension of the STNU that more precisely models how the durations of contingent edges are selected using probability density functions (PDFs) (Tsamardinos 2002; Brooks et al. 2015). Each contingent constraint $c_{ij} \in C_C$ is of the form $t_j - t_i = X_{ij}$, where X_{ij} is a random variable that is selected according to the PDF P_{ij}. In Figure 1, for example, robot B's uncertain navigation time is modeled by a normal distribution with mean 2 and standard deviation 1.

Measures of Flexibility and Robustness

Flexibility is an aggregate measure of slack contained within an STN. Naïvely, flexibility can be computed by summing together the slack (i.e., size of the domain) of each timepoint in a minimal STN representation. However, this systematically overestimates slack, since scheduling one timepoint can restrict when others can occur. This is corrected by decomposing timepoints before summing their slack to avoid double counting (Wilson et al. 2014). Flexibility has traditionally been used as a measure for how robust an STN is to scheduling perturbations but has the drawback that it is agnostic as to how disturbances arise in practice.

Robustness is a measure of the likelihood that a particular PSTN S can be executed successfully, $P(Success|S)$ (Tsamardinos 2002; Brooks et al. 2015). Naïvely, we can compute robustness by multiplying the amount of probability mass captured by each contingent edge in the minimal representation, $Robustness_N = \prod_{c_{ij} \in C_C} \int_{-c_{ji}}^{c_{ij}} P_{ij}(x)dx$. For instance, in Figure 1, both contingent edges have between 0 and 10 seconds to complete, which capture 99.725% and 97.725% of the probability mass of the navigation tasks for A and B, respectively. Thus, the naïve estimate would predict that agents executing our example should be successful $99.725 \times 97.725 \approx 92.46\%$ of the time.

However, like flexibility, this naïve approach systematically overestimates the overall likelihood of success because it fails to capture how various contingent edges interact. For instance, if robot A arrives earlier than expected, this shrinks the amount of time that robot B can take to navigate. Thus, in practice, if both robots start navigating at the same time ($t = 0$), it is very likely that robot B will arrive much earlier than A will, reducing robustness to only 17.21%. Unfortunately, exactly computing robustness requires evaluating the

definite integral formed by a combinatorial convolution of the PDFs of contingent edges, where the bounds on each integral must capture the complex interactions between constraints! Because exactly calculating robustness quickly becomes intractable except for the smallest, simplest STNs, previous methods have turned to approximation.

Tsamardinos (2002) approximates robustness by making strong assumptions about the nature of contingent edges (e.g., that their PDFs are continuous, time-independent, and that no temporal dependencies can exist between contingent timepoints). These assumptions allow the use of the naïve robustness metric discussed above, and in turn, allows the scheduling optimization problem to be formulated as a linear program (LP). However, such assumptions eliminate many interesting, real-world scheduling problems that include rich networks of interrelated temporal dependencies, including even our simple example problem. Brooks et al. (2015) avoid these limiting assumptions by using a Monte Carlo approach—repeatedly simulating the execution of the PSTN to approximate robustness, sampling from the PDF whenever a contingent edge is encountered, and then reporting the portion that complete successfully.

STN Dispatch

Dispatching is the real-time decision of agents about when to execute their events. For PSTNs and STNUs, this is particularly difficult because the agent does not control (or know) the exact duration of contingent edges until they are received from nature. One approach to dealing with this is to make an STNU **strongly controllable** (Vidal and Ghallab 1996) where any assignment of values to executable timepoints is *guaranteed* to be consistent with all constraints regardless of how nature sets contingent edges. Strong controllability is computed offline by temporally decoupling each executable timepoint from contingent timepoints by restricting its domain to values that are guaranteed to work with all contingencies. An STNU is **dynamically controllable** (Morris and Muscettola 2005; Morris 2014) if executable timepoints can be scheduled online *during* execution (i.e., based on observed values of past contingent edges) in a way that guarantees success.

A controllable STNU is ready to be dispatched. An executable timepoint i is considered **live** if the current time is within its bounds $[-b_{i0}, b_{0i}]$, and it is **enabled** if it can be executed at the present time without violating any constraints (e.g., all timepoints that precede it have been executed). A contingent timepoint can also be live, and is enabled if and only if the timepoint at the beginning of its contingent edge has been executed. However, no dispatch decisions are made; instead an agent **receives** the value of a contingent timepoint from nature. Once an STN has been made minimal, or an STNU has been made controllable, its dispatch strategy falls out naturally—it can be dispatched using an **early execution** policy that executes timepoints as soon as they become both live and enabled.

The only known methods for dispatching either do not generalize to all PSTNs (Tsamardinos 2002) or are satisficing approaches that establish controllable networks within a set failure rate tolerance (Tsamardinos, Pollack, and Ramakrishnan 2003; Fang, Yu, and Williams 2014; Santana et al. 2016). We address these limitations by introducing two novel approximate approaches for *optimizing* the robustness of dispatch strategies across *general* PSTNs

The Robust Execution Problem

An important detail that has been overlooked in previous work is that robustness and dispatch are inextricably linked—a probabilistic temporal network with theoretical guarantees of robustness cannot be used without a dispatch strategy that can help achieve that level of robustness in practice. We note that we can clarify the definition of robustness by parameterizing it to consider not only the input PSTN S, but also a dispatch strategy D: $Robustness(S, D) = P(Success|S, D)$. Only considering the times permitted by the dispatch strategy makes the definition and the computation of robustness much more precise. Without this explicit parameterization, one must either consider all possible dispatch strategies, which the original theoretical definition of robustness does, or make some other assumption about how timepoints are executed (e.g., Brooks et al. (2015) assumed the simple early execution strategy discussed above).

We can now formalize the **Robust Execution Problem (REP)** as finding an optimally robust dispatch strategy D for a given PSTN S:

$$\begin{aligned} \text{maximize} \quad & Robustness(S, D) \\ \text{subject to} \quad & t_j - t_i \leq b_{ij} \ \ \forall \ c_{ij} \in C_R \\ & t_j - t_i \leq b_{ij} \ \ \forall \ c_{ij} \in C_D \end{aligned}$$

where our parameterized $Robustness(S, D)$ is the primary objective, C_R is the set of required constraints as dictated by the original PSTN, and C_D is the set of constraints (possibly dynamically) added by the dispatch strategy. The output of the REP is a dispatch strategy that maximizes the likelihood of success (i.e., robustness) subject to all original constraints. Note that if D is static the set C_D can be completely determined prior to execution (e.g., as in strong controllability), and this is simply a non-linear optimization problem.

Recall that exactly computing robustness is generally intractable for instances of PSTNs. Our optimization above requires evaluating robustness across a combinatorially large space of dispatch strategies. Even if we limit ourselves to the simpler task of computing static dispatch strategies (e.g., ones that attempt to temporally decouple against uncertain edges), computing optimal temporal decouplings is generally NP-hard for non-linear objective functions (Planken, de Weerdt, and Witteveen 2010). Thus, we propose new approximate methods for solving the REP.

The Static Robust Execution Algorithm

In this section, we introduce our Static Robust Execution Algorithm (SREA), an approximate method for solving the REP that produces an offline dispatch strategy. The goal of SREA is to find an optimal dispatch strategy where all scheduling decisions must be made offline *before* execution,

prior to receiving any information about contingent timepoints. Our approach leverages the notion of *strong controllability* for STNUs. We can translate our PSTN into an STNU to utilize an LP formulation, but we must be careful about how we extract bounds for the contingent edges. Maximizing overall robustness requires capturing as much of the probability mass as possible on each contingent edge. To do this, we select an acceptable level of risk, $\alpha \in [0, 1]$, which is the allowable amount of probability mass we are willing to sacrifice in coming up with a dispatch strategy.

Then, for each contingent edge, $k_{ij} \in C_C$, we define $b^{\alpha}_{ij} = F^{-1}\left(1 - \frac{\alpha}{2}\right)$ and $b^{\alpha}_{ji} = -F^{-1}\left(\frac{\alpha}{2}\right)$, where $F^{-1}(x)$ is the inverse CDF of the probability distribution on the edge from i to j. This gives us a bound for the contingent edge that contains $(1 - \alpha)$ of the probability mass. Additionally, we expand each of these bounds by some $\delta_{ij} > 0$: $\left[-b^{\alpha}_{ji} - \delta_{ji}, b^{\alpha}_{ij} + \delta_{ij}\right]$ to heuristically capture as much additional probability mass as possible.

$$\text{Maximize: } \sum_{c_{ij} \in C_c} \delta_{ij}$$

$$\begin{aligned}
\text{Subject to: } & t^+_i \geq t^-_i && \forall\, t_i \in T && (1)\\
& t^+_j - t^-_i \leq b_{ij} && \forall\, c_{ij} \in C_R && (2)\\
& t^+_j - t^+_i = b^{\alpha}_{ij} + \delta_{ij} && \forall\, c_{ij} \in C_C : t_j \in T_C && (3)\\
& t^-_j - t^-_i = -b^{\alpha}_{ji} - \delta_{ji} && \forall\, c_{ij} \in C_C : t_j \in T_C && (4)\\
& \delta_{ij} \geq 0 && \forall\, \delta_{ij} && (5)\\
& t^+_0 = t^-_0 = 0 && && (6)
\end{aligned}$$

Figure 2: LP for finding a strongly controllable STNU that ensures each contingent edge incurs at most α risk of failure.

Our LP finds *new* upper and lower bounds on a timepoint t as t^+ and t^- (thus initially, $t^+_i = b_{0i}$ and $t^-_i = -b_{i0}$). Lines 1-2 ensure that these new timepoint bounds are consistent with all original constraints. Lines 3-4 ensure that the new timepoint bounds are consistent with at least (1-α) of the PDF associated with that contingent edge (represented by b^{α}_{ij} bounds computed above). The δ_{ij} that appear in the objective function to heuristically maximize the additional probability mass captured by each contingent edge.

Theorem 1. *A PSTN that satisfies the constraints of the LP is guaranteed to have a robustness of at least* $(1 - \alpha)^{|T_c|}$.

Proof. Because the STNU output by the LP is strongly controllable, the schedule will succeed if the received value for every edge is within its bounds. By construction, the received value for a contingent edge will fall within its bounds with a probability of at least $1 - \alpha$. Therefore, because there are $|T_c|$ contingent edges that will succeed with probability at least $1 - \alpha$ independent of the execution of other edges, the overall probability of success is at least $(1 - \alpha)^{|T_c|}$. □

Notice that the guarantee of robustness given by Theorem 1 makes no assumption about dispatch strategy, so applies no matter how the agents choose to dispatch the schedule output by our LP. Thus, in our empirical evaluations, we use the early execution strategy on the output from our LP to dispatch schedules. However, whereas the choice of dispatch strategy has a negligible impact, the choice of α significantly impacts performance, as we discuss next.

Algorithm 1: Static Robust Execution Algorithm

Input : A PSTN S, a resolution r, and range $[\alpha^-, \alpha^+]$
Output: The execution strategy with minimum α
if $(\alpha^+ - \alpha^-) \leq r$ **then**
  return LP(S,α^+)
$\alpha' = \frac{(\alpha^- + \alpha^+)}{2}$
if LP(S,α') *is infeasible* **then**
  return SREA$(S, r, [\alpha', \alpha^+])$
return SREA$(S, r, [\alpha^-, \alpha'])$

To maximize the benefit of our approach, we find the minimal α (i.e. maximum guaranteed robustness) such that strong controllability can still be established. We minimize α by running a binary search over the rang $[0, 1]$ for a given resolution r, evaluating the LP at every point, and returning the schedule corresponding to the minimum feasible α. Because we evaluate the LP for every value of α during the binary search, the runtime of our static approach is $O(LP \cdot \log(\frac{1}{1-r}))$ where LP is the runtime of the linear program and r is the resolution at which the range $[0, 1]$ is divided up during the binary search for α. Our search for the optimal α, coupled with our LP that heuristically maximizes the probability mass captured per contingent edge, guarantees that the resulting dispatch strategy is strictly more robust than previous satisficing methods (Fang, Yu, and Williams 2014).

The result of applying SREA to our example problem is shown in Figure 3. The best α level found by the algorithm was 0.506, corresponding to bounds of $[4.66, 7.33]$ on robot A's contingent edge, and $[1.35, 2.67]$ on robot B's contingent edge. Notice that the algorithm expanded robot A's contingent edge very slightly to $[4.64, 7.33]$. The robustness of this modified schedule is now 24.61%, as compared to 17.21% with the early execution dispatch method. This points to the fact that the efficacy of *statically* computing a wait time that optimizes for the expected case is inherently limited in its ability to sync up events that are subject to high amounts of uncertainty. Next we define an approach that can dynamically exploit new information that arises during execution.

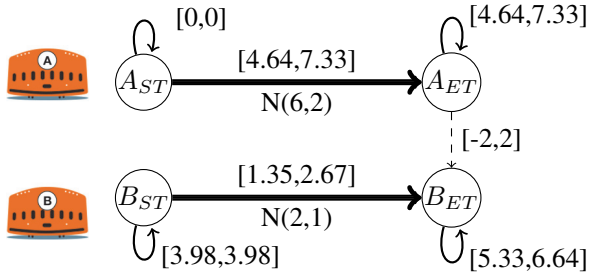


Figure 3: The example of Figure 1 after applying SREA. $\alpha = 0.506$, and the STN robustness is 24.61%.

The Dynamic Robust Execution Algorithm

Our static approach exploits strong controllability to approximate an optimal offline dispatch strategy. It works by limiting the range of executable timepoints so that nature can consistently choose *any* time within the $\left[-b^{\alpha}_{ji}-\delta_{ji}, b^{\alpha}_{ij}+\delta_{ij}\right]$ bounds of each contingent edge, thus ensuring a guaranteed robustness level. However, this approach has two limitations. First, the approach is conservative—even though nature is more likely to pick a value near the mode of its distribution, the $\left[-b^{\alpha}_{ji}-\delta_{ji}, b^{\alpha}_{ij}+\delta_{ij}\right]$ intervals are constructed to include as much of the distribution as possible. The problem with this is that hedging against uncertainty in one portion of the temporal network can inhibit our ability to hedge against uncertainty in other places. Second, if nature does choose a timepoint outside the restricted range (e.g., the process runs unexpectedly long), the strongly controllable network guiding execution simply breaks.

Our dynamic approach is to re-evaluate the LP whenever additional information about contingent constraints is received. The result is an online algorithm that exploits favorable outcomes (e.g., extra time due to a robot finishing unexpectedly early) by readjusting the remaining schedule to increase the guaranteed level of robustness (reducing α). Conversely, if nature chooses an extreme value outside the $\left[-b^{\alpha}_{ji}-\delta_{ji}, b^{\alpha}_{ij}+\delta_{ij}\right]$ interval, our approach attempts to adjust to this unlucky break by gracefully decreasing robustness (increasing α). Not waiting until the controllable STNU breaks before adjusting ensures that schedule dispatch will not fail unless there is no longer any way to satisfy the original constraints. This resolves a question from previous work of what to do if a PSTN grounded as a controllable STNU fails (Tsamardinos, Pollack, and Ramakrishnan 2003). Our approach mimics the idea of dynamic controllability in STNUs, since execution decisions are responsive to environmental dynamism, but as we show later, it performs much better than dynamic controllability in practice.

Our Dynamic Robust Execution Algorithm (DREA), summarized in Algorithm 2, detects when a contingent timepoint is enabled or received and re-evaluates SREA on the schedule. Executable timepoints are executed if they are live and enabled according to the most recent instantiation of SREA. Two things happen any time a contingent timepoint is received or becomes enabled: (1) that timepoint is updated—either assigned to its received value or the corresponding PDF is renormalized if it has not yet been received—and (2) a new set of dispatch instructions is computed using SREA. The speed of this algorithm can be improved by caching the α value between subsequent executions of SREA so that subsequent binary searches can be seeded with this value. Re-evaluating any time a contingent timepoint is enabled even if it is not yet received ensures that executable timepoints that are (or may become) live are informed if a contingent timepoint is taking longer (or shorter) than expected. This allows the agent to opportunistically readjust its schedule to pursue more (or less) aggressive levels of risk, α. Because DREA can dynamically adjust the execution of B_{ST} based on the status of A_{ET}, the expected success rate (computed as described in Experimental Setup) for our example problem (Figure 1) improves to 68.04% as compared to 24.61% when using SREA.

Algorithm 2: Dynamic Robust Execution Algorithm

```
Input : A PSTN S
guideSTN ← SREA(S)
while S.isConsistent() and not S.allExecuted() do
    if any t ∈ T_C is received or enabled then
        S.update(t)
        guideSTN ← SREA(S)
    else
        foreach live & enabled t ∈ T_X according to
          guideSTN do
            S.execute(t)
            guideSTN.execute(t)
```

DREA Example

To illustrate the usefulness of DREA, we walk through a simulated execution of our DREA algorithm applied to the running example.

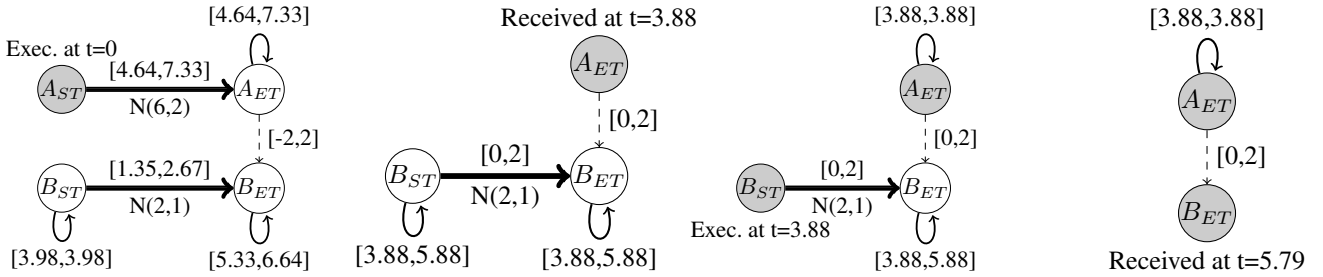
Step 1: First, SREA is applied to the initial problem (Figure 1), which produces the PSTN shown in Figure 3. Dispatch begins by setting A_{ST} to the current time $t = 0$. This produces the PSTN in Figure 4a.

Step 2: We now have an action with an uncertain duration (robot A's navigation) in progress. During every time step that A_{ET} is enabled but not received, the algorithm updates by renormalizing the associated PDF and re-running SREA so that it can shift the recommended start time of any executable timepoints with respect to the latest information. If robot A does not arrive before B_{ST} is scheduled to start, the algorithm would execute it anyway. However, in this particular simulation, robot A arrives in less time than expected, 3.88 minutes, and thus is received before we dispatch robot B. At this point, we run SREA again to produce the PSTN in Figure 4b. Node A_{ST} can be removed in the process because all of its children have been executed and it is no longer relevant to future execution.

Step 3: Once A_{ET} is received, robot B can start navigating early as well, since there is no longer any benefit to waiting. Robot B departs at our new earliest possible time, $t = 3.88$, which produces the PSTN in Figure 4c.

Step 4: Now the only thing left to do is wait for robot B to arrive. In this simulation, B arrives at the pick station in 1.91 minutes, and so we receive B_{ET} at $t = 5.79$ as shown in Figure 4d.

This sample walk-through represents a successful execution because the all requirement constraints were satisfied at every point during execution. Although this example is very simple, and determining the best course of action is



(a) **Step 1:** Set $A_{ST} = 0$ (b) **Step 2:** Receive $A_{ET} = 3.88$ (c) **Step 3:** Set $B_{ST} = 3.88$ (d) **Step 4:** Receive $B_{ET} = 5.79$

Figure 4: Sample execution of our DREA algorithm on our example problem.

relatively straightforward, it shows how DREA can adjust the schedule to take advantage of unexpected situations. In larger examples, where the best course of action is not as easy to determine, dynamically adjusting the schedule is successful in improving robustness.

Empirical Performance Analysis

We compare our approaches against three approaches: (1) the simple **early execution** strategy described earlier, (2) the **static-controllability**-based approach described by Fang, Yu, and Williams (2014), and (3) the **dynamic-controllability**-based approach of Tsamardinos, Pollack, and Ramakrishnan (2003) using the latest dynamic controllability algorithm (Morris 2014). The latter two approaches represent the current state-of-the art. Note, that Fang, Yu, and Williams's approach is a satisficing approach that requires a risk budget as input—we assign risk to its best setting by performing a binary search for the minimal risk budget that still permits a strongly controllable STNU.

Experimental Setup

We adapt the random robot navigation problem generator of Brooks et al. (2015) to generate random PSTNs with varying numbers of timepoints and constraint characteristics. Structurally, each PSTN is composed of several agent subproblems (i.e., one with no concurrent operations) that are connected through interagent constraints with a check to avoid causal loops. The PDFs associated with contingent edges were all normal distributions, with means of less than 10 seconds. The *standard deviations* (σ) of these distributions were systematically varied using values between 1 and 5, which varied the entropy/kurtosis of the contingent edges' distributions. As distributions become wider and flatter, it may become difficult to capture as much probability mass.

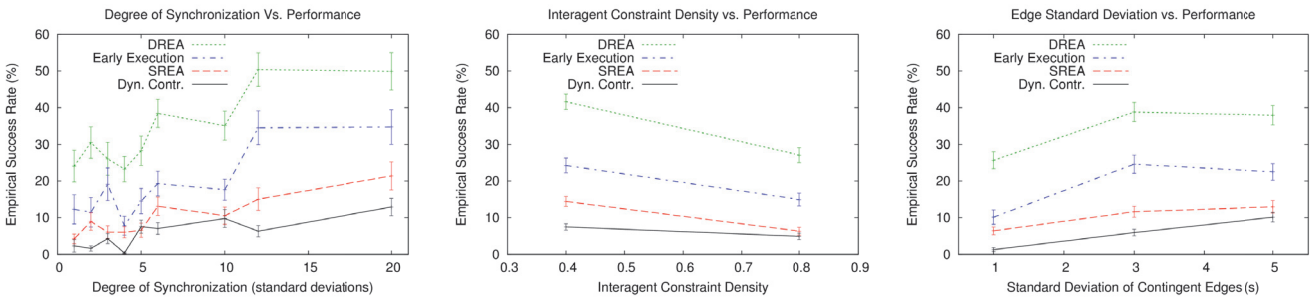
The problems we generated generally have 20 timepoint variables divided among 2 to 4 agents and 20 to 35 constraints depending on the selected interagent constraint density, of which up to 15 are contingent. Requirement constraints were set with a lower bound of 0 and no upper bound, and the total time (makespan) given to complete each schedule was the midpoint between the minimum and maximum possible times of the critical path through the PSTN. We also varied the *interagent constraint density*, which is the proportion of constraints between agents' sequences of actions and which allows us to explore the impact of different degrees of interagent coupling. Finally, the *degree of synchronization* represents the "tightness" of the bounds on interagent constraints, which were set to $[0, n \cdot \sigma]$ with $n = 1, 2, 4$ ($n = 1$ by default). We implemented our approaches in Python using the PuLP linear programming library. For each parameter setting that we tested, we report an empirically derived execution success rate computed by simulating the execution of 20 PSTN schedules 500 times each using a robot navigation simulator (Brooks et al. 2015). The simulation engine sets executable timepoints according to the dispatch strategy being tested. It samples the values of contingent edges as it encounters them according to their PDF, and then assigns the corresponding contingent timepoint once the correct time arrives. We ran simulations on a Linux machine with 96 Xeon E7540 cores.[1]

Analysis

As shown in Figures 5a - 5c, DREA results in execution strategies with the highest success rates across the board, early execution is next, followed by SREA and dynamic controllability. Note, for clarity, we omitted plotting curves for static controllability—a restricted case of both the dynamic and SREA approaches—since it was strictly dominated by all other approaches. Our results demonstrate that DREA effectively balances the agility to respond effectively to dynamism in the environment with the ability to proactively guide agents towards more promising solutions.

The strong performance of early execution, particularly relative to SREA, points to the value of dynamically reacting to, *vs.* proactively hedging against, uncertainty in the environment. Early execution can react to the results of nature assigning contingent timepoints over the course of the experiment, while SREA makes all decisions at the very start, before any contingent timepoints are assigned. Controllability-based approaches like SREA attempt to guide agents away from schedules that are particularly unlikely to succeed by pruning the space of schedules available at execution time. For these problems, the benefits do not appear to outweigh the costs. However, the robustness guarantees provided by Theorem 1 may still be useful in critical situations where complexity or limited communication may preclude using

[1] Simulation code and problem instances available upon request.



(a) Varying Degree of Synchrony (b) Varying Interagent Constraint Density (c) Varying Standard Deviation

Figure 5: Simulated success rates of DREA, Early Execution, SREA, and Dynamic Controllability.

early execution or DREA. Further, the example in Figure 3 demonstrates that these guarantees may become more valuable in problems where waiting and synchronization are critical for success.

Interestingly, SREA consistently outperforms the dynamic controllability approach, albeit not always by a statistically significant margin. Recall that the dynamic controllability approach is dynamic in the sense that it allows execution decisions to be conditioned on events—however, all such conditionals are still precomputed prior to execution. Dynamic controllability appears to constrain the solution space *as much as* SREA, which heuristically tries to recover as much of the solution space as possible. Hence, for this particular space of problems, the benefits of heuristically expanding how much probability mass is captured by each contingent edge outpaces the benefits of dynamic controllability.

The success rates of all approaches improved as the degree of synchrony between agents became less tight (Figure 5a). This is intuitive—as the synchronization windows between uncertain processes increase, all strategies become more robust. As interagent constraint density increases, the success rate of all approaches generally decreased (Figure 5b). This is as expected—more dependencies between agents means more opportunities for failure. At the same time, SREA becomes much more competitive with the early execution strategy. This points to the fact that as the density of constraints increases, SREA's ability to guide agents away from schedules that are unlikely to lead to successful coordinations becomes increasingly valuable.

Surprisingly, the success rates of all approaches increase as standard deviations on distributions grow (Figure 5c). We expected our approaches to benefit from peaky distributions containing dense information about when things were likely to occur. We note that SREA attempts to increase the bounds of contingent edges without regard to the associated increase in probability mass, which works well for symmetrical, (e.g., normal) distributions. We suspect that the performance of our algorithms would be improved if we biased which bounds we increased in distributions exhibiting skew.

We also tracked the computational overhead introduced by our methods in terms of runtime. Across all experiments, SREA took 872 ± 218 milliseconds in expectation to complete each evaluation, which includes resolving the LP for each α value during the binary search over α (with a resolution of $r = 0.001$). Our DREA introduced more overhead per problem instance, with on average approximately 26 seconds per sample over the course of the simulated execution. Recall that DREA works by reapplying SREA many times during execution. As executed timepoints are removed from the problem during simulation, the runtime of each application of SREA progressively decreases, allowing scheduling decisions to be safely dispatched within a half second.

Discussion

In this paper, we defined the Robust Execution Problem for finding maximally robust dispatch strategies in probabilistic temporal planning. We also contributed two approximate methods for solving the REP. The first is an LP-based approximation that finds a static dispatch strategy by minimizing the risk of failure associated with the uncertainty of contingent edges and provides minimal robustness guarantees. Our second approach is the first to dynamically reoptimize as execution unfolds, lowering risk when execution is turning out favorably, and gracefully increasing risk when execution violates expectations. Our empirical evaluation demonstrates that our dynamic approach significantly outperforms all known approaches in terms of execution success rate across a set of randomly-generated PSTNs.

In the future, we would like to evaluate the efficacy of our approaches on real-world scheduling problems including recently published PSTN benchmarks (Santana et al. 2016) and by deploying to real multi-robot systems. Additionally, we would like to generalize and test our approaches on problems that contain the types of uncertainty found in both PSTNs and STNUs (Santana et al. 2016). We would also like explore whether there are improvements that can be made to our LP if it is known *a priori* that replanning may occur dynamically. We believe that robust dispatch strategies may be particularly useful in multiagent settings, and would like to explore decentralized methods for robustly managing multiagent temporal plans. Finally, we would like to explore more explicitly the connection between multi-robot scheduling and the recent advances in constrained multi-robot path planning (Ulusoy et al. 2013; Hönig et al. 2016).

Acknowledgments

Funding for this work was graciously provided by Harvey Mudd College and the Engman Family through the Engman Research and Experiential Learning Fund. We would like to thank Jeb Brooks and Emi Reed for lending their guidance, expertise, and code in this work. We also thank Sam Echevarria, Erin Paeng, Jane Wu, the HMC CS staff and faculty, and the anonymous reviewers for their support and constructive feedback.

References

Brooks, J.; Reed, E.; Gruver, A.; and Boerkoel, J. C. 2015. Robustness in probabilistic temporal planning. In *Proc. of AAAI-15*, 3239–3246.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. In *Knowledge Representation*, volume 49, 61–95.

Fang, C.; Yu, P.; and Williams, B. C. 2014. Chance-constrained probabilistic simple temporal problems. In *Proc. of AAAI-14*, 2264–2270.

Hönig, W.; Kumar, T. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic constraints. In *Proc. of ICAPS-16*, To Appear.

Morris, P., and Muscettola, N. 2005. Temporal dynamic controllability revisited. In *Proc. of AAAI-05*, 1193–1198.

Morris, P. 2014. Dynamic controllability and dispatchability relationships. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 464–479. Springer.

Planken, L. R.; de Weerdt, M. M.; and Witteveen, C. 2010. Optimal temporal decoupling in multiagent systems. In *Proc. of AAMAS-10*, 789–796.

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 ICAPS-16*, 267–275.

Tsamardinos, I.; Pollack, M. E.; and Ramakrishnan, S. 2003. Assessing the Probability of Legal Execution of Plans with Temporal Uncertainty. In *ICAPS-03 Workshop on Planning under Uncertainty*.

Tsamardinos, I. 2002. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence*. Springer. 97–108.

Ulusoy, A.; Smith, S. L.; Ding, X. C.; Belta, C.; and Rus, D. 2013. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research* 32(8):889–911.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proc. ECAI-96*, 48–54.

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