Multiagent Disjunctive Temporal Networks

Nikhil Bhargava
Massachusetts Institute of Technology
Cambridge, MA
nkb@mit.edu

Brian Williams
Massachusetts Institute of Technology
Cambridge, MA
williams@mit.edu

ABSTRACT

Temporal network formalisms allow us to encode a set of constraints relating distinct events in time, and by deploying algorithms over these networks, we can determine whether schedules for these networks exist that satisfy all constraints. By augmenting simple temporal networks, we can consider the effects that disjunctive constraints, temporal uncertainty, and coordinating agents have on modeling fidelity and the algorithmic efficiency of schedule construction. In this paper, we introduce Partially Observable Disjunctive Temporal Networks with Uncertainty (PODTNUs) and Multiagent Disjunctive Temporal Networks with Uncertainty (MaDTNUs), generalizing previously studied multi-agent variants of temporal networks. We provide the first theoretical completeness results for the controllability of multiagent temporal network structures and discuss the importance of these results for modelers.

KEYWORDS

Single and multiagent planning and scheduling; Coordination and control models for multiagent systems

ACM Reference Format:
Nikhil Bhargava and Brian Williams. 2019. Multiagent Disjunctive Temporal Networks. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

In temporal planning problems, agents are tasked with finding schedules for a series of events that are jointly constrained. The problem of constructing a valid schedule becomes more difficult as temporal planning models admit more features. In this paper, we examine the difficulty of constructing schedules in a multiagent context with disjunctive constraints and temporal uncertainty. These features are important for faithfully capturing the semantics of real-world situations and for planning over them.

Disjunctive constraints are important for modeling common phenomena like resource constraints and mutual exclusion (i.e. I can eat 30 minutes before swimming or after, but cannot eat while in the pool). These types of networks have been studied extensively in the forms of Temporal Constraint Satisfaction Problems (TCSPs) [6] and Disjunctive Temporal Networks (DTNs) [11]. Another important feature that is needed to faithfully model non-determinism in temporal events, such as the effect of traffic on a drive across town, is temporal uncertainty. Temporal uncertainty and disjunction have been studied together in Temporal Constraint Satisfaction Problems with Uncertainty (TCSPUs) [13] and Disjunctive Temporal Networks with Uncertainty (DTNUs) [14].

These networks, however, focus exclusively on single-agent planning and scheduling without capturing the richness of multiagent scenarios. It is of course possible to model multi-agent problems using these formalisms, but such approaches have their shortcomings. Such models either assume that all agents can be jointly controlled, which is too strong of an assumption in practice, or more conservatively, that the actions of other agents are themselves modeled as temporally uncertain processes and cannot be strategically reasoned about. While such approaches provide some correctness guarantees, they require agents to act in a way that is robust to all possible actions that other agents might take rather than reasoning over joint strategies that permit coordination without absolute control of all other agents.

The main contribution of this paper is the introduction of multiagent temporal networks with disjunctions and uncertainty and providing completeness results for their complexity. We demonstrate that the addition of temporal uncertainty to multi-agent disjunctive temporal network problems raises the complexity to PSPACE-completeness in the case of PODTNUs and to NEXP-completeness in the case of MaDTNUs. These results represent the first completeness results for multiagent temporal networks. We conclude with a discussion of these results in the context of the complexity of evaluating other temporal networks.

2 MOTIVATION

The example we use to motivate our investigation of these new temporal networks involves a series of humans and robots in a warehouse working collaboratively to fill orders. Two dexterous (potentially heterogeneous) picker robots are tasked with retrieving individual items from bins across the warehouse and a third delivery robot is tasked with taking bins of objects that compose orders to a group of human packers who will inspect the items for defects before placing them in a box with appropriate packing materials.

We have two new orders come in each with two different objects, and the two picker robots are tasked with retrieving one item from each order as matches their skill sets. Retrieving a single item takes 20-30 minutes, and each picker locally has the flexibility to choose the order in which it undertakes tasks. Once a bin of objects is assembled, it takes 15 minutes for the delivery robot to bring it to the human packer. It takes a human packer 15 minutes to inspect and assemble a package once the items are delivered.

In this problem, warehouses must maintain a high level of throughput, imposing temporal deadlines, and traffic within the warehouse and the variable difficulties of grasping tasks implies that there is temporal variability in the execution of actions. It's clear that there needs to exist some capability of representing choices (i.e. choose

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

which object to grab first) and disjunctive constraints give us a powerful way to do that (i.e. we require that either the pickup of object 1 ends before pickup of object 2 starts or vice versa). The remaining question is whether it suffices to use a series of single-agent simplifications, such as DTNs or DTNUs, to model this problem.

Unfortunately, the answer is no. A single-agent projection of our problem allows a robot to freely choose the ordering of their actions but the actions of others are then abstracted to entirely stochastic, uncorrelated and uncontrollable actions. If the two packages must be fully prepared within 90 minutes, we know the task is impossible. The first picker robot cannot know a priori which object the second one will grab first and so the two may choose objects from different orders. If it took both robots the full 30 minutes to grab the orders, then the delivery robot could only deliver the packages after an hour; the packer would get the deliveries after 75 minutes and would need at least 30 to finish with both.

In contrast, modeling the system using multi-agent temporal networks, as is done with POSTNUs and MaSTNUs, would allow us to correctly determine that all constraints can be satisfied. Though the pickers may not know what the other is doing in real-time, they can adopt contingency and coordination strategies that eliminate much of the network's cross-agent uncertainty. PODTNUs and MaDTNUs allow for the encoding of multi-agent coordination strategies that is are common across many other game-playing formalisms.

3 BACKGROUND

The Simple Temporal Network (STN) is an effective formalism for modeling temporal constraints and the construction of schedules around those constraints [6], but as modelers, we often find that they lack the expressivity commonly found in the types of scenarios we encounter in our daily lives. STNs allow us to schedule a series of *timepoints* and enforce simple *temporal constraints* between them (i.e. event A must happen at least 20 minutes after event B), and while these networks are able to richly model temporal flexibility, they fail to capture the semantics of more expressive constraints.

Adding disjunctive constraints can significantly expand the types of problems we can model. TCSPs add *simple disjunctions*, which admit a series of allowed temporal distances between events (i.e. I either want a quick workout that takes at most 25 minutes, or I want an hour-long workout, with nothing in between) [6]. DTNs allow *full disjunctions*, meaning disjunctions across sets of timepoints are allowed [11]. With DTNs, we can express novel behaviors, like resource constraints, by imposing ordering (i.e. only one truck can use the loading dock at a time, so either truck A finishes unloading before truck B starts or vice versa).

What these models lack is the ability to construct a schedule when information is unavailable during both planning and execution phases, or, in other words, to construct schedules when the values of certain timepoints cannot be fully controlled and are subject to temporal uncertainty.

Simple Temporal Networks with Uncertainty (STNUs) [15] and DTNUs provide a way to model events that are outside of the control of any individual agent (i.e. a person can choose when to start their morning commute but when it ends is dependent on factors outside of their control, like traffic and the weather); the constraints that model this uncertainty are called *contingent constraints* in contrast to constraints like the ones in STNs, which we call *requirement constraints*. With the distinction between contingent and requirement constraints, we also split our timepoints into *contingent* ones and *executable* ones. Contingent timepoints are one whose values are determined by nature in accordance with the contingent constraints. Executable timepoints are timepoints that are scheduled directly by the agent. It is worth noting that requirement constraints are the responsibility of the executing agent and are free to constraint any timepoints.

STNUs can be used to model multiagent problems from the perspective of a single agent by interpreting the actions of all other agents as highly uncertain. While modeling these behaviors with an STNU provides us with efficient, polynomial time algorithms [9], they preclude us from reasoning about joint strategies that may guarantee valid schedules in a world of temporal uncertainty.

Taking this approach oversimplifies and is overly conservative because it ignores shared sources of uncertainty [14]. For example, an office colleague might send over a copy of their report between 9am and 10am and separately send a budget between 9:30am and 10:30am, but the uncertainty across both events might be entirely determined by whether their commute that day took 30 or 90 minutes. To account for the correlation in these uncertainties, we can extend DTNUs to consider Partial Observability just like we do with STNUs and Partially Observable Simple Temporal Networks with Uncertainty (POSTNUs) [7]

4 MULTIAGENT DISJUNCTIVE DEFINITIONS

Extending DTNUs to include partial observability yields Partially Observable Disjunctive Temporal Networks with Uncertainty (PODTNUs). PODTNUs make a distinction between contingent timepoints that are observable and unobservable, allowing the modeler to chain together contingent constraints to model shared causes of stochasticity.

Definition 1. PODTNU
A PODTNU is a 5-tuple $\langle X_e, X_c, X_u, R_r, R_c \rangle$ where:

- X_e is the set of executable timepoint variables
- X_c is the set of observable contingent timepoint variables
- X_u is the set of unobservable contingent timepoint variables
- R_r is the set of full disjunctive temporal constraints indexed by k, called requirement constraints, of the form $\bigvee_k \left(l_{r,k} \leq x_{r,k} - y_{r,k} \leq u_{r,k}\right)$, where $x_{r,k}, y_{r,k} \in X_e \cup X_c \cup X_u$ and $l_{r,k}, u_{r,k} \in \mathbb{R}$
- R_c is the set of simple disjunctive contingent constraints indexed by k, of the form $x_r - y_r \in \bigcup_k [l_{r,k}, u_{r,k}]$, where $y_r \in X_e \cup X_c \cup X_u$, $x_r \in X_c \cup X_u$, and $l_{r,k}, u_{r,k} \in \mathbb{R}$

PODTNUs are powerful because they allow us to model shared dependencies between different events, but they still make the assumption that external agents act without regard to the ego agent's goals and constraints. To truly take advantage of multi-agent coordination that we see in multiagent interaction, we should not assume randomness from an agent's collaborators. Instead, we should recognize that we can at least partially coordinate joint approaches to guarantee constraint satisfaction. With POSTNUs, we accomplish this by extending our model to Multiagent Simple Temporal Networks with Uncertainty (MaSTNUs) [4], and for PODTNUs, we can

similarly extend our formalism to that of Multiagent Disjunctive Temporal Networks with Uncertainty (MaDTNUs).

Definition 2. MaDTNU
An MaDTNU is a 5-tuple $\langle A, X_e, X_c, R_r, R_c \rangle$ where:

- A is a (non-empty) set of agents
- X_e is the set of executable timepoint variables
- X_c is the set of contingent timepoint variables
- R_r is the set of full disjunctive temporal constraints indexed by k, called requirement constraints, of the form $\bigvee_k \left(l_{r,k} \leq x_{r,k} - y_{r,k} \leq u_{r,k}\right)$, where $x_{r,k}, y_{r,k} \in X_e \cup X_c$ and $l_{r,k}, u_{r,k} \in \mathbb{R}$
- R_c is the set of simple disjunctive contingent constraints indexed by k, of the form $x_r - y_r \in \bigcup_k [l_{r,k}, u_{r,k}]$, where $y_r \in X_e \cup X_c$, $x_r \in X_c$, and $l_{r,k}, u_{r,k} \in \mathbb{R}$

The set of timepoints $X = X_e \cup X_c$ is partitioned across all agents in A, such that each timepoint is assigned to exactly one agent. We generally carve out an exemption for a single anchor timepoint Z, visible to all agents, that represents the start of all execution. MaDTNUs require us to specify the observability of each timepoint during execution, as the observation of events by specific agents can significantly aid in the eventual success of the scheduling process. With MaDTNUs, we assume that each timepoint, whether executable or contingent, can only be observed by the agent the timepoint is assigned to. In order to make a timepoint observable to another agent, it suffices to add a new contingent link from the original timepoint to a new contingent timepoint with zero duration that is observable by the second agent. The second agent should be able to infer the timing of the original timepoint from the contingent timepoint they can observe.

When we consider the feasibility of temporal networks with uncertainty, like PODTNUs and MaDTNUs, we cannot just validate a statically provided solution. The actual outputted schedule depends heavily on the conditions under which we observe the temporally uncertain events. As such, we often consider a temporal network's *controllability* in determining whether or not it is possible to construct a schedule. We traditionally care about the *strong, dynamic, or weak controllability* of a network, concepts that are adapted from the evaluation of STNUs [15].

Informally, strong controllability evaluates whether a single schedule can be constructed for all executable timepoints such that for any possible realization of contingent link durations, all constraints are enforced. Dynamic controllability considers whether a schedule can be constructed in a just-in-time manner given each agent's observation of existing temporal uncertainty. Finally, we say a temporal network is weakly controllable if for any realization of temporal uncertainty, there exists some schedule that satisfies all constraints for that particular realization.

Dynamic controllability tends to be the most interesting of the three forms of controllability as it provides the agents the power to react to the actual situations that manifest themselves during execution. As such, the remainder of this paper will focus on understanding the complexity of determining the dynamic controllability of these networks; we will call these problems DC-PODTNU and DC-MaDTNU for short.

In practice, we find it useful to refer to temporal networks using their graphical structures, where timepoints are represented by nodes and constraints by edges. We say that an edge exists from node A to node B if there is a constraint, or a disjunct of a constraint, that involves the difference $B - A$. We use $A \xrightarrow{R} B$ to represent a requirement constraint from A to B and $A \overset{R}{\Rightarrow} B$ to represent a contingent constraint from A to B and use R to represent the interval that the difference is constrained to belong to.

5 PODTNU CONTROLLABILITY

In the case of partially observable temporal networks, determining strong and weak controllability reduces to computing the same type of controllability over a fully observable version of the same network. This follows naturally from the definitional differences between the two different types of networks. Strong controllability assesses whether a schedule can be obstructed in absence of any observations, in essence making all contingent timepoints unobservable, whereas weak controllability assesses whether a schedule can always be constructed when given perfect foresight, adding a condition even stronger than making all contingent timepoints observable.

The same reasoning cannot be applied to dynamic controllability in PODTNUs. When dynamically executing a PODTNU, certain values remain hidden while others are readily exposed upon execution. What is quite noteworthy, however, is that the revealed value of certain observable contingent timepoints may reveal information about other unobservable ones.

This makes the question of determining DC-PODTNU more involved than that of determining dynamic controllability for ordinary DTNUs. In this section, we will show that despite this difference, DC-PODTNU complexity matches that of dynamic controllability for DTNUs, which is PSPACE-complete [2]. In order to prove that DC-PODTNU is PSPACE-complete, we must show that it is both PSPACE-hard and that it is solvable in PSPACE. We know that PODTNUs generalize DTNUs, as a DTNU is a PODTNU without unobservable contingent timepoints. Because we know checking the dynamic controllability of a DTNU is PSPACE-hard [2], we thus know that DC-PODTNU is PSPACE-hard. What remains is to show that DC-PODTNU is solvable in PSPACE.

Theorem 5.1. *DC-PODTNU ∈ PSPACE.*

Proof. In order to demonstrate this, we will provide an algorithm that checks the dynamic controllability of a PODTNU (Algorithm 1) and show that it runs in PSPACE. Our algorithm is adapted heavily from the algorithm for checking dynamic controllability of Conditional Disjunctive Temporal Networks with Uncertainty [2] and as such relies on the fact that at most a polynomial number of bits are being used to represent timepoint values. Our algorithm, however, makes no assumptions about how specifically numbers are represented and only relies on the ability of a computer to iterate through all representable numbers.

Our algorithm operates on our original PODTNU P and makes use of a DTNU D derived from it. D is constructed by removing all unobservable timepoints and modifying the related set of temporal constraints. We start with requirement constraints that involve

Input: A list of timepoints with assigned values, T;
A list of active contingent links, A;
A set of yet-to-be-executed timepoints E;
The input PODTNU P;
P's projection to a DTNU, D;
The current time, τ
Output: Whether the PODTNU is dynamically controllable.
CheckDC:

```
1  if E.empty() then
2      for realization ∈ D.realizationsFrom(A, τ) do
3          T' ← T.extend(realization);
4          for unobsRealiz ∈ P.unobsRealizFrom(T') do
5              if contingentMismatch(unobsRealiz, T') then
6                  continue;
7              if !P.isConsistent(T') then
8                  return false;
9      return true;
10 for t ∈ E do
11     for τ' ∈ [τ, P.tMax] do
12         allSatisfied ← true;
13         for realization ∈ D.realizationsFrom(A, τ) do
14             earliest ← realization.earliest();
15             if earliest.time ≤ τ' then
16                 if !CheckDC(T ∪ {earliest},
17                 A.nextContingents(earliest),
18                 E, P, D, earliest.time) then
19                     allSatisfied ← false;
20                     break;
21             else
22                 if !CheckDC(T ∪
                     {TimepointAssignment(t, τ')},
23                 A.nextContingents(TimepointAssignment(t, τ')),
24                 E \ t, P, D, τ') then
25                     allSatisfied ← false;
26                     break;
27         if allSatisfied then
28             return true;
29 return false;
```

Algorithm 1: PSPACE algorithm for checking DC-PODTNU.

unobservable contingent timepoints. For a given unobservable contingent timepoint B, we eliminate all disjuncts of requirement constraints that involve B. Note that if a fully disjunctive temporal constraint involves B in every disjunct, then the entire constraint is eliminated. Next, we consider all contingent constraints that involve B. If B is involved in more than one contingent constraint then it must be in the form $A \overset{R_1}{\Longrightarrow} B \overset{R_2}{\Longrightarrow} C$ since each contingent constraint has a unique endpoint. We can eliminate B in D by replacing each such chain of contingent links with a new single link $A \overset{R_1+R_2}{\Longrightarrow} C$, where the new constraint bounds are given using standard interval arithmetic [12]. After recursively applying this procedure, we have eliminated all constraints involving unobservable contingent timepoints and thus can safely remove those timepoints to turn our PODTNU into a DTNU. We then feed in the original PODTNU P and the derived DTNU D into our algorithm, CheckDC.

CheckDC works by recursively enumerating all possible strategies for assignments to executable timepoints and accurately simulating all possible observable timepoints of the inputted DTNU. At any given call to CheckDC, some timepoint values have been fixed, which we denote by T, and some contingent constraints have their starting timepoint executed but their ending timepoint still unexecuted, which we denote by A. We start by picking an unexecuted timepoint (line 10) and a time at which to execute it (line 11). We then look over all possible values of the ending contingent links of A (line 13) and check whether it comes before our stated execution time. If it does, we check whether our strategy still holds after we observe the ending contingent timepoint (lines 16-20), and if not, we execute our chosen timepoint and continue onwards (lines 22-26). We use *allSatisfied* to keep track of whether for any particular choice of timepoint to execute, every possible scenario still guarantees success.

It is important to note that during this part of the algorithm, we are not considering the effect of our requirement constraints and that the contingent constraints that we are considering to inform our search come from D and not P. It is possible then that our choice of values for received contingent timepoints differs from what is possible in P. For example if we originally had contingent links $A \overset{[0,10]}{\Longrightarrow} B$, $B \overset{[0,1]}{\Longrightarrow} C$, and $B \overset{[0,1]}{\Longrightarrow} C'$ in P, where only B was unobservable, our transformation to D would give us edges $A \overset{[0,11]}{\Longrightarrow} C$, and $A \overset{[0,11]}{\Longrightarrow} C'$. This would suggest that we might be able to have $A = 0, C = 0$, and $C' = 11$, but this is inconsistent with P. We will remedy this problem shortly.

Eventually, we will reach a point where E is empty because in each recursive step, we either assign a contingent timepoint (line 18) or assign an executable timepoint (line 22). When E is empty, we then check for feasibility. At this point, we consider the constraints of P. We first assign any unassigned observable timepoints (line 2) and then iterate through all possible values for the unobservable contingent values that were unassigned during execution (line 4). If we discover a scenario where the choice of observable timepoints does not match the contingent constraints of the POSTNU, we skip that particular choice (lines 5-6). If it is a valid configuration, we then check whether the POSTNU constraints are satisfied. If the POSTNU constraints are satisfied across all possible valid choices of unobservable contingent timepoints, we return true, and if not, we return false. Thus, our procedure returns the correct answer because it considers execution strategies operating over a superset of all possible situations of POSTNU P and returns true if there exists an execution strategy that satisfies all valid realizations of uncertain values.

What remains is to show that we use at most polynomial space when executing the procedure. If n is the number of timepoints in our graph, we know that there are at most n recursive calls at any one given time since each call assigns a new value to a variable. Within the algorithm, we iterate over each timepoint (line 10), each potential time (line 11), and each realization of contingent values (line 13). While there are exponentially many values each of these can take on, writing them down requires only polynomially many bits. Finally checking for valid contingent timepoint values when

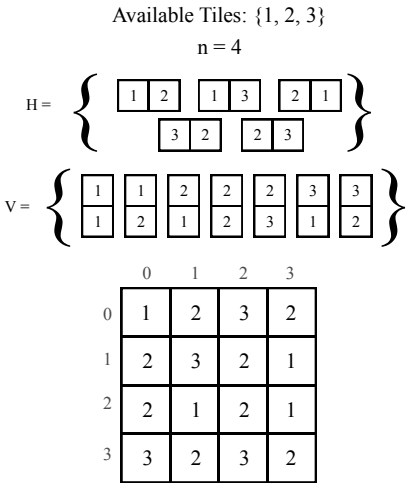

Figure 1: Example TILING problem with accompanying solution.

adding observable values and checking overall PODTNU consistency takes linear time (and thus at most linear space) and can be done simply by iterating through each constraint and checking correctness. Thus, the procedure determining dynamic controllability requires at most polynomial space.

□

Because DC-PODTNU is PSPACE-hard and can be determined with polynomial space, we know that DC-PODTNU is PSPACE-complete.

6 MADTNU CONTROLLABILITY

When we expand multiagent disjunctive reasoning to consider agents that can coordinate their strategies, the task of determining controllability becomes much more difficult.

To show that DC-MaDTNU is NEXP-hard, we first introduce the TILING problem which is itself NEXP-complete [1, 10]. The TILING problem asks whether it is possible to number a $n \times n$ board according to the following rules (see Figure 1 for reference). Each tile on the board can be filled in with a number from 1 to m, and there are a set of horizontal and vertical pairwise rules, respectively H and V, that indicate how numbers can adjoin each other on the board. We say that $f : \{0, 1, ..., n-1\} \times \{0, 1, ..., n-1\} \rightarrow \{1, 2, ..., m\}$ is a tiling function mapping from each of the row and column indices to the number on that tile. In order to ensure that the horizontal and vertical pairwise rules are respected, we require that $\forall i \in \{0, 1, ..., n-1\}, \forall j \in \{0, 1, ..., n-2\} : \langle f(i, j), f(i, j+1)\rangle \in H$ and $\langle f(j, i), f(j+1, i)\rangle \in V$. We say that a solution for the TILING problem exists if there exists an f satisfying the pairwise rules with $f(0, 0) = 1$. Note that since it takes $u = \lceil \log n \rceil$ bits to represent n, enumerating a tiling function to serve as a certificate may take exponential time.

In order to prove hardness, we simply show that TILING is reducible to DC-MaDTNU.

Lemma 6.1. *DC-MaDTNU is NEXP-hard.*

Proof. To show that DC-MaDTNU is NEXP-hard, we demonstrate how to construct an MaDTNU that is dynamically controllable if and only if a corresponding TILING problem has a valid solution.

Our strategy for the reduction is to give each of two agents a location on the TILING grid and have them report back a tile value to put at that spot without knowing which location was provided to the other agent. As such, our MaDTNU construction must first simulate the hidden random TILING grid location selection and must also enforce adjacency rules if the two agents are given locations that adjoin one another.

We construct our two-agent MaDTNU as follows (see Figure 2). The MaDTNU has a single timepoint Z that is observable by all (which for convenience we will assume is always assigned at time $t = 0$), and all other timepoints will be visible to exactly one of the agents. Each agent will have $2u$ contingent links that the other cannot see, and we say that each contingent link is made up of timepoints $A_{i,p} \xrightarrow{[0,0]\vee[2^i,2^i]} C_{i,p}$, where i is the index of the contingent link (from 0 to $2u-1$) and p represents the player it corresponds to. We add a requirement link $Z \xrightarrow{[0,0]} A_{0,p}$ for each agent as well as requirement links $C_{i,p} \xrightarrow{[0,0]} A_{i+1,p}$ for each i and p. We finally add new timepoints X_1, X_2 with requirement links enforcing $C_{2u-1,p} \xrightarrow{\{1,2,\ldots,m\}} X_p$.

With the given structure, we have a way to implicitly select a spot on the original TILING grid for each agent. The first u contingent links represent the selection of the row index and the second u contingent links represent the column index. Specifically, we can represent row index r and column index c by assigning contingent links in a way that ensures $C_{2u-1,p}$ occurs at time $r + c \cdot 2^u$. It is worth noting that in instances where n is not a power of two, there will be some assignments of contingent links that do not correspond to valid positions on the TILING grid; we will handle these situations when we consider how to enforce adjacency rules.

Before we add any of the tiling adjacency constraints, it is clear that our MaDTNU is dynamically controllable. Each $A_{i+1,p}$ is assigned immediately when $C_{i,p}$ is assigned and any valid tile value can be picked to satisfy the requirement links associated with X_1 and X_2. We now add constraints to enforce that adjacent tiles respect the pairwise tiling rules and show how those new constraints are sufficient to complete the reduction.

First, we consider how to handle contingent link values that are not valid tiling locations. In these cases, we should assume that any tiling choices are valid and ensure that all constraints are satisfied in those instances. To accommodate this for each of the constraints ψ we introduce, we amend the constraint to instead be $\phi_{oob} \vee \psi$, where:

$$\phi_{oob} = \left(C_{u-1,1} - Z \geq n\right) \vee \left(C_{u-1,2} - Z \geq n\right) \vee$$
$$\left(C_{2u-1,1} - C_{u-1,1} \geq 2^u \cdot n\right) \vee \left(C_{2u-1,2} - C_{u-1,2} \geq 2^u \cdot n\right)$$

In the instance that the contingent links are realized such that the corresponding tiling location is outside the established bounds, all constraints vacuously hold, and the network is dynamically controllable.

Now, we can move on to encoding the pairwise rules. For convenience, we will use the shorthand T_p to represent $X_p - C_{2u-1,p}$. We

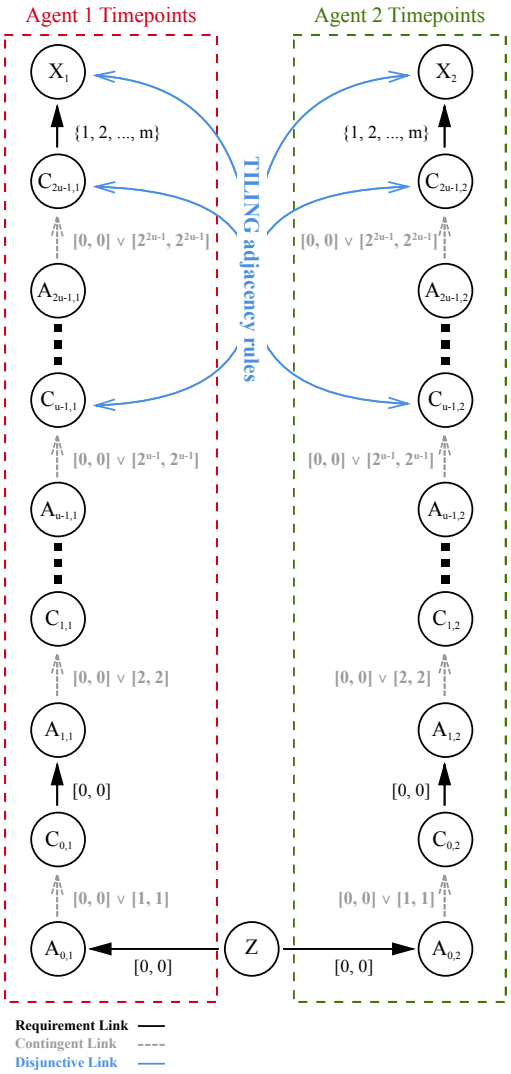

Figure 2: The two-agent MaDTNU produced by a reduction from an input TILING problem. There are $O(\log n)$ time-points in total and $O(|H| + |V| + \log n)$ constraints in total, each of which are $O(|H| + |V|)$ in size.

can write the horizontal rules as follows, starting with the instance where agent 1 must pick a tile to the left that of agent 2:

$$\bar{\phi}_{oob} \wedge \left(C_{2u-1,2} - C_{2u-1,1} = 2^u\right) \implies \langle T_1, T_2 \rangle \in H$$

To simplify our exposition, we will split this constraint into several constraints that vary based on agent 1's choice for T_1. In other words, for each $j \in \{1, 2, ..., m\}$, we now consider the constraint:

$$\bar{\phi}_{oob} \wedge \left(C_{2u-1,2} - C_{2u-1,1} = 2^u\right) \wedge (T_1 = j) \implies \langle j, T_2 \rangle \in H$$

We can rewrite our equation to eliminate the implication and since all of our values are guaranteed to be integers by construction, we can rewrite any statements involving $\neq$ with inequalities on both sides. Finally, we know that $\langle j, T_2 \rangle \in H$ is equivalent to $\bigvee_{l \in \{1,2,...,m\}: \langle j,l \rangle \in H} T_2 = l$, and substituting those values in we get:

$$\phi_{oob} \vee \left(C_{2u-1,2} - C_{2u-1,1} \geq 2^u + 1\right) \vee \left(C_{2u-1,2} - C_{2u-1,1} \leq 2^u - 1\right) \vee (T_1 \geq j + 1) \vee (T_1 \leq j - 1) \vee \bigvee_{l \in \{1,2,...,m\}: \langle j,l \rangle \in H} T_2 = l$$

It is worth noting that this approach adds $O(|H|)$ constraints each of which is $O(|H|)$ in size, meaning that our reduction still requires at most polynomial time. For completeness, we also must consider the case where agent 1 picks a tile to the right of agent 2. This requires switching the roles of agents 1 and 2 in the rules:

$$\bar{\phi}_{oob} \wedge \left(C_{2u-1,1} - C_{2u-1,2} = 2^u\right) \implies \langle T_2, T_1 \rangle \in H$$

and again, this can be expanded for each $j \in \{1, 2, ..., m\}$ into a simple disjunctive constraint of the form:

$$\phi_{oob} \vee \left(C_{2u-1,1} - C_{2u-1,2} \geq 2^u + 1\right) \vee \left(C_{2u-1,1} - C_{2u-1,2} \leq 2^u - 1\right) \vee (T_2 \geq j + 1) \vee (T_2 \leq j - 1) \vee \bigvee_{l \in \{1,2,...,m\}: \langle j,l \rangle \in H} T_1 = l$$

We can take a similar approach for satisfying the vertical rules, seeing if the final contingent link values of the two agents differ by exactly one, but in this case, we now need to make sure that the two values are still in the same column instead of being wrapped around to a new one. To accommodate this, we now have to additionally verify that the larger of the two tile indices is not in the 0^{th} row. This yields constraints of the form:

$$\bar{\phi}_{oob} \wedge \left(C_{2u-1,2} - C_{2u-1,1} = 1\right) \wedge \left(C_{u-1,2} \neq 0\right) \implies \langle T_1, T_2 \rangle \in V$$

and:

$$\bar{\phi}_{oob} \wedge \left(C_{2u-1,1} - C_{2u-1,2} = 1\right) \wedge \left(C_{u-1,1} \neq 0\right) \implies \langle T_2, T_1 \rangle \in V$$

By applying the same approach as with the horizontal rules, we again create distinct constraints for each $j \in \{1, 2, ..., m\}$ and can rewrite our rules to get simple disjunctive constraints:

$$\phi_{oob} \vee \left(C_{2u-1,2} - C_{2u-1,1} \geq 2\right) \vee \left(C_{2u-1,1} - C_{2u-1,2} \leq 0\right) \vee \left(C_{u-1,2} = 0\right) \vee (T_1 \geq j + 1) \vee (T_1 \leq j - 1) \vee \bigvee_{l \in \{1,2,...,m\}: \langle j,l \rangle \in V} T_2 = l$$

and:

$$\phi_{oob} \vee \left(C_{2u-1,1} - C_{2u-1,2} \geq 2\right) \vee \left(C_{2u-1,1} - C_{2u-1,2} \leq 0\right) \vee \left(C_{u-1,1} = 0\right) \vee (T_2 \geq j + 1) \vee (T_2 \leq j - 1) \vee \bigvee_{l \in \{1,2,...,m\}: \langle j,l \rangle \in V} T_1 = l$$

Finally, for the sake of simplicity, we will also require that if the two agents receive the same value from their contingent links, then they must report the same T_p. This adds the constraint:

$$\bar{\phi}_{oob} \vee \left(C_{2u-1,1} = C_{2u-1,1}\right) \implies T_1 = T_2$$

which can be rewritten as:

$$\phi_{oob} \vee \left(C_{2u-1,1} \geq C_{2u-1,1} + 1\right)$$

$$\vee \left(C_{2u-1,1} \leq C_{2u-1,1} - 1\right) \vee (X_1 - X_2 = 0)$$

It is clear by construction that if a solutions exists for a TILING grid then the MaDTNU we constructed is dynamically controllable. An acceptable strategy is for both agents to precompute a shared valid tiling and to use that tiling to pick X_1, X_2 based on the values of the contingent links each can observe. What remains is to demonstrate that knowing that our constructed MaDTNU is dynamically controllable implies that the corresponding TILING problem has a valid solution.

Our decision to add a constraint requiring that both agents report the same value when queried for the same tile simplifies our analysis, as it requires the two agents to have the same, deterministic strategy. Thus, our problem reduces to being able to prove that a solution to the TILING problem exists if the agents have a strategy that renders the network controllable.

We start with the observation that each agent really has only one decision point, namely when to schedule X_p. Each of the intermediate $A_{i,p}$s are scheduled immediately after the corresponding preceding contingent timepoints, so there are no real decisions to be made in scheduling $A_{i,p}$. There are 2^{2u} possible values that timepoint $C_{2u-1,p}$ can be assigned to, but thanks to ϕ_{oob}, we only care about n^2 of them. If we take the agent's strategy for those n^2 time points that correspond to a row-column indexing into the original grid, we can immediately translate that into a valid tiling. We prove this by contradiction.

Assume momentarily that translating the first agent's strategy does not yield a viable tiling. If the agent's strategy were stochastic, we can take any possible grounded strategy and use that as our main one, as controllability implies that all constraints are satisfied across all possible agent strategy realizations. This means that for the tiling to be invalid, there must be a pair of tiles that adjoin one another that violate the original tiling rules. Let's call these indices i and j and the corresponding tiles assigned by agent 1 for these indices T_i and T_j.

We know that in order for the system to be controllable, agent 2's strategies for i and j must also be to assign T_i and T_j because of the rule that requires reporting the same values when the contingent link values are all the same. Because our original MaDTNU was known to be dynamically controllable, this means that for all possible uncertain values all constraints are satisfied, including when agent 1 must assign a value for index i and when agent 2 must assign a value for index j. In this situation, they must produce values T_i and T_j, respectively, but because indexes i and j adjoin one another and all constraints are known to be satisfied, then T_i and T_j must satisfy the appropriate adjacency relation. This means that the tiling derived from the strategy cannot have a pair of indices that adjoin one another and violate a rule, concluding the proof that TILING is reducible to DC-MaDTNU and that DC-MaDTNU is NEXP-hard.

□

Our reduction demonstrates that DC-MaDTNU is NEXP-hard. Now, we show that DC-MaDTNU can be solved by a non-deterministic Turing machine in exponential time.

Lemma 6.2. *DC-MaDTNU ∈ NEXP.*

Proof. To show that DC-MaDTNU ∈ NEXP, we will generate strategies for each agent and show that the joint execution of these strategies guarantees success.

First, we need to generate strategies for each agent. We know that it is possible to describe a dynamic execution strategy for an agent in a DTNU that can be efficiently executed [5]. Even though these strategies when enumerated can be exponential in the size of the input, this will suffice for our purposes. Note that an important part of this approach is that we assume that it takes a fixed number of bits (though possibly polynomially many) to represent individual numbers but is agnostic as to how specifically numbers are represented.

In order to construct a strategy for the overall MaDTNU, we will start by guessing a strategy for each agent with respect to their locally projected timepoints. We define the locally projected timepoints of an agent in an MaDTNU as the collection of timepoints that are directly observable by that agent. Whereas timepoints in the MaDTNU were subdivided into timepoints assigned by the ego agent, timepoints assigned by nature, and timepoints assigned by each of the other agents, the locally projected timepoints will only be subdivided into timepoint assigned by the ego agent and timepoints assigned by others.

Given a set of locally projected timepoints, we guess a DTNU strategy non-deterministically over that set of timepoints. Note that we are guessing this strategy without explicit knowledge of any dependencies between timepoints or decisions strategically made by other agents; at this moment, we are leaning on non-determinism to generate local strategies for each agent that are together globally consistent.

The strategies we generated can each be exponentially large, but, importantly, it takes at most exponential time to guess an exponentially large string. Through the strategy generation step, we are still well within our established time bounds.

Now, we must validate that the strategies we guessed ensure dynamic controllability. Or that for any possible realization of the uncertain duration of links, we can still guaranteeably satisfy all links. We can do so by brute force iteration (see Algorithm 2).

Our brute force enumeration relies on the fact that the bits required to encode any particular uncertain state are polynomial in the problem input size. In other words, writing down a realization of contingent link values takes at most polynomial space even though there are exponentially many such realizations.

For any given eventual realization of contingent link durations, a fixed strategy will yield a deterministic output. Our algorithm gives a straightforward simulation operating on behalf of each individual agent. We build out our simulation by iteratively grounding the values of individual timepoints based on agent strategies and the different realized durations of contingent links; these values are stored in the *assigned* variable (line 3).

Each agent's strategy only allows them to observe a subset of timepoints and make decisions off of those timepoints, and those decisions reduce to unconditionally executing a timepoint at a certain point in time or conditionally waiting to observe an uncontrolled timepoint before making a decision. In the event that an agent's action is to conditionally wait for another timepoint, we instead

Input: An MaDTNU G
Output: Whether G is dynamically controllable.
Initialization:
1 $strategies \leftarrow$ guessed local strategies for each agent;
DC-MaDTNU:
2 **for** $\omega \in G.uncertainRealizations()$ **do**
3 $assigned \leftarrow \emptyset$;
4 $time \leftarrow 0$;
5 **while** $len(assigned) < G.numTimepoints()$ **do**
6 $possibleActions \leftarrow \{s.nextAction(assigned, time) \mid s \in strategies\}$;
7 $nextAction \leftarrow possibleActions.earliest()$;
8 $possibleConts \leftarrow \{\langle y, t\rangle \mid y \notin assigned \wedge \exists x \in assigned : \langle x, y, t\rangle \in G.contingents()\}$;
9 $nextCont \leftarrow possibleConts.earliest()$;
10 **if** $nextAction.before(nextCont)$ **then**
11 $assigned.add(\langle nextAction.timepoint(), nextAction.time()\rangle)$;
12 $time \leftarrow nextAction.time()$;
13 **else**
14 $assigned.add(\langle nextCont.timepoint(), nextCont.time()\rangle)$;
15 $time \leftarrow nextCont.time()$;
16 **if** $G.constraintsViolated(assigned)$ **then**
17 **return** $false$;
18 **return** $true$;

Algorithm 2: NEXP algorithm for checking DC-MaDTNU.

record the agent that action would take if that uncontrolled timepoint took on its latest allowable value; these actions are chosen at line 6 from the nondeterministically guessed agent strategy.

While some uncontrollable actions are chosen by other agents, some are controlled by nature through contingent links. Though the durations of these contingent links are determined when we grounded them (line 2), their values are not yet visible to the agents and so must be learned iteratively. We can imagine that nature behaves like a non-cooperative (or for the sake of controllability checking, even adversarial) agent in the way that it picks its timepoints, and so similarly consider the next contingent values to be realized (line 8). We update the *assigned* variable one timepoint at a time, selecting the earliest values from the derived agent actions and contingent link values (lines 7, 9, 10) to ensure that strategies can be adjusted based on new information.

Now we show that the process as a whole takes at most exponential time on a non-deterministic Turing machine.

We have already demonstrated that it takes exponential time to generate the strategies at line 1 of Algorithm 2, and since there are exponentially many realizations of contingent link uncertainties, we turn our focus to the runtime of the body of the for loop at line 2. The while loop goes through $O(n)$ iterations in total since *assigned* grows by one after each iteration (lines 11 and 14). The generation of *possibleConts* at line 8 takes at most $O(n)$ time since there are at most n contingent links, but the most expensive part of the process is the generation of *possibleActions* at line 6. Since each strategy is exponentially large, and it takes time linear in strategy size to determine what action to take next, the strategy generation at line 6 takes exponential time. However, that this exponential time operation happens an exponential number of times still guarantees that the overall runtime of the algorithm is exponential. Thus, Algorithm 2 runs in NEXP time, and DC-MaDTNU $\in$ NEXP.

□

By Lemma 6.1, we know that DC-MaDTNU is NEXP-hard, and by Lemma 6.2, we know that DC-MaDTNU $\in$ NEXP. Thus, DC-MaDTNU is NEXP-complete.

7 DISCUSSION

In this paper, we took a deep look at multiagent disjunctive temporal networks in order to better understand the feasibility of constructing schedules with such models. While constructing a schedule for PODTNUs is a PSPACE-complete problem, guaranteeing the existence of a schedule for MaDTNUs in a dynamically controllable setting is NEXP-complete.

Modelers have many options when investigating a problem, including simplifying their models in order to guarantee faster runtimes. While MaDTNUs provide a high degree of fidelity for modelers, the extreme complexity of deriving a solution makes it an undesirable framework to use in practice.

Taking a pragmatic approach, we have two axes against which we can select our model. The first axis considers whether we admit disjunctive constraints and the latter considers the fidelity of multiagent interactions. If we assume a fully observable model of multiagent uncertainty, our two options are DTNUs and STNUs. While dynamic controllability can be computed for the former is PSPACE-complete [2], the latter can be determined in $O(n^3)$ time [8]. When we expand our views to include partial observability, we see that while DC-PODTNU has the same computational complexity as dynamic controllability checking for DTNUs, we do not yet know the computational complexity of checking the controllability of POSTNUs or even MaSTNUs. While we do have polynomial time algorithms for checking the dynamic controllability of POSTNUs [3] and MaSTNUs [4], these algorithms are not complete.

Here, we seek to highlight the future importance of investigating the theoretical complexity of dynamic controllability checking for POSTNUs and MaSTNUs. Our work demonstrates that adding partial observability to DTNUs has no impact on the computational complexity of solving the problem, but it is not immediately clear whether the same can be said for STNUs and if they can, whether those benefits continue to hold for full multiagent networks. While current work has established that certain POSTNUs and MaSTNUs can be checked for controllability in polynomial time, proving a result analogous to the one we present here would significantly expand the set of situations that can be modeled and evaluated efficiently. We believe addressing this question represents an important avenue for future research.

ACKNOWLEDGMENTS

This work was partially supported by the Toyota Research Institute (TRI). However, this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

REFERENCES

[1] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research* 27, 4 (2002), 819–840.

[2] Nikhil Bhargava and Brian Williams. 2019. Complexity Bounds for the Controllability of Temporal Networks with Conditions, Disjunctions, and Uncertainty. *Artificial Intelligence* 271 (2019), 1–17.

[3] Arthur Bit-Monnot, Malik Ghallab, and Félix Ingrand. 2016. Which contingent events to observe for the dynamic controllability of a plan. In *International Joint Conference on Artificial Intelligence (IJCAI-16)*.

[4] Guillaume Casanova, Cédric Pralet, Charles Lesire, and Thierry Vidal. 2016. Solving Dynamic Controllability Problem of Multi-Agent Plans with Uncertainty Using Mixed Integer Linear Programming.. In *ECAI*. 930–938.

[5] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. 2016. Dynamic Controllability of Disjunctive Temporal Networks: Validation and Synthesis of Executable Strategies.. In *AAAI*. 3116–3122.

[6] Rina Dechter, Itay Meiri, and Judea Pearl. 1991. Temporal constraint networks. *Artificial intelligence* 49, 1-3 (1991), 61–95.

[7] Michael D Moffitt. 2007. On the partial observability of temporal uncertainty. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 22. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 1031.

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

[9] Paul H Morris and Nicola Muscettola. 2005. Temporal dynamic controllability revisited. In *AAAI*. 1193–1198.

[10] C.H. Papadimitriou. 1994. Computational Complexity. *Addison-Wesley Reading* (1994).

[11] Kostas Stergiou and Manolis Koubarakis. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120, 1 (2000), 81–117.

[12] Teruo Sunaga. 1958. Theory of interval algebra and its application to numerical analysis. *RAAG memoirs* 2, 29-46 (1958), 209.

[13] Kristen Brent Venable, Michele Volpato, Bart Peintner, and Neil Yorke-Smith. 2010. Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *Workshop on constraint satisfaction techniques for planning & scheduling*. 50–59.

[14] Kristen Brent Venable and Neil Yorke-Smith. 2005. Disjunctive Temporal Planning with Uncertainty.. In *IJCAI*. 1721–1722.

[15] Thierry Vidal and Helene Fargier. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence* 11, 1 (1999), 23–45.