Simple Temporal Networks for Improvisational Teamwork

Malia Morgan, Julianna Schalkwyk, Huaxiaoyue Wang, Hannah Davalos, Ryan Martinez, Vibha Rohilla, James Boerkoel

Harvey Mudd College, Claremont, CA, USA
{mgmorgan, jschalkwyk, yukwang, hdavalos, rmmartinez, vrohilla, boerkoel}@g.hmc.edu

Abstract

When communication between teammates is limited to observations of each other's actions, agents may need to *improvise* to stay coordinated. Unfortunately, current methods inadequately capture the uncertainty introduced by a lack of direct communication. This paper augments existing frameworks to introduce Simple Temporal Networks for Improvisational Teamwork (STN-IT) — a formulation that captures both the temporal dependencies and uncertainties between agents who need to coordinate but lack reliable communication. We define the notion of *strong controllability* for STN-ITs, which establishes a static scheduling strategy for controllable agents that produces a consistent team schedule, as long as non-communicative teammates act within known problem constraints. We provide both an exact and approximate approach for finding strongly controllable schedules, empirically demonstrate the trade-offs between these approaches on benchmarks of STN-ITs, and show analytically that the exact method is correct.

Introduction

In a team where agents must work together, they would ideally be able to either pre-negotiate a coordination strategy or communicate one in real-time. However, there may be situations, such as ad-hoc teams, where agents must find a way to work together despite not being able to communicate directly. In such improvisational teams, agents must coordinate their tasks by only observing already-executed actions.

Existing work in multi-agent coordination either relies on solving the problem centrally or requiring communication before or during execution (Boerkoel and Durfee 2013; Boerkoel et al. 2013). Other work in human-robot teams allows the robots to dynamically recompute their plans in response to the humans' actions to deal with their uncertainty (Castro et al. 2017; Hoffman and Breazeal 2007). By contrast, we assume an agent must schedule its actions before execution and account for its teammates without negotiation. In addition, current work in temporal controllability places assumptions on the forms of uncertainty that a teammate could introduce (Vidal and Fargier 1999; Hunsberger 2009), which are restrictions we attempt to avoid.

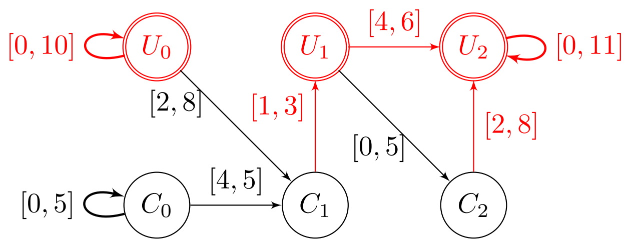


Figure 1: Distance graph of our STN-IT example.

This paper introduces the *Simple Temporal Network for Improvisational Teamwork (STN-IT)* to account for temporal planning situations where rational agents must coordinate their actions to complete a task but can only do so by observing each other's actions. We characterize strong controllability for STN-ITs and discuss the challenges for establishing strong controllability in STN-ITs. We present both an exact, Mixed Integer Linear Program for finding strongly controllable solutions to an STN-IT in a sound and complete manner and an efficient Linear-Programming-based approximate approach. Then, we empirically compare our approaches both across both a new and existing benchmarks of STN-ITs for efficiency and accuracy.

Background

As a motivating example used throughout this paper, consider a scenario where a robot and a novel human teammate are assigned to pack a box together (Figure 1). Both have packed similar boxes before and are aware of the steps and timing constraints. The robot, agent C, and human, agent U, move to the table, and each press a button to start the process. It takes the human between 0 and 10 seconds to get there (U_0), and robot C between 0 to 5 seconds (C_0). A box then enters on a conveyor belt, giving the robot between 4 and 5 seconds to remove it and place it on the table after they press the button and 2 to 8 seconds after the human presses their button (C_1). From there, the human will take between 1 and 3 seconds to place the packing material in it (U_1). The robot can then place the item 0 to 5 seconds later (C_2). Once the robot has placed the item, the human will take between 2 and 8 seconds to seal the box (U_2). However, to allow enough time to obtain supplies, the human has the additional constraint that their second action (U_2) must occur between 4 and 6 seconds after their first action (U_1). The box must be fully packed within 11 seconds.

In this section, we introduce how we can represent the structure of this problem using existing temporal network formulations and discuss how current representations are inadequate in capturing the robot's uncertainty introduced by the lack of direct communication with its human teammate.

Simple Temporal Networks

A Simple Temporal Network (STN) is a graph that consists of a set of timepoints T, constraints between those timepoints C, and a "zero" timepoint z that acts as a reference point and is assigned the time 0 (Deichter, Meiri, and Pearl 1991). A constraint in C is represented as $t_j - t_i \leq c_{ij}$ for timepoints $t_i, t_j \in T$. When $t_i - t_j \leq c_{ji}$ also exists, then $-c_{ji} \leq t_j - t_i \leq c_{ij}$, which we rewrite as $t_j - t_i \in [-c_{ji}, c_{ij}]$. An STN solution, or schedule, is an assignment of the timepoints in T such that all constraints in C are satisfied (Deichter, Meiri, and Pearl 1991). As shown in Figure 1, an STN can be represented graphically, where each timepoint appears as a node and each constraint $t_j - t_i \in [-c_{ji}, c_{ij}]$ appears as a directed edge from t_i to t_j with label $[-c_{ji}, c_{ij}]$. The direction of the edges is based on a partial ordering of the nodes with respect to time. Constraints involving the zero timepoint with the form $t_j - z \in [-c_{jz}, c_{zj}]$ are represented as self-loops with label $[-c_{jz}, c_{zj}]$. Distance graphs, shown in Figure 2, are helpful in determining a solution because they can be used to calculate implicit constraints between two timepoints using shortest path algorithms. Maintaining distance graphs can also be useful in guiding scheduling decisions during *dispatch*, the process of the agent deciding when to execute its events.

Multi-agent STN

A *Multi-agent Simple Temporal Network* (MaSTN), M, consists of multiple local STNs for each agent along with global constraints C^M that span between agents' local STNs. In a MaSTN involving a set of n agents, A, we can define each agent i's local STN as $S^i = \langle z, T^i, C^i \rangle$. Then, an MaSTN is defined $M = \langle A, C^M \rangle$, where $A = \langle S^1, \ldots, S^n \rangle$ and C^M is the set of global constraints. We set z as the shared zero timepoint across each agent's local STN to ensure a common reference point. Furthermore, agent i is responsible for assigning times to the timepoints T^i (Boerkoel and Durfee 2013). Figure 1 is an example of a MaSTN involving two agents U and C. A MaSTN is *decoupled* if every combination of solutions of the n local agents' STNs is a solution to the original MaSTN. Generally, decoupling a MaSTN places additional constraints onto each agent to embed the global constraints. For instance, the example STN could be decoupled if we assigned $C_0 = [1, 3]$, $U_0 = [0, 4]$, $C_1 = [6, 6]$, $U_1 = [7, 9]$, $C_2 = [7, 9]$, and $U_2 = [11, 11]$. However, decoupling assumes agents can negotiate a strategy before execution, which is not the case in improvisational teams.

STN with Uncertainty

A *Simple Temporal Network with Uncertainty* (STNU) is an STN with the set of events, T, partitioned into T^c, controllable events, and T^u, uncontrollable events (Vidal and Fargier 1999). While events in T^c are *decisions* made by an agent in terms of assigning the timepoints, those in T^u are decided by "Nature," an external force not controlled by the agent and *realized* during execution. Each uncontrollable event, $t_i \in T^u$, is associated with a unique *contingent* constraint of the form $t_i - t_j \in [lb, ub]$, where $0 < lb < ub$ and $t_j \in T^c$. Once t_j is executed, the value of t_i is *received* from Nature, which determines how long after t_j that t_i will occur by sampling the interval $[lb, ub]$. Thus, C can be partitioned into contingent constraints (C_C), described above, and *requirement* constraints (C_R), which are normal STN constraints (Hunsberger 2009).

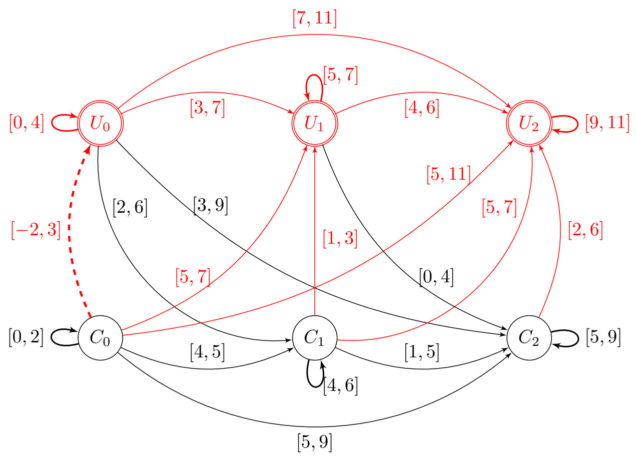


Figure 2: Distance graph of example problem after running Floyd-Warshall

An STNU is *controllable* when there exists a strategy to work around the uncertainty in the problem. Strong controllability occurs when an agent can schedule its controllable events pre-dispatch in a way that guarantees to be consistent with every realization of the uncontrollable timepoints. The STNU is dynamically controllable when there is a real-time strategy for scheduling controllable events that is guaranteed to work while only requiring information about past events (Vidal and Fargier 1999).

The robot in our running example is unsure of exactly when the human will execute their events, drawn in red in Figure 1. However, this problem cannot be captured as an STNU, since it violates multiple STNU assumptions. For instance, U_2 has three incoming contingent edges, which violates the assumption that uncontrollable timepoints can be the target of at most one contingent edge. Further, one of the incoming constraints has a lower bound of 0, which violates the constraint that contingent edges must strictly order events (i.e., $0 < lb < ub$).

Additionally, there is a difference in the nature of the uncertainty between STNUs and our running example. While STNUs characterize which events and constraints are uncontrollable/contingent (e.g., due to the inherent external sources of uncertainty), we need a *new* formulation that is capable of characterizing entire *agents* as uncontrollable (e.g., due to a lack of communication). Said another way, in STNUs, uncertainty is due to unknown, exogenous processes deciding parts of the schedule. In contrast, in our example, the robot knows that the human will determine their own schedule but has no way to coordinate and so is uncertain about *which* schedule the human will choose.

STN for Improvisational Teamwork

Like in past MaSTN work, we will assume each agent acting within an Improvisational Team is responsible for scheduling its local events in a way that is consistent with the global problem. However, unlike past MaSTN work, we do not assume that all agents can communicate to coordinate their local solutions. Instead, we designate a set of agents as *controllable* in the sense that a scheduling algorithm can communicate and coordinate solutions across these agents' subproblems. We designate the set of remaining agents, with which we can*not* coordinate as *uncontrollable*, since the scheduling algorithm is uncertain about which temporal plan they will choose.

We define an STN for Improvisational Teamwork (STN-IT) as a MaSTN where the set of agents, A, is partitioned between a set of controllable agents, A^C, and uncontrollable agents, A^U, $A = \langle A^C, A^U \rangle$. That is, an STN-IT is defined as $\mathcal{M} = \langle A^C, A^U, C^M \rangle$. Notably, our formulation does *not* designate specific edges or events as contingent or uncontrollable, just which type of agent is responsible for them.

While we assume no ability for a scheduling algorithm determining the controllable agents' schedule to control or communicate with agents in A^U directly, we do make a limited set of assumptions for how *all* agents will behave.

1. **Problem Observability:** Agents can observe and reason about the global STN-IT, $\mathcal{M}$.
2. **Event Observability:** Agents can observe when all events occur as they are executed.
3. **Execution Consistency:** Agents will choose an execution strategy that is consistent with all problem constraints and event observations once they occur.

These assumptions provide essential information about how the set of uncontrollable agents will behave in deciding their schedules. We acknowledge that these assumptions may limit the types of agents with which we can achieve improvisational teamwork. For instance, these assumptions may apply better to an ad-hoc robot teammate than to a human teammate (e.g., humans are generally known to be irrational). We elaborate on ideas for relaxing these assumptions in our discussion of future work.

In the remainder of this paper, we assume a single agent of each type for ease of composition. We believe most methods will extend to teams of agents where agents of the same type can coordinate with each other, but expanding this work to more general teams is left as future work.

Order of Events

Running the Floyd-Warshall algorithm on the STN-IT's distance graph reveals all implicit constraints (Figure 2). This naturally imposes an order between most, but not all, pairs of events. Consider two arbitrary events i and j, where we assume w.l.o.g. that $-c_{ji}$ and c_{ij} are the lower and upper bounds on the time that elapses between i and j: $t_j - t_i \in [-c_{ji}, c_{ij}]$. This pair will have one of two relationships, which we define next.

Case 1: ***precedes*** $(-c_{ji}, c_{ij} \geq 0)$ The time differences between i and j are non-negative, so event i must happen earlier than j. We define i *precedes* j ($i \rightarrow j$), which implies that j's agent has the responsibility to satisfy the constraints between them and account for how i is executed. For example, in Figure 2, the edge from C_0 to C_1 has the weights $[4, 5]$, so C_0 precedes C_1.

Notice that Case 1 also includes edges with $[0, 0]$ weights, which require i and j to be fully synchronous. In this case, if i and j belong to the same agent, we define an arbitrary ordering. If i and j belong to different agents, we assume that the controllable node precedes the uncontrollable node, which places the onus on the uncontrollable agent to observe and synchronize its event. In practice, exact synchrony may be impossible, so practitioners may choose to replace synchronous constraints with ones with more built-in tolerance.

Case 2: ***unordered*** $(-c_{ji} < 0, c_{ij} > 0)$ Case 2 occurs when there is no clear precedence order between nodes, which we define as *unordered* (i.e., $i \leftrightarrow j$). These two nodes *share* a conditional responsibility that is triggered by whichever one acts first. While both nodes remain unexecuted, neither agent needs to worry about satisfying the constraints between them. However, as soon as one agent executes, the unordered edge gets converted to an ordered one with one node preceding the other (Case 1). Specifically, if i executes first, j should take the responsibility and treat their edges as if $-c_{ji} = 0$ and vice versa. In Figure 2, the dotted edge from C_0 to U_0 has weights $[-2, 3]$, which is equivalent to a directed edge from U_0 to C_0 with weights $[-3, 2]$. So, if U_0 executes first, C_0 must happen within 3 units of time, else U_0 must happen within 2 units of time after C_0.

Strong Controllability of an STN-IT

We formally define an STN-IT to be *strongly controllable* if we can assign specific times to the controllable timepoints in a way that is guaranteed to work with any consistent, dynamically determined realization of uncontrollable timepoints. That is, we assume uncontrollable timepoints will be executed in a way that is consistent with observations about the events that precede them. More precisely, if we denote the space of solutions for controllable agents as $\mathcal{S}^C$ and uncontrollable agents as $\mathcal{S}^U$, then a strongly controllable STN-IT solution is a schedule to the controllable agent's problem, $s^C \in \mathcal{S}^C$ such that s^C is guaranteed to be consistent with *all* solutions $\mathcal{S}^U_{s^C} \subseteq \mathcal{S}^U$, where $\mathcal{S}^U_{s^C}$ represents the subset of $\mathcal{S}^U$ that is consistent with real-time observations of s^C.

Strong controllability takes on a different meaning within an STN-IT than in an STNU. In STNUs, 'nature' is agnostic to information about the temporal network when deciding contingent edges. Thus, contingent edges are independently sampled using some underlying, unknown distribution. This implies that agents must account for the entire cross-product of all possible realizations of contingent edges. As described in (Akmal et al. 2020), this forms a realization space that can be thought of as a hyperrectangle, with each contingent edge adding a new dimension. In STN-ITs, on the other hand, the uncertainty is over *which* temporal plan will be chosen by the other agent. Therefore, we need to be robust to just the uncontrollable agent's space of viable temporal plans. Further, our assumptions mean that controllable agents have a way to influence the uncontrollable agent's space of viable temporal plans through observations (e.g., the robot could opt to make the human wait before handing over a tool). This tightens the problem and allows for additional solutions that an STNU might not reveal.

Our assumptions mean that the uncontrollable agent can both reason about the problem constraints and dynamically respond to observations involving a constraint from the controllable agent to the uncontrollable agent that precedes one of its events. However, when one of the uncontrollable agent's events precedes one of the controllable agent's events, we do *not* assume that the uncontrollable agent can anticipate any effects other than those naturally implied by the original problem constraints. Once the distance graph has been computed, the uncontrollable agent only needs to know about its timepoints and any incoming or unordered edges involved in them, as highlighted in red in Figure 2. The only additional information we assume it has access to are real-time observations of the controllable agent during execution. Even with our assumptions, finding a strongly controllable STN-IT is non-trivial—there may be many local schedules that correspond to a global solution, but since agents cannot communicate, the controllable agent must reason over the uncontrollable agent's entire solution space.

Approaches for STN-IT Strong Controllability

This section explores two methods for finding a strongly controllable solution to an STN-IT. The first uses a Mixed Integer Linear Program (MILP) to find an exact solution. However, MILP's are generally NP-Hard to solve, so we provide a method for efficiently finding an approximate solution using a Linear Program (LP). Note that the STN-IT assumptions, particularly problem observability, mean that the controllable agent can access and reason about full problem using a centralized algorithm, but agents cannot access other agents' reasoning until the decision becomes apparent through observation.

An Exact Algorithm: STN-IT-SC-MILP

We introduce a method for determining a strongly controllable solution to an STN-IT using a Mixed Integer Linear Program (MILP)[1]. We also adopt notation from Wilson et al. (2014), which introduces decision variables t_i^- and t_i^+ that serve as shorthands for the lower and upper bounds on the constraint between t_i and the zero timepoint. Thus, if the STN-IT is strongly controllable, our MILP will return assignments to decision variables t_i^- and t_i^+ that specify the range of times each event can occur for that particular controllable solution.

[1]We express constraints using min/max, which Gurobi converts into a linearized MILP (Gurobi 2021; pp. 623-4)

STN-IT-SC-MILP:

$$t_j^+ = t_j^- \qquad \forall t_j \in T^c \qquad (1)$$

$$t_i^+ - t_j^- \leq c_{ji} \qquad \forall i,j \mid t_j \in T^c, i \rightarrow j \qquad (2)$$

$$t_j^+ - t_i^- \leq c_{ij} \qquad \forall i,j \mid t_j \in T^c, j \nrightarrow i \qquad (3)$$

$$t_j^- \leq t_j^+ \qquad \forall t_j \in T^u \qquad (4)$$

$$t_j^- = \max_{i \mid i \rightarrow j}\{t_i^- - c_{ji}\} \qquad \forall t_j \in T^u \qquad (5)$$

$$t_j^+ = \min_{i \mid j \nrightarrow i}\{t_i^+ + c_{ij}\} \qquad \forall t_j \in T^u \qquad (6)$$

STN-IT-SC-MILP fully assigns all *controllable* timepoints by enforcing $t_j^+ = t_j^-$ (Eq. 1) for all controllable timepoints. Controllable timepoints are assigned to be consistent with the extreme values of any preceding uncontrollable timepoints in Eqs. 2-3. Eq. 2 ensures that t_j's lower bound is consistent with full range of times for all timepoints t_i that precede it ($i \rightarrow j$), while Eq. 3 does the same for t_j's upper bound. Note that the notation $j \nrightarrow i$ in Eq. 3 is short for the cases when i precedes *or* is unordered with j, and handles the case when uncontrollable agent acts first in an unordered edge. Because we assign specific times to controllable timepoints, Eqs. 2-3 also naturally enforce all of the controllable agent's internal constraints.

At the same time, the MILP enforces that each uncontrollable timepoint t_j maintains the full range of possible times that ensures strong controllability, with Eq. 4 ensuring the intervals are well-formed. Because strong controllability assumes uncontrollable agents can only adjust to events that have already occurred, our MILP adjusts the ranges of uncontrollable timepoints only in response to the timepoints t_i that precede t_j. Note that the lower bound of the constraint from t_i to t_j is $t_i - t_j \leq c_{ji}$, which can be rewritten as $t_j \geq t_i - c_{ji}$. Then, the smallest adjustment we can make to t_j's lower bound (t_j^-) while ensuring $t_j \geq t_i - c_{ji}$ holds for all t_i that precede t_j is exactly $\max_{i|i\rightarrow j}\{t_i^- - c_{ji}\}$, which is achieved by Eq. 5. Similarly, Eq. 6 achieves the necessary updates to t_j's upper bound. Finally, unordered constraints only need to be used to update the upper bound of t_j, so they are included in Eq. 6 but not 5. Directly assigning each controllable timepoint randomly within its allowed bounds could remove solutions that may optimize secondary goals, such as flexibility or makespan. Finally, while MILPs traditionally have an objective function, none is needed in this case. An objective function could optionally be added to prioritize among solutions when more than one exists.

Example Execution We now step through how our MILP would apply to the distance graph of our running example displayed in Figure 2 with the output shown as Figure 3. While for ease of explanation, we discuss the MILP as operating sequentially on each timepoint, in reality, the fact that the MILP considers all constraints simultaneously is essential in its ability to find strongly controllable solutions. First, since C_0 has no timepoints that precede it, the MILP only has to assure that C_0's assignment is consistent with both its original range of times and the unordered constraint shared with U_0 (Eq. 3). These two constraints enforce C_0 to execute

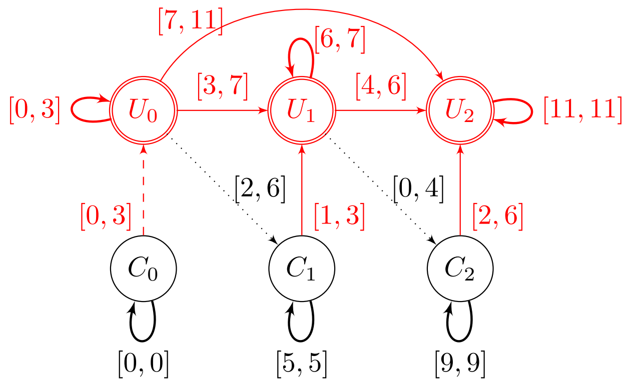


Figure 3: Strongly controllable solution to our example

within the range of $[0, 2]$, so the MILP happens to assign C_0 to 0. With $C_0 = 0$, the MILP can now effectively treat the originally unordered edge as one with C_0 preceding U_0 and label $[0, 3]$, which it does naturally by only concerning itself with the upper bound on the constraint from C_0 to U_0 (Eq. 6). Thus, our MILP updates the upper bound of U_0 to 3. Note that during actual execution, agent U will start off believing it has until time 4 to complete U_0 as shown in Figure 2. However, as soon as it observes that C_0 happens at time 0, it will update its executable range to be [0,3], as captured in Figure 3.

Next, the MILP considers that C_1 (and similarly C_2) must be able to account for the full range values that agent U might consider for completing U_0 without knowing its value in advance. Since agent U_0 could choose to execute as U_0 as early as 0, we know the latest C_1 can occur is $0+6 = 6$ (Eq. 3), and similarly, since the latest U_0 could occur is 3, the earliest C_1 can take place is $3 + 2 = 5$ (Eq. 2). Ultimately, the MILP decides to assign $C_1 = 5$. Finally, when adjusting U_1's range of values (and similarly U_2), the MILP considers U_1's current range $[5, 7]$, along with the ranges implied by both incoming edges (e.g., the edge from U_0, which implies a range for U_1 of $[0, 3] + [3, 7] = [3, 10]$, and the edge from C_1, which implies a range of $[5, 5] + [1, 3] = [6, 8]$.) Eqs. 5-6 then take the intersection of these three to obtain U_1's new range $U_1 : [5, 7] \cup [3, 10] \cup [6, 8] = [6, 7]$. Again, before actual execution, agent U may not realize it cannot start U_1 before time 6, but it will realize it by time 5 when it observes C_1 has occurred. As illustrated in Figure 3, the MILP ends up using the same logic to constrain $C_2 = [9, 9]$ and $U_2 = [11, 11]$ to give us our final, strongly controllable solution.

Correctness of STN-IT-SC-MILP

After defining our MILP, we argue that our proposed MILP is complete and sound by presenting two theorems and the corresponding proof sketches[2].

Theorem 1. *STN-IT-SC-MILP will return a strongly controllable schedule to an STN-IT any time one exists.*

[2]Full proofs available upon request.

Proof (Sketch). Eq. 4 ensures a valid interval of times for each timepoint. By using proof by contradiction, we must show that Eqs. 5-6 set each uncontrollable timepoint's t_j^- and t_j^+ while assuring that the corresponding interval retains the full range of time that is consistent with all events that precede it, as required by the definition of strong controllability.

We assume that there exists at least one uncontrollable timepoint t_k, where its earliest possible time t_k^- does not satisfy Eq. 5 (the proof for the upper bound t_k^+ follows symmetrically). However, t_k^- cannot be less than $\max_{i|i\to k}\{t_i^- - c_{ki}\}$, since t_k^- would then be inconsistent with respect to its constraints with at least one timepoint t_i that precedes it, which violates our assumption. Similarly, t_k^- cannot be greater than $\max_{i|i\to k}\{t_i^- - c_{ki}\}$, since that would contradict the assumption that we are maintaining the largest possible interval for t_k^-. Thus, Eq. 5 (and symmetrically Eq. 6) must hold for all uncontrollable timepoints.

Next, we consider the controllable timepoints. If an STN-IT is strongly controllable, there must exist a fully assigned schedule for the controllable timepoints that will work regardless of how the uncontrollable timepoints are chosen and how the controllable timepoints can be assigned (Eq. 1). Similar to Eqs. 5-6, we can argue by contradiction that any strongly controllable solution to an STN-IT must satisfy Eqs. 2-3. Assume that there is a strongly controllable solution that violates either Eq. 2 or Eq. 3. However, if the two timepoints involved in the violation are controllable, not satisfying either Eq. 2 or 3 would imply violating the original problem's constraint between controllable variables (since $t_j^- = t_j^+ \ \forall t_j \in T^c$), which contradicts our assumption of a strongly controllable solution. Similarly, if the other timepoint involved is uncontrollable, we can show that a violation of Eq. 2 or 3 would imply conflicts with either the lower or upper bound on the full range of times for that uncontrollable timepoint, thereby contradicting our assumption.

Therefore, any strongly controllable STN-IT will yield an assignment of controllable timepoints consistent with the constraints of our STN-IT-SC-MILP. □

Theorem 2. *Any assignment of t_i^- and t_i^+ that satisfies the constraints of the STN-IT-SC-MILP results in a strongly controllable solution to the original STN-IT.*

Proof (Sketch). First, consider any assignment of t_j^- and t_j^+ to the controllable timepoints that satisfies the MILP. From Eqs. 2-3, we see that for any timepoint t_i that precedes a controllable timepoint t_j, t_j will happen no earlier than t_i's latest time plus the lower bound between them ($-c_{ji}$) and no later than t_i's earliest time plus the upper bound between them (c_{ij}). Thus, t_j is guaranteed to be consistent with t_i.

Now consider any assignment of t_j^- and t_j^+ to the uncontrollable timepoints that satisfies the MILP. To start, we consider the relationship between an uncontrollable timepoint t_j and an arbitrary timepoint t_i that precedes or is unordered with t_j ($j \nrightarrow i$). When t_i happens maximally early, t_j can happen no later than t_i^- plus the upper bound between them, which gives us (a): $t_j^- \leq t_i^- + c_{ij}$. When t_i happens maximally late, t_j can happen no earlier than t_i^+ plus the lower bound between them, which gives us (b): $t_j^+ \geq t_i^+ - c_{ji}$. If t_i is controllable, we can derive (a) and (b) directly from the MILP.

When t_i is uncontrollable, we cannot do so. Instead, we use proof by strong induction on n, the number of uncontrollable timepoints in the STN-IT, to prove that (a) and (b) hold between any pair of uncontrollable timepoints. When there is only one uncontrollable timepoint, its local problem must be consistent and we have already shown that (a) and (b) hold between controllable and uncontrollable timepoints. Hence, the base case holds. For our inductive hypothesis, we assume that for every STN-IT that has n or fewer uncontrollable timepoints, equations (a) and (b) hold for all uncontrollable timepoints t_i that precedes or is unordered with other uncontrollable nodes t_j $(j \nrightarrow i)$. Then we use proof by contradiction to prove the inductive hypothesis holds for $n+1$ uncontrollable timepoints.

Consider when (a) does not hold in an STN-IT with $n+1$ uncontrollable timepoints. There must be at least one uncontrollable timepoint that does not enable any other uncontrollable timepoint. Let's call it the maximal timepoint t_m. In the STN-IT without t_m, (a) does not fail between any of the remaining uncontrollable timepoints by the inductive hypothesis so it must fail between t_m and another uncontrollable timepoint t_i. Because (a) has failed between t_m and t_i, timepoint t_i cannot dominate the maximum of MILP constraint (1), so another timepoint t_k must do so. When investigating t_k's relationship to t_i, we get $c_{ik} > c_{im} + c_{mk}$, which implies that there is a shorter path from t_i to t_k through t_m. This result contradicts the assumption that we ran Floyd-Warshall. Hence, (a) must hold between all uncontrollable timepoints in the STN-IT with $n+1$ uncontrollable timepoints. We can use the same reasoning to prove that (b) must also hold.

Overall, we have shown that every solution to the MILP results in a strongly controllable solution to the STN-IT. Thus, our MILP is sound. □

An Approximate Algorithm—STN-IT-SC-LP

Although our MILP can correctly determine whether an STN-IT is strongly controllable, because MILP formulations are generally NP-Hard, this approach may be intractable for some problems. Thus, we also developed a linear program (LP) version of the algorithm that approximates the MILP result. Our basic approach is to replace equations that contained the non-linear max/min functions that requires formulation as a MILP with linear constraints by swapping Eqs. 5-6 for 7-8:

$$t_j^- \geq t_i^- - c_{ji} \qquad \forall i,j \mid t_j \in T^u, i \rightarrow j \qquad (7)$$

$$t_j^+ \leq t_i^+ + c_{ij} \qquad \forall i,j \mid t_j \in T^u, j \nrightarrow i \qquad (8)$$

Next, in order to encourage t_j^- and t_j^+ to approximate their respective maximum and minimum value, we add an objective that maximizes the sum of uncontrollable timepoint's time interval $(t_j^+ - t_j^-)$ relative to its time initial interval, $c_{zj} + c_{jz}$, computed by Floyd-Warshall:

$$\textit{maximize} \sum_{t_j \in T^u} \omega_j \cdot \frac{t_j^+ - t_j^-}{c_{zj} + c_{jz}}$$

In addition, the weight ω_j allows us to explore different ways of relatively weighting the uncontrollable timepoints. We have determined that giving uncontrollable timepoints that are earlier or have shorter durations higher weights has a small, but positive impact. Specifically, if a timepoint t_j is the x^{th} earliest one among y uncontrollable timepoints, it will receive a weight of $\omega_j = \frac{y-x}{\frac{y(1+y)}{2}}$. In summary, the approximate LP weights early timepoints and timepoints with short durations higher than late, long ones.

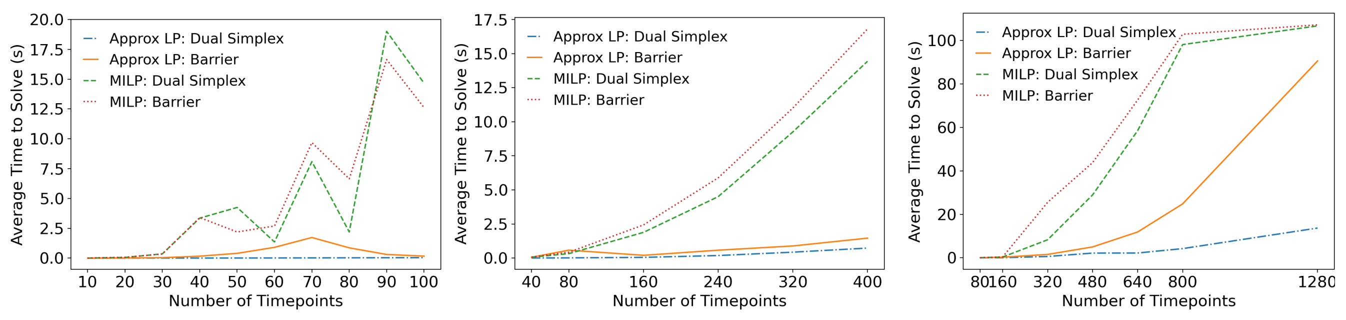
Empirical Evaluation

We evaluate our algorithms against three benchmarks. The first two are the original MaSTN benchmarks published by Boerkoel et al. (2013). The *WS benchmark* was generated to mimic a realistic a multiagent factory scheduling domain. *BDH benchmark* was generated using Boerkoel and Durfee (2013) multiagent adaptation of Hunsberger (2002) random STN generator. Though these benchmarks had up to 32 agents, we converted them to two agent problems by combining all even numbered agents into the controllable agent and the rest into the uncontrollable. However, while these benchmarks were useful in evaluating how our algorithms scaled, our initial investigations suggested they were relatively loosely coupled between agents and often proved solvable by the naïve algorithms that we explain later.

Thus, we decided to generate additional, more interesting STN-IT instances that require greater coordination and are more difficult to resolve by further adapting Boerkoel and Durfee (2013)'s MaSTN benchmark generator. Like the original generator, ours first randomly assigns some tasks to the controllable and uncontrollable agents, each with start and end timepoints and bounds on its duration. We modified how we define constraints between tasks to create more interesting STN-IT examples requiring greater coordination. Specifically, each constraint has a 25% probability of being an unordered constraint rather than an ordered one. After randomly deciding the type, our generator determines bounds by uniformly sampling [-60,60] for local constraints and [-120, 120] for global constraints. To ensure a consistent problem, the generator checks each constraint, skips adding it if it results in a conflict, and then randomly generates a replacement. Finally, the generator stops once the temporal network is connected. Using this generator, we created a set of 1,000 example problems in total, consisting of 100 problems in each size from 10 to 100 in steps of 10.

Efficiency

To test the efficiency of our MILP and Approximate LP, we used the Gurobi Optimizer's provided implementation of the Dual Simplex and Barrier methods (Gurobi 2021). While Simplex methods have a worst-case exponential runtime, they are often efficient in practice (Spielman and Teng 2004). Meanwhile, the Barrier method can find the solution



(a) New STN-IT benchmark (b) BDH benchmark (c) WS benchmark

Figure 4: Solve time (seconds) vs. problem size (number of timepoints) across three benchmarks.

in a worst-case polynomial runtime by traversing inside or outside of the feasible region, but each step is relatively expensive (Nocedal and Wright 2006).

Figure 4 reports the average runtime of each approach as the problem size scales. Note that the scale of the x axis and y axis of each benchmark is different. The 1,000 newly generated STN-ITs include a few problems that the MILP could not solve in under a 5 minute time limit (Dual Simplex timed out on 1 problem with 40 timepoints, 2 with 70, 5 with 90, and 3 with 100, whereas Barrier timed out on 1 problem with 40 timepoints, 2 with 70, 4 with 90, and 2 with 100). Similarly, there are 11 timed-out problems in the WS benchmark (most involving 1280 timepoints) but *none* in the BDH benchmark. Overall, the Approximate LP scales significantly better overall than the MILP. The Dual Simplex method generally performed best, though the Barrier method has a theoretically better worst-case runtime. We believe the Barrier method's vulnerability to numerical issues causes the peak in the LP runtime at 70 timepoints (Gurobi 2021; pp. 941-2). While our newly generated benchmark proved to quickly overtax the MILP algorithms as seen in 4a, we did generate problems with up to 512 timepoints and verified that the LP's runtime scaled efficiently. In addition, Figures 4b and 4c show how performance scales to larger problems where coordination was easier.

Factors impacting MILP efficiency Next, we explored which features of the new STN-IT benchmark impact the MILP performance using Dual Simplex. While we found no strongly correlated features, we did find that STN-ITs that timed out or took a significant time to solve tended to be ones with a smaller portion of unordered edges, as shown in Figure 5. We also found that among strongly controllable STN-ITs, those with longer solve times tended to have smaller average executable ranges (i.e., measured as the size of the interval at the time of the uncontrollable agent's execution) as shown in Figure 6. We tested a variety of other features (e.g. the ratio between uncontrollable to controllable timepoints, the ordering between uncontrollable and controllable timepoints, etc.) that ended up not being all that predictive of MILP runtime. In summary, MILP runtime is most affected by the problem size, the tightness of timepoints' executable ranges, and the strictness of the timepoint ordering.

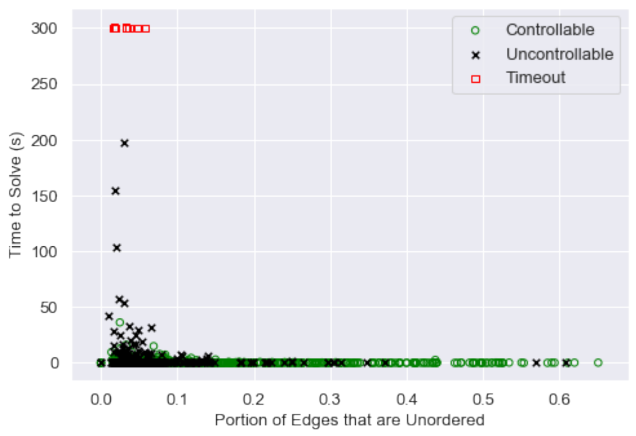


Figure 5: Solve time (s) *vs.* the portion of edges in the problem that are unordered.

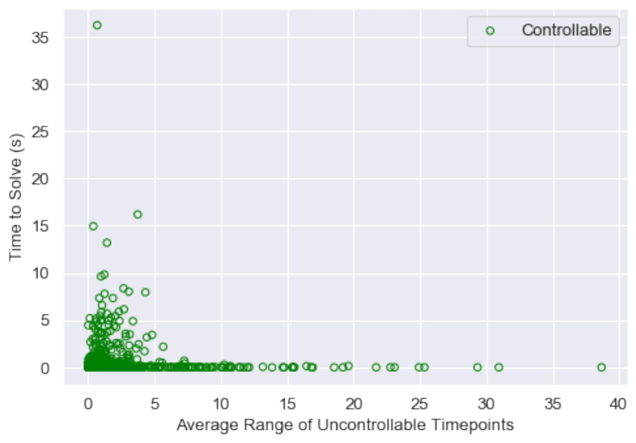


Figure 6: Solve time (s) *vs.* average range of uncontrollable timepoints for strongly controllable problems.

Ctrl Method	# of solns	# of failures	# of timeouts	Emp. Verif.
MILP	634	355	11	634
LP	1000	0	0	286
Early	1000	0	0	333
Rand.	1000	0	0	[85 - 101]

Table 1: Number of solutions, failures, timeouts, and empirically verified solutions returned by each method.

Ctrl Method	Rand. Unctrl.		Early Unctrl.	
	All	SC	All	SC
MILP	63.4%	100%	63.4%	100%
LP	56.52%	69.6%	52.8%	62.0%
Early	47.32%	63.6%	100%	100%
Rand.	[23.0 - 24.6%]	[31.6%, - 34.0%]	[22.0% - 25.3%]	[30.4% - 35.0%]

Table 2: Empirical performance against two simulated uncontrollable agents reported across *all* problems and just the strongly controllable (SC) instances.

Accuracy

Next, we examined the trade-offs in terms of accuracy between the Approximate LP and exact MILP approaches using the more efficient Dual Simplex Method on the same, newly generated benchmark. We also compared against two other naïve algorithms: an *early* dispatch strategy where each controllable timepoint executes at its earliest possible time; and a *random* dispatch strategy where each controllable timepoint executes at a random time in the interval defined by its bound with the zero timepoint z. We repeated the random strategy 50 times and report the 95% confidence interval across all instances.

Strong Controllability Validation First, we developed a program to verify the correctness of an STN-IT's solution. It checked that assignments to controllable timepoints were a valid solution to the controllable agent's subproblem. It also ensured that the solution was consistent with all possible assignments to uncontrollable intervals. To mimic the dynamic solving process of the uncontrollable agent, we updated the time interval for each uncontrollable timepoint any time a timepoint preceding it executed. During this dynamic process, we verified that the full range of the time for each uncontrollable timepoint satisfied its constraints, which guarantees the solution is strongly controllable.

Table 1 shows the results for each method. Note the first column reports the number of (approximate) solutions returned, while the last reports the number empirically validated as strongly controllable. The MILP outperforms the other strategies, with the highest accuracy, as all of its solutions are empirically validated as correct. However, it does time out on 11 of the problems. The random strategy finds only 13.4-15.9% of the correct solutions that the MILP does, while the Approximate LP and early strategies find 45.1% and 52.5%, respectively. It is surprising that the naïve early first approach led to strongly controllable solutions more often than our LP approximation. This points to the existence of structural features that may make the *early* strategy a reasonable choice in some cases and validates work that shows *early* is surprisingly effective in realistic, probabalistic settings (Saint-Guillain et al. 2021).

Empirical Performance One advantage of the Approximate LP is that they always return an approximate solution, giving the team a chance of success even if it is not guaranteed. We tested each method against two simulated models of uncontrollable agents, one that uniformly randomly selects times from its interval and the other that selects the earliest time. We verified how often the solution returned by each method resulted in a consistent simulated execution.

There are several conclusions that we can draw from Table 2. First, the approximate methods worked substantially better on the set of strongly controllable problems, including the random method, which had a statistically significant increase in solutions. Second, the Approximate LP performed especially well against the random uncontrollable agent in strongly controllable problems, succeeding nearly 70% of time. The LP, which gives the controllable agent a chance to succeed, even if it is not guaranteed, closes the relative gap across all problems, succeeding 7% less often than MILP, though MILP was still most likely to succeed in expectation.

The one exception is that the early strategy had 100% accuracy for situations where the uncontrollable agent also used the early strategy. This is a tautological result since if a solution exists, a dynamic early strategy paired with a strong early strategy is guaranteed to find it. We hypothesize that there exist problem structures where the LP is a better choice than the exact MILP solution given that the LP scales significantly better than the MILP and still performs well.

Discussion

Our new framework of an STN for Improvisational Teamwork allows us to model impromptu teamwork performed without reliable communication. We determined a set of assumptions that enable the first definition of strong controllability for improvisational teams. We translated this definition into a Mixed Integer Linear Program that characterizes strongly controllable solutions to STN-ITs when they exist and argued analytically that the MILP was both sound and complete. We showed empirically that the MILP correctly identified which STN-IT's were strongly controllable across a new benchmark of 1000 randomly generated STN-ITs, though MILP is generally NP-hard. We also provided an efficient, LP-based approach that approximates the MILP result. While the MILP scales reasonably well on problems with fewer than 100 timepoints, we showed that the Approximate LP scales much better but sacrifices overall accuracy, leading to a lower likelihood of successful execution.

Future work includes analyzing the characteristics of real-world human-robot teamwork to create a more accurate, interesting set of benchmarks. We are also interested in extending to teams with more than two agents, or relaxing the assumptions that the uncontrollable agent is fully rational and has full observability. Finally, we hope to validate our approaches in an actual human-robot deployment.

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

Akmal, S.; Ammons, S.; Li, H.; Gao, M.; Popowski, L.; and Boerkoel, J. C. 2020. Quantifying controllability in temporal networks with uncertainty. *Artificial Intelligence*, 289: 103384.

Boerkoel, J.; and Durfee, E. 2013. Distributed Reasoning for Multiagent Simple Temporal Problems. *Journal of Artificial Intelligence Research*, 47: 95–156.

Boerkoel, J.; Planken, L.; Wilcox, R.; and Shah, J. 2013. Distributed Algorithms for Incrementally Maintaining Multiagent Simple Temporal Networks. In *Proc. of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-13)*, 11–19.

Castro, B.; Roberts, M.; Mena, K.; and Boerkoel, J. 2017. Who Takes the Lead? Automated Scheduling for Human-Robot Teams. In *Proc. of AAAI Fall Symposium on Artificial Intelligence for Human-Robot Interaction; AAAI Technical Report FS-17-01*, 85–89.

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

Hoffman, G.; and Breazeal, C. 2007. Effects of anticipatory action on human-robot teamwork efficiency, fluency, and perception of team. In *Proc. of The 2nd Annual Conference on Human-Robot Interaction (HRI-07)*, 1–8.

Hunsberger, L. 2002. Algorithms for a Temporal Decoupling Problem in Multiagent Planning. In *Proc. of The 18th National Conference on Artificial Intelligence (AAAI-02)*, 468–475.

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.

Nocedal, J.; and Wright, S. J. 2006. *Numerical optimization*, 563–597. New York: Springer, 2nd ed. edition.

Saint-Guillain, M.; Vaquero, T.; Chien, S.; Agrawal, J.; and Abrahams, J. 2021. Probabilistic Temporal Networks with Ordinary Distributions: Theory, Robustness and Expected Utility. *Journal of Artificial Intelligence Research*, 71: 1091–1136.

Spielman, D. A.; and Teng, S.-H. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3): 385–463.

Vidal, T.; and Fargier, H. 1999. Handling Contingency in Temporal Constraint Networks: from Consistency to Controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1): 23–45.

Wilson, M.; Klos, T.; Witteveen, C.; and Huisman, B. 2014. Flexibility and decoupling in Simple Temporal Networks. In *Artificial Intelligence 214*, 26–44.