Dynamic Control of Probabilistic Simple Temporal Networks

Michael Gao,* **Lindsay Popowski,*** **James C. Boerkoel Jr.**
Human Experience and Agent Teamwork Lab (heatlab.org)
Harvey Mudd College
Claremont, California 91711
{mgao, lpopowski, boerkoel}@hmc.edu

Abstract

The controllability of a temporal network is defined as an agent's ability to navigate around the uncertainty in its schedule and is well-studied for certain networks of temporal constraints. However, many interesting real-world problems can be better represented as Probabilistic Simple Temporal Networks (PSTNs) in which the uncertain durations are represented using potentially-unbounded probability density functions. This can make it inherently impossible to control for all eventualities. In this paper, we propose two new dynamic controllability algorithms that attempt to maximize the likelihood of successfully executing a schedule within a PSTN. The first approach, which we call MIN-LOSS DC, finds a dynamic scheduling strategy that minimizes loss of control by using a conflict-directed search to decide where to sacrifice the control in a way that optimizes overall success. The second approach, which we call MAX-GAIN DC, works in the other direction: it finds a dynamically controllable schedule and then attempts to progressively strengthen it by capturing additional uncertainty. Our approaches are the first known that work by finding maximally *dynamically controllable* schedules. We empirically compare our approaches against two existing PSTN offline dispatch approaches and one online approach and show that our MIN-LOSS DC algorithm outperforms the others in terms of maximizing execution success while maintaining competitive runtimes.

Introduction

Ideally, an agent would be able to schedule its events and activities in a way that will succeed in every possible situation. However, in many circumstances, there are processes with uncertain timings that make scheduling difficult. The ability of an agent to navigate around this uncertainty is known as *controllability*. While the challenge of controlling for scheduling uncertainty is a well-studied problem, many scheduling problems have more uncertainty than can be controlled, for instance, tasks whose durations are determined by an unbounded probability density function. In such scenarios, our goal becomes maximizing the chance of success.

For example, consider the situation where Mr. X wants to make dinner for his family. Mr. X needs to cook two dishes in his oven, but it is too small to handle both at once. The first takes a duration modelled by a normal distribution of mean $\mu = 20$ and standard deviation $\sigma = 2$ (i.e., $\mathrm{N}(20, 4)$, where variance $\sigma^2 = 4$), and needs to be taken out within 5 minutes of finishing to avoid becoming over-baked. The second, which goes in right after, takes a time drawn from the distribution N(27.5, 9), and again must be removed from the oven in 5 minutes or under. His family will be home in 50 minutes and past experience tells him he has 5 minutes from the time they arrive to the time dinner is served before his hungry toddler has a meltdown. Thus, he must complete dinner prep in between 50 and 55 minutes so it is ready on time without either dish going cold. Problems with probabilistically determined durations like Mr X's can be represented using Probabilistic Simple Temporal Networks (PSTNs).

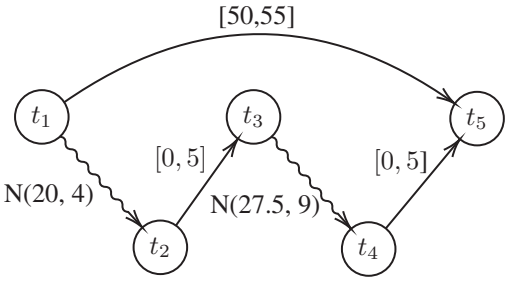

Figure 1: Graphical representation of the Probabilistic Simple Temporal Network corresponding to our running example. Mr. X has between 50 and 55 minutes to prepare dinner, which requires sequentially baking two different dishes, one which is expected to take about 20 minutes to bake and the second which is expected to take about 27.5 minutes.

Before execution starts, Mr. X knows he will not be successful in every possible situation; failure under certain baking times is inevitable under these constraints. A main advantage of PSTNs is the fact that uncertain durations are characterized by distributions, which provide a rich source of information that the dispatcher can leverage to push the schedule towards success. However, the uncertainty of durations in PSTNs may be unbounded, and so the resulting network may be impossible to fully control. This highlights the

*Authors listed alphabetically but contributed equally.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

key challenge we address in this paper: for many interesting real-world problems we may still need or wish to attempt executing a plan, even if full controllability is impossible.

Despite the fact that it may be impossible to guarantee a hot meal on the table with no toddler tears, Mr. X boldly proceeds in making dinner. While he can hope for favorable bake times that do not preclude the possibility of success, he also would like to schedule tasks in a way that tries to maximize chance of overall success. To do this, Mr. X will have to be both reactive and predictive in his scheduling, trying to steer the end of his cooking work towards the desired range of completion times while keeping in mind the possibility for future uncertainty. Unfortunately, he cannot default to simple strategies such as executing everything as early (or late) as possible, since these run the risk of either dinner not being ready or getting cold.

In this paper, we propose two new dispatch strategies for PSTNs that attempt to maximize the likelihood of successfully completing a schedule. Previous attempts to solve this problem have focused on finding a *strongly controllable* schedule prior to execution that approximately maximizes the likelihood of success. Our approaches, on the other hand, are the first known that find a *dynamically controllable* schedule that maximizes the likelihood of success while recognizing the opportunity to take advantage of new information received about events that have been completed.

Our two approaches are complementary. The first approach, which we call MIN-LOSS DC finds a dynamic scheduling strategy that minimizes loss of control by using a conflict-directed search to decide where to sacrifice control in a way that optimizes overall success. The second approach, which we call MAX-GAIN DC, works in the other direction—it finds a dynamically controllable schedule and then attempts to progressively strengthen it by capturing additional uncertainty. We empirically compare our approaches against a current state-of-the-art strongly controllable approach (Lund et al. 2017) and also an existing dynamic control strategy built for a different type of problem that yields an approximate strategy when applied to PSTNs (Nilsson, Kvarnström, and Doherty 2014). We also show that our algorithm performs favorably compared to an approach that recomputes optimal schedules in an online fashion during execution (Abrahams et al. 2019).

Background

In this section, we discuss the building blocks necessary for representing problems such as Mr. X's dinner preparation. We also highlight existing approaches to dispatching schedules in the face of uncertainty.

Simple Temporal Networks

A *Simple Temporal Network* (STN) is a representation of a scheduling problem, with set of temporal events $T = \{t_1, t_2, \ldots, t_n\}$ and set of constraints C, where constraints are of the form $t_j - t_i \leq c_{ij}$ (Deichter, Meiri, and Pearl 1991). A *schedule* is an assignment of times to each event, and is a *solution* if it satisfies all constraints. An STN is called *consistent* if it has at least one solution.

STNs can be represented by directed graphs, similarly to how the Mr. X problem (which is a PSTN, explained later) is depicted in Figure 1, where vertices constitute events and edges constitute constraints. The constraints representing the upper and lower bounds of a duration are often combined into a single directed edge. For example, the arrow from t_1 to t_5 with bounds $[50, 55]$ combines the constraints that dinner prep time must take at least 50 minutes (i.e., $t_1 - t_5 \leq -50$) and at most 55 minutes (i.e., $t_5 - t_1 \leq 55$).

Simple Temporal Network with Uncertainty

A *Simple Temporal Network with Uncertainty* (STNU) is an STN that explicitly considers uncertain durations. The set of constraints is divided into two disjoint sets, the set of *contingent* constraints C_c and the set of *requirement* constraints C_r. Contingent constraints are of the form $t_j - t_i \in [l_{ij}, u_{ij}]$ and represent that the time that elapses between t_j and t_i is chosen by a process that is outside of the control of the executing agent and unknown before execution. Requirement constraints are all the remaining constraints, and are in the same form as those in an STN. In Figure 1, contingent constraints are represented with squiggly lines and capture uncertain bake times whereas the requirement constraints are represented as solid edges and capture constraints such as the fact that dinner must be completed in 50 to 55 minutes.

If an event t_j has only incoming requirement constraints, it is an *executable event*, meaning that its time of execution may be chosen by the agent. Events that are not executable are known as *contingent events*. We therefore divide our set of temporal events into two disjoint sets: the set of executable events T_c and the set of contingent events T_u. In our running example, Mr. X gets to choose events such as the start of bake times thus making t_1 and t_3 executable, but end times t_2 and t_4 are determined by uncertain bake times and thus are contingent.

Controllability STNUs are *strongly controllable* if there is a specific solution that works for *every* possible contingent outcome (Vidal and Fargier 1999). In other words, before execution begins, a time can be assigned to every executable timepoint so that, no matter the timing of contingent timepoints, all constraints are guaranteed to be satisfied. An STNU is *dynamically controllable* (DC) if we can determine a scheduling strategy that guarantees success based on only information about the timing of contingent timepoints that have already occurred (Vidal and Fargier 1999; Hunsberger 2009). Dynamic controllability is much less restrictive than strong controllability, since information about previous, contingent events can be leveraged in decisions about the timing of executable events. Both strong and dynamic controllability were originally defined in the context of STNUs as binary properties of temporal networks, though recent work has proposed a *degree of controllability* metric for determining how close to controllable a network is (Akmal et al. 2019).

Once controllability has been established, the temporal network is ready to be dispatched. *Dispatch* refers to the online process of an agent deciding when to execute its events. Most controllability methods offer mechanisms for compil-

ing a temporal network to dispatchable form. For instance, many dynamic controllability algorithms work by inferring additional constraint edges that reveal implied constraints between edges that were previously not explicitly linked using *wait constraints*, which describe how events should be executed with respect to their predecessors (Morris 2006; Nilsson, Kvarnström, and Doherty 2014). Wait constraints allow for the execution of events to be dynamically conditioned on the outcome of preceding events.

Conflicts are defined as a set of constraints for which no dynamic execution strategy can guarantee success. Said another way, an STNU with no conflicts is guaranteed to be dynamically controllable (Morris 2006). In STNUs, conflicts have traditionally been determined by finding *semi-reducible negative cycles* in an augmented representation of the temporal constrain network (Morris 2006). We can use Mr. X's PSTN to illustrate the concept of a conflict. While it is true that some realization of the two uncontrollable events can be satisfied with a dispatch strategy, there are still many possibilities for failure. If the two dishes both take two standard deviations longer than expected (24 and 33.5 respectively), then no matter how Mr. X schedules his cooking, he could not satisfy the overarching constraint of finishing everything under 55 minutes. The dishes both finishing at two standard deviations below their expected values would likewise cause Mr. X to always violate the constraint of taking at least 50 minutes.

Probabilistic Simple Temporal Network

A *Probabilistic Simple Temporal Network* (PSTN) extends the formalism of an STNU to include additional information about the realization of contingent edges. For each contingent edge, the elapsed time between t_j and t_i is determined by a random variable X_{ij} whose value is determined by a probability density function P_{ij} (Tsamardinos 2002). These probability density functions can be unbounded, for instance when contingent edges are determined by normal distributions such as in Figure 1, and therefore the contingent edges they govern are no longer guaranteed to occur within some finite interval $[l_{ij}, u_{ij}]$.

Controllability Extending the concepts of STNU controllability to PSTNs is natural and straightforward when all probability density functions are bounded. However, for many realistic problems, such as our running Mr. X example, the duration may be more accurately determined by an unbounded distribution. In such cases, completely controlling for all uncertainty becomes inherently impossible and conflicts can become both prevalent and inevitable. Typical approaches for dealing with this have relied on approximating the PSTN with an STNU by establishing bounds for each contingent edge and then establishing a strongly controllable schedule on the resulting STNU (Fang, Yu, and Williams 2014; Santana et al. 2016; Lund et al. 2017). These bounds are commonly determined according to a risk budget α, which establishes how much probability mass per contingent edge is allowed to be sacrificed (often symmetrically from each tail) in order to create a bounded contingent edge.

In the case of Mr. X, the first dish takes a time to cook that is best modelled by a probability distribution, N(20,4), and the second dish by N(27.5,9). However, if we set a risk budget of $\alpha = 0.05$, we can convert the original PSTN into an STNU where the first takes between 16.1 and 23.9 minutes and the second takes between 21.6 and 33.4 minutes. If the resulting network were strongly controllable (which it is not), we would expect this schedule to succeed 90.25% (0.95×0.95) of the time.

Related Work

Next, we highlight the two approaches that are most closely related to our approach of using dynamically controllable STNUs to maximize the likelihood of execution success for PSTNs. These will form the primary points of comparison for our empirical analysis.

SREA SREA, the Static Robust Execution Algorithm (Lund et al. 2017), works in two stages to reduce a PSTN to a strongly controllable STNU. The first step of SREA is a binary search for the minimal value of a risk parameter α that yields a strongly controllable STNU. α determines how contingent PSTN edges are converted to contingent STNU edges, where for each contingent PSTN edge, $\frac{\alpha}{2}$ of the probability mass is trimmed from each end of the distributions to establish STNU-style bounds on the contingent edge that capture $(1 - \alpha)$ of the probability mass. The controllability is determined by the second stage of the process, which is an LP that maximizes how much the bounds of contingent edges can expanded while maintaining a controllable STNU. As a whole, SREA shrinks the allowable ranges for contingent edges across the entire network, and then re-expands bounds back out on individual contingent edges to maximize the overall probability captured to the best of its ability. The result is a schedule where all controllable timepoints are assigned a specific time. For instance, if we apply SREA to our running example, it finds that the optimal risk level is $\alpha = 0.406$. The SREA LP then expands these intervals as much as possible to result in a strongly controllable STNU where the first dish takes between 16.7 and 21.7 minutes and the second takes between 22.6 and 32.4 minutes. This strongly controllable strategy succeeds 75% of the time.

We chose SREA as a primary point of comparison since it explicitly attempts to optimize for probability of success. The other related approaches (Fang, Yu, and Williams 2014; Santana et al. 2016) use a similar risk-based STNU approximation approach, but then look to optimize for a different criterion, making SREA the best point of comparison. SREA has since been expanded to an algorithm called Dynamic Robust Execution Algorithm with Mitigation (DREAM), which reruns the SREA optimization online during execution in hopes of better responding to how uncertainty has been resolved.

DREAM recognizes that re-running SREA any time new information arrives can be expensive. By parameterizing how often schedules are recomputed and re-communicated to mitigate the high computational overhead, DREAM provides a graceful trade-off between SREA, an algorithm that never recomputes when there is new information, and DREA, the one that *always* recomputes in the original work

by Lund et al. (2017). In the case of the Mr. X problem, DREA achieves 89% robustness, a significant improvement on the performance of SREA that DREA achieves by reoptimizing the schedule once the first bake time is realized. As shown by Abrahams et al. (2019), while DREA achieves significantly higher success rates than SREA, it does so with significant computational overhead. In contrast, this paper focuses on dynamically-controllable approaches that can be computed *offline* prior to execution.

Dynamic Control Dispatch (DC-Dispatch) The basic dynamic dispatch algorithm that we use is the early execution dispatch with inferred edges designed by Nilsson, Kvarnström, and Doherty (2014) to dispatch the uncontrollable PSTNs. This dispatcher works on controllable STNUs, and as shown by Akmal et al. (2019), can, but is not guaranteed to, work on uncontrollable ones. The strategy works by generating additional (wait) constraints and using them to help decide when a contingent timepoint is enabled and can therefore be executed. However, if these generated constraints cannot all be satisfied, as in the case of some uncontrollable STNUs, the execution will fail, even if perhaps all of the real constraints could be satisfied in that particular realization. When we apply DC-DISPATCH to our running Mr. X example by first extracting an STNU with risk $\alpha = 0.05$ ([16.1,23.9] and [21.6,33.4] respectively), it succeeds 48% of the time.

Expected-value PSTN (EPSTN) The Expected-value PSTN (EPSTN) representation (Frank 2019) extends PSTNs with a value function that weights which bounds to sacrifice when establishing strong controllability on over-constrained EPSTNs. Frank provides some sophisticated linear-approximations for PDFs that allows solving the problem of finding strongly controllable schedules using a Mixed Integer Linear Program. The complexity of this approximation grows exponentially in the number of constraints. In contrast, our approaches operate on PSTNs, which assume required edges to be hard constraints, and *efficiently* find *dynamically* controllable schedules, as we explain next.

Dynamic Controllability for PSTNs

In this section, we highlight our two new dispatch algorithms for PSTNs. Both are approximate methods that work by attempting to find an STNU that captures the probability density across contingent edges while maintaining DC.

Min-Loss DC

The MIN-LOSS DC algorithm, summarized as Algorithm 1, works in a two-step process. First, it extracts a bounded contingent edge from each probabilistic contingent edge in the original PSTN using a procedure we call EXTRACT-STNU-EDGES(E^P, α). EXTRACT-STNU-EDGES takes in a set of probabilistic edges and a risk budget α, which represents the maximum risk of violating any constraint that we are willing to accept, and returns a set of bounded STNU edges, where each (potentially unbounded) probabilistic contingent edge has been converted to a bounded STNU edge by truncating

Algorithm 1: Min-Loss DC

Input : A PSTN $S = \langle V, E \rangle$ and a risk tolerance α
Output: An approximately optimal DC STNU
$E^U \leftarrow$ EXTRACT-STNU-EDGES$\big(E^P, \alpha\big)$;
if not DC-CHECK$\big(\langle V, E^U \rangle\big)$ **then**
  $E^U \leftarrow$ OPTIMAL-DC-RELAX$\big(\langle V, E^U \rangle\big)$;
return $\langle V, E^U \rangle$;

$\frac{\alpha}{2}$ of the probability density from the each tail of the distribution. The result is a set of STNU edges, where for each edge, the probability that the realized value will fall between the bounds is guaranteed to be at least $(1 - \alpha)$. Note that because we truncate equal mass from each tail, this procedure naturally somewhat accounts for distributions with various skews, kurtoses, or modalities.

Second, if the resulting STNU is DC, we execute with the DC-DISPATCH strategy mentioned earlier. If it not DC, then we apply the Optimal DC Relaxation algorithm of Akmal et al. (2019), which returns the STNU that corresponds to minimally "relaxing" the problem, in this case by squeezing the intervals associated with the contingent edges involved in the conflict. The terminology "relax" is used to reflect that we reduce the amount of uncertainty we need to control (by contracting, rather than expanding, contingent interval bounds), which makes the controllability problem *easier* to solve. MIN-LOSS resolves conflicts one at a time, so it only sacrifices probability mass on the side of the uncertain distribution that is needed. The Optimal DC Relaxation algorithm is optimal in the case of a single relaxation for uniform distributions, but optimality is no longer guaranteed when we have different probability distributions or when multiple relaxations must be performed. Once a dynamically controllable STNU is produced, the Dispatch-DC method can be used to dispatch the network.

In the case of Mr. X, using a risk budget of 0.05, the MIN-LOSS algorithm first restricts each contingent edge to cover 95% of the probability distribution, restricting the first dish to take approximately between 16.1 and 23.9 minutes, and the second between 21.6 and 33.4. Then the optimal relaxation algorithm takes over and restricts this STNU to a dynamically controllable one, where the first dish can now only take from 21.6 to 22.6 minutes, and the second only between 25.9 and 32.3 minutes, as shown in Fig 2. The resulting dynamically controllable dispatch strategy succeeds 74% of the time in expectation. Note, this is much better than the 48% success rate of DC-DISPATCH, but slightly less than the 75% success rate of the strong-controllability-based SREA. This is an artifact of the simplicity of the network—SREA only attempts to pick the time for t_3 that leads to the highest success rate in expectation, which is relatively straightforward. However, as we show in our Empirical Evaluation, in more complex networks DC-based approaches tend to outperform SC-based approaches because there are more opportunities to dynamically recover and exploit slack.

The MIN-LOSS DC described here affords a natural pa-

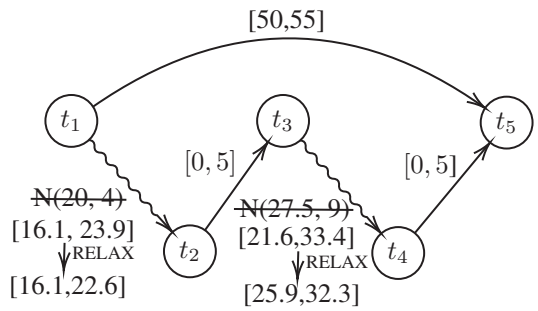

Figure 2: Illustration of MIN-LOSS DC on our running example. First, probabilistically determined bounds are replaced by with intervals capturing $(1-\alpha)$ of the probability mass. Second, any edges involved in conflict are minimally relaxed until the resulting STNU is DC.

rameterization based on the initial risk level α. We suspect that setting α too high will result in riskier edges and unnecessarily sacrifices too much likelihood, hurting overall performance. However, setting α too low might also hurt performance, since it will require more relaxations of the STNU edges. While the Optimal DC Relaxation algorithm attempts to relax the network optimally, it does so on STNU edges that are agnostic to the underlying probability density functions. We explore how α impacts our performance in our empirical analysis.

Max-Gain DC

Our MIN-LOSS DC algorithm starts by extracting an STNU that captures a sufficiently large amount of probability, and then attempts to minimize the additional loss of probability to convert the STNU to one that is DC. Our MAX-GAIN DC algorithm, on the other hand, works by building an STNU that tries to maximize the probability density captured along each contingent edge of the original PSTN.

MAX-GAIN DC, summarized as Algorithm 2, builds an STNU in the following way. It starts by finding what we call the *riskiest conflict* using a binary search over α, which is defined the same way as before. During the binary search over α, we use the same EXTRACT-STNU-EDGE procedure as discussed in MIN-LOSS and perform a DC-check on the resulting STNU. When the DC-check returns true, we continue searching for less risky bounds. Otherwise, we extract and record a conflict, and then proceed with a riskier α, which sacrifices more probability mass, making the uncertainty easier to control. By the time we reach the end of our binary search (when $(\alpha^+ - \alpha^-) \leq r$ for some minimal search resolution r), we will have found a transition between controllable and uncontrollable setting of α: the α^- that corresponds to the riskiest α that still causes a conflict as well as the set of edges E^C involved in that conflict, and α^+, the least risky α that remains DC and therefore does not cause conflicts.

We now know that the lowest we can set α without causing a conflict is α^+. We also know that lowering α beyond this point will result in a conflict involving edges E^C. Thus,

Algorithm 2: MAX-GAIN DC Search

Input : A PSTN $S^P = \langle V, E^P \rangle$, an STNU $S^U = \langle V, E^U \rangle$, a resolution r for the binary search, range $[\alpha^-, \alpha^+]$, and a set of edges E^C
Output: An approximately optimal DC STNU
if $E^P = \{\}$ **then**
  return S^U ;
if $(\alpha^+ - \alpha^-) \leq r$ **then**
  $E^P \leftarrow E^P - E^C$;
  $E^U \leftarrow E^U \cup$ EXTRACT-STNU-EDGES (E^C, α^+) ;
  return MAX-GAIN DC$(S^P, S^U, r, [0, \alpha^+], \{\})$
$\alpha' \leftarrow \frac{(\alpha^- + \alpha^+)}{2}$;
$E^{\alpha'} \leftarrow$ EXTRACT-STNU-EDGES(E^P, α') ;
if DC-CHECK$\left(\left\langle V, E^U \cup E^{\alpha'} \right\rangle\right)$ **then**
  return MAX-GAIN DC$(S^P, S^U, r, [\alpha^-, \alpha'], E^C)$
else
  $E^C \leftarrow$ EXTRACTCONFLICT$\left(\left\langle V, E^U \cup E^{\alpha'} \right\rangle\right)$;
  return MAX-GAIN DC$(S^P, S^U, r, [\alpha', \alpha^+], E^C)$

in the STNU we ultimately return, edges E^C should be set to α^+, which is the most risk we can handle for these edges. Thus, we remove the PSTN edges corresponding to E^C from the input PSTN, then for each edge in E^C, extract a new STNU edge at the α^+ risk level, and finally add these new edges to the output STNU that we are building to approximate the original PSTN input. The remaining edges of the PSTN were not part of any conflicts and so could potentially be set to less risky values (i.e. capture a greater portion of the likelihood). Thus, we recursively proceed to find the next riskiest conflict by restarting our binary search over the remaining PSTN edges. This process continues until all PSTN edges have been converted to STNU edges at their least risky, dynamically controllable level. The end result is a dynamically controllable STNU that approximates the original PSTN by capturing as much probability mass per contingent edge as possible. Like MIN-LOSS, the resulting DC STNU can be put in dispatcable form and subsequently dispatched using the Dispatch-DC algorithm.

We illustrate MAX-GAIN DC on our running example in Figure 3. Since both contingent edges are involved in the same conflict, the binary search runs once, finding the least risk budget that afford dynamic controllability. Note that unlike MIN-LOSS DC, the risk budget is consistent across all contingent edges within a conflict. This means the bounds on each contingent edge within a conflict are set so that they are each equally likely to fail, and for each individual constraint, the risk of failure is symmetric, meaning equal likelihood is sacrificed on either side of the distribution. The dynamically controllable network output by MAX-GAIN on the Mr. X example is successfully dispatchable 69% of the time in expectation.

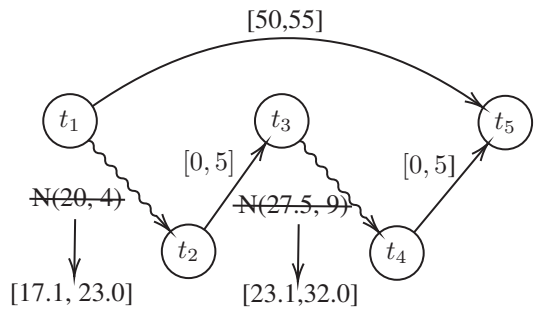

Figure 3: Illustration of MAX-GAIN DC on our running example. Since both contingent edges are involved in the same conflict, the binary search of α will take place once, expanding both sides of the distribution to capture approximately 86% of the probability mass for each contingent edge.

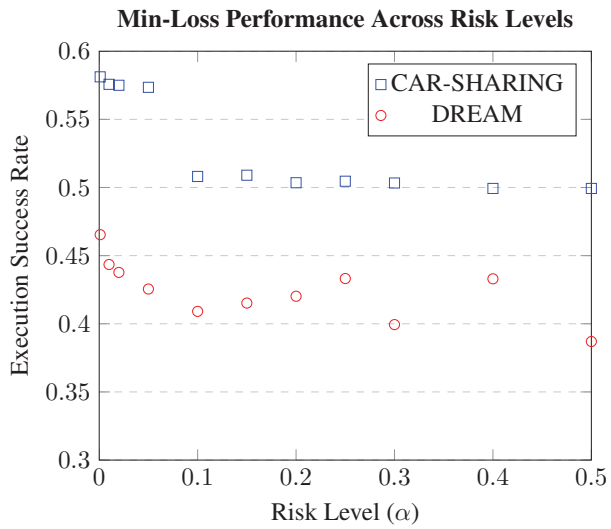

Figure 4: MIN-LOSS DC execution success rate across different risk levels (α). Generally, MIN-LOSS DC performed at its best when risk was minimized.

Empirical Evaluation

We evaluate our methods on two PSTN benchmarks. The first is the DREAM benchmark of 540 PSTNs (Abrahams et al. 2019), where each instance has 21 events and multiple agents. Of this dataset, 85 had no success under any dispatch strategy, and were therefore omitted from the rest of our data analysis. The second is the adapted version of the CAR-SHARING dataset originally from Santana et al. (2016), which was edited by Akmal et al. (2019) to be to be consistent but not fully controllable. The original CAR-SHARING dataset contained a mix of PSTN- and STNU-style contingent edges. Akmal et al. (2019) translated this into a benchmark containing pure STNUs by converting PSTN-style contingent edges to STNU-style contingent edges whose bounds encompassed two standard deviations on either side of the mean. We create an equivalent benchmark of pure PSTNs by running this conversion process in reverse. We tested on 169 different PSTNs, 14 of which no dispatch strategy was ever able to successfully dispatch, and thus were omitted from our analysis. For each of the methods that we tested—MIN-LOSS, MAX-GAIN, SREA, DREA, and DC-DISPATCH—we simulated dispatch at 200 times per problem instance, sampling randomly according to the probability distribution of each contingent edge. All code and problem instances are available at http://github.com/HEATlab/DCforPSTN.

Finding an Optimal Min-Loss DC Risk Level

In order to maximize the performance of MIN-LOSS, we first need to determine what risk level (setting of α across all edges) resulted in superior performance. We tested on both the DREAM and CAR-SHARING data sets and found that the general trends were consistent between the two data sets. The results are summarized in Figure 4.

Interestingly, we found that MIN-LOSS performed best when set to the lowest risk level we tested, $\alpha = 0.001$, which corresponded to extracting an STNU where the bounds over uncertain edges each capture 99.9% of the likelihood. This ran a bit counter to our intuition—we expected setting α too low might hurt performance because it puts more responsibility on the probability-blind OPTIMAL-DC-RELAX algorithm. However, in practice, this did not seem to hurt performance. We suspect this is due to a few reasons. First, since OPTIMAL-DC-RELAX tries to minimize how much it moves bounds, the probability being lost is most commonly in a tail of the distribution, and thus may only have a negligible effect. Second, when conflicts exist, there might not be much freedom in dividing up how uncertain bounds are relaxed. Third, the uncertain edges of both data sets contain a mix of normal and uniform distributions. The majority of the distributions were normal distributions, and their variances are also quite similar. Since these distributions were symmetric and had fairly consistent variances, there might be less to gain from being selective about what parts of the distributions were omitted in the relaxation step. We do not believe that the optimal risk level is as low as possible for every dataset; the optimal value most likely depends heavily on the distributions. In future work, we hope to explore how MIN-LOSS DC performs on problems that contain a larger diversity of distributions (e.g., varying skew, kurtosis, etc.).

Empirical Comparison

We evaluated MIN-LOSS and MAX-GAIN in comparison to SREA, the Static Robust Execution Algorithm (Lund et al. 2017), which employs a static execution method on uncontrollable STN problems, as well as DC-DISPATCH, the dynamic controllability algorithm taken from Nilsson, Kvarnström, and Doherty (2014) and also used by Akmal et al. (2019). All approaches compute a dispatch network in an offline fashion, prior to execution. SREA establishes a strongly controllable schedule, and thus its dispatch strategy is not at all contingent on how uncertain edges are resolved. The other three methods use dynamic controllability, and thus dispatch decisions can depend on the uncertain outcomes of preceding events. Finally, we also compare against

Method	Wins	Success Rate	Runtime (milliseconds)
Min-Loss DC	**207.6**	**0.46**	43.0
Max-Gain DC	108.1	0.37	100.8
DC-Dispatch	76.1	0.32	**10.8**
SREA	16.2	0.22	307.2
DREA	47.1	0.32	1966.9

Table 1: Performance of dispatch strategies for DREAM benchmark.

Method	Wins	Success Rate	Runtime (milliseconds)
Min-Loss DC	**54.3**	**0.57**	161.2
Max-Gain DC	17.4	0.50	1164.2
DC-Dispatch	38.4	0.50	**41.5**
SREA	20.5	0.52	7430.1
DREA	24.4	0.55	10911.4

Table 2: Performance of dispatch strategies for CAR-SHARING benchmark.

one online approach, DREA, the Dynamic Robust Execution Algorithm (Lund et al. 2017) that reruns SREA every time new information is received.

The results of our empirical comparison across the five algorithms are available in Tables 1 and 2. Success Rate was determined by the number of times the dispatch resulted in a valid schedule for a particular PSTN, also known as robustness. "Wins" occur when one algorithm has higher robustness on a particular PSTN instance; an n-way tie counts as $\frac{1}{n}$ of a win for each of the tying algorithms. Finally, we tracked the average time it took to compute the dispatchable form of the schedule. All differences in success rate were statistically significant with $p < 0.05$ for the DREAM benchmark *except* for the difference between DREA and DC-DISPATCH. On the CAR-SHARING benchmark, the only statistically significant differences were that both MIN-LOSS DC and DREA had statistically significant higher success rates than either DC-DISPATCH or MAX-GAIN DC.

We start by comparing approaches in terms of execution success rate. In both datasets, MIN-LOSS DC demonstrates the highest overall success rate and the highest win rate, meaning its gains in successful execution were robust across a wide diversity of problem instances.

Interestingly, the margin of improvement for MIN-LOSS was greater on the DREAM benchmark (Table 1) than the CAR-SHARING benchmark (Table 2). We believe that the difference is due to the nature of the two problem sets. The DREAM benchmarks contained a larger proportion of what we call *synchronization constraints* between agents' schedules whereas in the CAR-SHARING benchmarks tended to have more resource *handoff constraints* between agents' schedules. The main difference is that synchronization constraints have tight upper and lower bounds (e.g., the constraint that Mr. X must prepare dinner within 50-55 minutes), which means uncertainty can cause 'double-sided' problems—tasks can finish too early *or* too late. Resource hand-off constraints (e.g., agent X needs to finish using resource Y before passing it off to agent Z), on the other hand, tend to only lead to 'one-sided' problems—tasks that run over deadline. The wait-constraints of DC-based approaches tend to be more useful situations where there is a risk of beginning tasks too early such as in the DREAM benchmark, which perhaps why we see better overall performance for DC-based methods in Table 1 than Table 2.

At first, we were a bit surprised that MAX-GAIN DC did not perform more consistently with (or even outperform) MIN-LOSS DC. However, upon closer inspection, we realize that MIN-LOSS DC has a notable advantage over MAX-GAIN, which is that it considers the *directionality* of conflicts. That is, while MAX-GAIN's binary search over α attempts to find the least risky way to approximate the probabilistic duration, it does so by trimming equal amounts of probability mass from each tail of the distribution. However, in many cases, the potential conflict may have only been due to just one side of the distribution, say, the upper bound, so also trimming off probability mass from the lower bound is a waste. MIN-LOSS, on the other hand, resolves each conflict independently, and thus does not need to sacrifice additional probability mass from both sides of the distribution.

SREA, which is based on a strong-controllability approximation, has the worst success rate in the DREAM benchmark, and is not significantly better than any of the other methods on CAR-SHARING. This demonstrates the value of using a dynamically controllable approach that can react to how uncertain events are resolved during execution. These results corroborate the original empirical evaluation of SREA, which showed that a simple early-first strategy, which is essentially an unsophisticated version of DC-DISPATCH that executes events as soon as all preceding events had been completed, can outperform SREA.

We note that both of our approaches took longer to compute than the DC-DISPATCH algorithm, which is to be expected given that it does not attempt to optimize for over-constrained PSTNs. MIN-LOSS took on average roughly 4 times longer to run than DC-DISPATCH. However, we note that this computation occurs offline before execution begins, and so is unlikely to preclude its use in any situation where DC-DISPATCH can be deployed.

Most of our empirical comparisons are against approaches that computed their dispatch strategy in an offline fashion prior to execution. However, we compare also against DREA, the version of DREAM that reruns SREA *every* time new information about the schedule is received. As shown in Table 1, both of our DC based approaches outperform DREA in terms of execution success rate for the DREAM benchmark. For CAR-SHARING, only MIN-LOSS has a higher average robustness than DREA, though the difference is not statistically significant.

The fact that MIN-LOSS DC achieves a higher average success rate than DREA is surprising and notable, because MIN-LOSS DC (like MAX-GAIN DC) computes a dispatchable network *offline*, prior to execution and so incurs no additional computational overhead during execution compared to other dynamic dispatch methods. This is in stark contrast to DREA, whose advantage is that it gets to recom-

pute a completely new, strongly controllable schedule that re-optimizes for new information every time new information arrives. As noted in Table 1, this leads to immense computational overhead, with DREA expending two orders of magnitude more computation than MIN-LOSS DC. These results provide many promising future avenues of exploration; for instance, we believe MIN-LOSS DC could replace SREA as the re-optimization technique in an approach such as DREAM, since it provides both higher success rates at lower computational costs.

Conclusions and Future Work

This paper presents two new approaches for attempting to schedule around the uncertainty present in PSTNs. MIN-LOSS DC works by identifying potential conflicts and minimally relaxing contingent edge bounds to resolve the conflicts. MAX-GAIN DC, on the other hand, works by performing iterative binary searches over the bounds on contingent edges until it can identify the bounds that capture the most probability. These are the first known approaches that attempt to find *dynamically*, rather than strongly, controllable STNU approximations of the PSTNs that attempt to maximize the probability of successful execution. Finally, our empirical evaluation confirms that both of our approaches improve upon or match the success rate of two existing state-of-the-art offline approaches, with MIN-LOSS DC consistently providing impressive improvements. In fact, MIN-LOSS DC outperforms dynamic approaches that allow for complete online re-computation of the dispatch strategy during execution.

In the future, we are interested in better understanding how both of our methods perform on PSTN benchmarks containing a wider diversity of distribution shapes. For instance, the MAX-GAIN DC algorithm currently *symmetrically* expands from the center of the distribution to capture as much probability mass as possible. Thus, we could potentially enhance the algorithm by considering the expansions of the lower bounds and the upper bounds separately. This could also make the algorithm more robust when encountering asymmetric probability distributions.

We are also interested in expanding our comparison against the DREAM algorithm. We believe there are natural ways to integrate our DC-based approaches into the DREAM framework to gracefully trade more online computation for more optimization of our dispatch. As previously mentioned, the DREAM algorithm works by invoking SREA after enough new information is received (Abrahams et al. 2019). Swapping SREA out for a successful DC-based approach like MINLOSS DC may further mitigate the need to recompute. We believe there might be principled, conflict-directed heuristics for determining when computing a new dynamically controllable schedule might be worth the computational overheard.

Acknowledgements

Funding for this work was graciously provided by the National Science Foundation under grant IIS-1651822. Thanks to the anonymous reviewers, HMC faculty, staff and HEAT-lab members for their support and constructive feedback.

References

Abrahams, J. R.; Chu, D. A.; Diehl, G.; Knittel, M.; Lin, J.; Lloyd, W.; Boerkoel, J. C.; and Frank, J. 2019. Dream: An algorithm for mitigating the overhead of robust rescheduling. In *Proc. of the 29th International Conference on Automated Planning and Scheduling (ICAPS-19)*, 3–12.

Akmal, S.; Ammons, S.; Li, H.; and Boerkoel, J. C. 2019. Quantifying degrees of controllability in temporal networks with uncertainty. In *Proc. of the 29th International Conference on Automated Planning and Scheduling (ICAPS-19)*, 22–30.

Deichter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. In *Artificial Intelligence 49*, 61–95.

Fang, C.; Yu, P.; and Williams, B. C. 2014. Chance-constrained probabilistic simple temporal problems. In *Proc. of the 28th National Conference on Artificial Intelligence (AAAI-16)*, 2264–2270.

Frank, J. 2019. On expected value strong controllability. In *Proc. of the ICAPS Workshop on Integrating Planning, Acting, and Execution (IntEx)*, 10–18.

Hunsberger, L. 2009. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *Proc. of the 16th International Symposium on Temporal Representation and Reasoning (TIME-09)*, 155—162.

Lund, K.; Dietrich, S.; Chow, S.; and Boerkoel, J. 2017. Robust execution of probabilistic temporal plans. In *Proc. of the 31st National Conference on Artificial Intelligence (AAAI-17)*, 3597–3604.

Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *Proc. of Principles and Practice of Constraint Programming (CP-06)*, 375–389.

Nilsson, M.; Kvarnström, J.; and Doherty, P. 2014. Classical dynamic controllability revisited - a tighter bound on the classical algorithm. In *Proc. of 6th International Conference on Agents and Artificial Intelligence (ICAART-14), 6-8 March 2014, Angers, France*, 130–141.

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 (ICAPS-16)*, 267–275.

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

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. In *Journal of Experimental and Theoretical Artificial Intelligence*, 0–32.