Propagating Piecewise-Linear Weights in Temporal Networks

Luke Hunsberger[*]
Computer Science Department
Vassar College – Poughkeepsie, NY USA
hunsberger@vassar.edu

Roberto Posenato
Dipartimento di Informatica
Università degli Studi di Verona, Italy
roberto.posenato@univr.it

Abstract

This paper presents a novel technique using piecewise-linear functions (PLFs) as weights on edges in the graphs of two kinds of temporal networks to solve several previously open problems. Generalizing constraint-propagation rules to accommodate PLF weights requires implementing a small handful of functions. Most problems are solved by inserting one or more edges with an initial weight of δ (a variable), then using the modified rules to propagate the PLF weights. For one kind of network, a new set of propagation rules is introduced to avoid a non-termination issue that arises when propagating PLF weights. The paper also presents two new results for determining the tightest horizon that can be imposed while preserving a network's dynamic consistency/controllability.

Introduction

Planning and scheduling applications have used Simple Temporal Networks (STNs) for many years (Dechter, Meiri, and Pearl 1991; Gerevini, Saetti, and Serina 2006; Casanova, Pralet, and Lesire 2015). Unlike fixed-time schedules, STNs enable flexible management of common types of temporal constraints (e.g., release times, deadlines, and duration constraints). Polynomial-time algorithms can check the consistency of STNs and manage their real-time execution; however, STNs cannot deal with uncertainty. This paper focuses on extensions of STNs that accommodate two kinds of uncertainty. A *Simple Temporal Network with Uncertainty* (STNU) uses *contingent links* to represent actions whose durations are uncertain, but within known bounds; a *Conditional Simple Temporal Network* (CSTN) uses *observation time-points* to represent test actions that generate information during execution. An STNU is *dynamically controllable* if there is a strategy for executing its time-points that guarantees that all relevant constraints will be satisfied no matter how the contingent durations turn out; a CSTN is *dynamically consistent* if there is a strategy for executing its time-points that guarantees that all relevant constraints will be satisfied no matter what true-or-false answers the test actions generate. These related properties are both abbreviated by DC.

Different flavors of the DC property for STNUs and CSTNs depend on the assumptions made about the ability of the system to react to observations. The original version of DC assumes that the system can react after any arbitrarily small, positive delay. Another version assumes that the system can react instantaneously. Finally, an ϵ-DC property has been defined for CSTNs that assumes that the system's reaction time is bounded below by some fixed $\epsilon > 0$. An ϵ-DC property can also be defined for STNUs.

DC-checking algorithms have been given for most flavors of DC for STNUs and CSTNs (Morris 2006; Cairo, Hunsberger, and Rizzi 2018; Cairo and Rizzi 2016; Hunsberger and Posenato 2018b). The fastest DC-checking algorithm for STNUs has cubic time (Morris 2014), but the DC-checking problem for CSTNs is PSPACE-complete (Cairo and Rizzi 2016). However, propagation-based algorithms for CSTNs have shown promise (Hunsberger and Posenato 2018b).

Despite the many DC-checking algorithms for STNUs and CSTNs, many open problems remain, including: (1) given any time-points X and Y, what is the strongest constraint, $Y - X \leq \delta$, that can be added to the network while preserving the DC property?; (2) after executing the *zero time-point* Z, how much time can elapse before some other time-point must be executed?; and (3) if the reaction time of the dynamic strategy is bounded below by some fixed $\epsilon > 0$, what is the maximum value of ϵ that will preserve the DC property? And, for STNUs: (4) how much can the bounds on a contingent link be *loosened* while preserving the DC property?

This paper introduces a novel technique that can be used to solve all of the problems listed above. The technique generalizes the numerical weights of edges in an STNU or CSTN to piecewise-linear functions (PLFs). Although there may be many PLF weights in a given network, the domain of each PLF is the same. Thus, each PLF is a function over the same variable δ.[1] By extending the constraint-propagation rules for existing DC-checking algorithms to accommodate PLF weights, our technique is able to answer the sorts of questions listed above by determining the bounds on δ needed to preserve the DC property. Our algorithm computes an exact value for δ, whereas an approach based on binary search

[*] This work was funded in part by grants from the Lucy Maynard Salmon Research Fund and the University of Verona (*Programma COOPERINT 2018*).

[1] For this reason, our technique is completely different from prior work on STNs with preferences or fuzzy constraints (Kumar 2004; Morris et al. 2005; Rossi, Venable, and Yorke-Smith 2006).

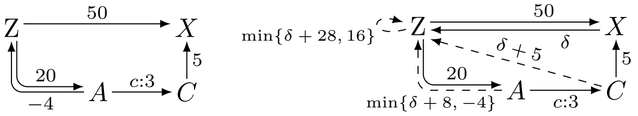


Figure 1: A sample STNU (a, left) before, and (b, right) after propagating edge-weights involving the variable δ

would compute an approximate value with a computation time dependent on the granularity of the temporal domain.

Overview of our Approach

In temporal networks, each constraint has a numerical weight. Our approach is more general in that it allows the weights of edges to be piecewise-linear functions—with the important caveat that all of the PLFs must be functions of the same variable, δ. To illustrate the need for PLFs, consider the network in Fig. 1a, which is a fragment of an STNU that can easily be checked to be DC. In this network, the interval from A to C is a *contingent link* with a minimum duration of 3, represented by the edge from A to C labeled by c:3.[2] Given that Z is fixed at 0, we can ask, “What is the maximum amount that the execution of X can be delayed?” To answer this question, we first insert a new edge from X to Z labeled not by a number, but by a variable δ, as shown in Fig. 1b. We seek the *minimum* value of δ that will preserve the DC property. (A more negative value of δ represents a greater lower bound for X.) Since any $\delta < -50$ would introduce a negative loop from Z to X to Z, corresponding to an unsatisfiable constraint, we must have $\delta \geq -50$. And since all time-points are implicitly assumed to occur at or after Z, we set the initial domain for δ to be the interval $[-50, 0]$.

Next, we propagate this new kind of constraint/edge across the network, not for one value of δ, *but for all values of δ simultaneously.* We do this by making slight modifications to a well known set of constraint-propagation rules for STNUs (Morris, Muscettola, and Vidal 2001; Morris and Muscettola 2005). For example, applying one of those rules to the edges from C to X to Z generates a new edge from C to Z of length $\delta + 5$, indicated by a dashed arrow in the figure. More interestingly, the next propagation, applied to the edges from A to C to Z, involves the *lower-case edge* from A to C. In this case, the propagation applies only if the weight of the edge from C to Z is negative. Therefore, the generated edge from A to Z has length $\delta + 8$—but only if $\delta + 5 < 0$ (i.e., only if $\delta < -5$).

With numerical weights, when a new edge from A to Z is generated, processing the new edge is easy: if the weight of the new edge is less than the weight of the pre-existing edge from A to Z (or if there is no pre-existing edge), then the new edge is added to the network, replacing the old edge; otherwise, the new edge is discarded. However, when the weights are piecewise-linear functions, the weight on the new edge may be stronger than the pre-existing weight for some values of δ, but weaker for others. As a result, the proper update is to set the weight of the edge to be the PLF obtained by taking the *minimum* of the old PLF and new PLF. In our example, the pre-existing edge from A to Z has weight -4 (constant, for all values of δ), while the new weight is $\delta + 8$, but only for $\delta < -5$. Hence, the appropriate update is to set the edge's weight to be $\min\{\delta + 8, -4\}$, as shown in Fig. 2.

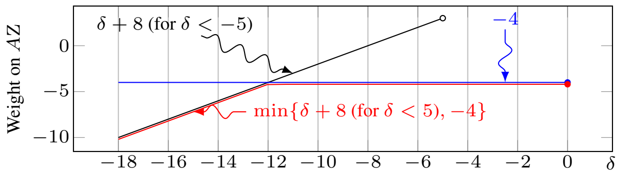


Figure 2: Taking the minimum of piecewise-linear functions

Finally, the path from Z to A to Z generates a self-loop at Z whose weight is $\min\{\delta + 8, -4\} + 20 = \min\{\delta + 28, 16\}$. *Since a negative loop represents an unsatisfiable constraint, the PLF weight on the loop at* Z *must be non-negative to preserve the DC property.* In this case, $\delta + 28 \geq 0$ (i.e., $\delta \geq -28$) must hold. *In this way, constraint propagation caused the set of possible values for δ to shrink,* which is a key feature of our approach. The implication for this toy example is that the greatest lower bound for X is 28.

Although this example focused on determining the greatest lower bound for a time-point X in a DC STNU, the same technique of using PLFs to represent the weights in a temporal network can be used to answer all of the kinds of questions listed earlier. Although this paper will focus on STNUs and CSTNs, the same techniques can be applied to CSTNUs—since prior work has shown how to reduce any DC CSTNU to an equivalent CSTN (Hunsberger and Posenato 2018c).

Piecewise-Linear Functions (PLFs). In general, a piecewise-linear function can take many forms; however, for our approach, it suffices to restrict attention to PLFs with the following characteristics.

- Each PLF is a non-decreasing function $f \colon \mathbb{R} \to \mathbb{R}$ of δ.
- The domain of each PLF is a union of *adjacent* intervals, each of the form $[a, b)$—hence, no holes in the domain.

For efficiency, the portion of the domain of a PLF f where $f(\delta) = \infty$, which, given the above assumptions, must be an interval of the form $[a, \infty)$, is not explicitly represented.

To propagate edges whose weights are represented by PLFs, we need to be able to compute the following for any PLFs f and g, and any $x \in \mathbb{R}$:

- $\mathit{sum}(f, x)$, $\mathit{min}(f, x)$, $\mathit{max}(f, x)$; (a PLF and a number)
- $\mathit{sum}(f, g)$, $\mathit{min}(f, g)$, $\mathit{max}(f, g)$; (two PLFs)
- $\delta_{\text{min}}^{f \geq 0} = \min\{\delta \mid f(\delta) \geq 0\} \in \mathbb{R}$;
- $f|^{<0}$: the PLF that is the same as f except that it includes only the portion of f's domain where $f(\delta) < 0$.
- $f|_{\delta \geq x}$: the PLF that is the same as f except that it includes only the portion of f's domain where $\delta \geq x$.
- $\mathit{merge}(f)$: same as f except that any adjacent intervals in its domain that involve the same function are merged.

[2] Contingent links and constraint-propagation rules for STNUs are addressed lightly here, but in great detail in the next section.

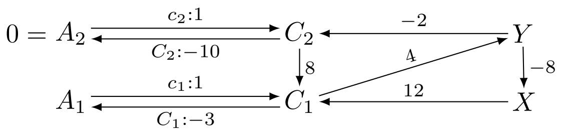


Figure 3: A sample STNU

Note: the superscript in $f|^{<0}$ stipulates that it is the *function* values that must be negative, whereas the subscript in $f|_{\delta \geq x}$ stipulates that it is the *domain* values that must be at least x.

Simple Temporal Networks with Uncertainty

Simple Temporal Networks with Uncertainty (STNUs) extend STNs to include *contingent links* which can be used to represent actions with uncertain durations (Morris, Muscettola, and Vidal 2001). Each contingent link has the form (A, x, y, C). A is called the *activation* time-point, C is called the *contingent* time-point, and $0 < x < y < \infty$. The link is activated when A is executed. After that, the execution of C is determined not by the executing agent (controller), but by the (possibly adversarial) environment. But the environment must execute C some time after A, such that $C - A \in [x, y]$. The duration $C - A$ is controlled by the environment, and is unknown to the controller until C is actually executed.

Graphically, a contingent link is represented by two labeled edges, a *lower-case* edge from A to C with label $c{:}x$, and an *upper-case* edge from C to A with label $C{:}{-y}$. The graph for an STNU with two contingent links, $(A_1, 1, 3, C_1)$ and $(A_2, 1, 10, C_2)$, is shown in Fig. 3. The value of A_2 is fixed at 0 (i.e., A_2 is serving as the zero time-point Z).

An STNU is *dynamically controllable* (DC) if there exists a *dynamic* strategy for the controller to execute all of the *non-contingent* (a.k.a., *executable*) time-points such that all of the constraints in the network are guaranteed to be satisfied no matter how the durations of the contingent links are chosen by the environment. The strategy can be *dynamic* in that the execution time it chooses for each executable time-point X can only depend on the durations of contingent links that have already completed (i.e., the strategy's execution decisions must not depend on future events). The STNU in Fig. 3 is DC since the following dynamic strategy ensures that all of its constraints will be satisfied: *Execute A_2 and X at 0. If C_2 executes at or before 2, then execute A_1 at 6 and Y at 8; otherwise, execute A_1 at 7 and Y at 12.*

The DC-checking problem for STNUs is that of determining whether any given STNU is DC. The fastest DC-checking algorithms for STNUs rely on rules for propagating constraints. Table 1 lists five constraint-propagation rules for STNUs that will be used in this paper. The first four rules are from Morris and Muscettola (2005); the last rule (i.e., LR$^+$) combines their Label Removal (LR) rule with the General Unordered Reduction (GUR) from Morris et al. (2001). The No Case (NC) rule is the constraint-propagation rule for STNs. The Upper Case (UC) rule generates a conditional *wait* constraint that guards against the contingent link (A, x, y, C) taking on its maximum value. The Lower Case (LC) rule generates an ordinary edge that guards against the contingent link taking on its minimum value. The Cross Case rule guards against one contingent link taking on its minimum value, while another takes on its maximum. The LR$^+$ rule specifies an unconditional (ordinary) constraint derived from a conditional (upper-case) constraint. This collection of rules is sound and complete for the STNU DC-checking problem. The edge-weights in these rules are all real numbers.

NC:	$X \xrightarrow{f} Y \xrightarrow{g} W$ $X \overset{f+g}{\dashrightarrow} W$	
UC:	$X \xrightarrow{f} Y \xrightarrow{C:g} A$ $X \overset{C:f+g}{\dashrightarrow} A$	
LC:	$A \xrightarrow{c:x} C \xrightarrow{g} Y$ $A \overset{x+g}{\dashrightarrow} Y$	$g < 0$
CC:	$A \xrightarrow{c:x} C \xrightarrow{C':g} A'$ $A \overset{C':x+g}{\dashrightarrow} A'$	$C \not\equiv C'$ $g < 0$
LR$^+$:	$Y \xrightarrow{C:f} A \xrightarrow{c:x} C$ $Y \overset{\max\{f,-x\}}{\dashrightarrow} A$	

Table 1: Known constraint-propagation rules for STNUs

Our novel approach to answering a variety of open questions about STNUs employs a simple, but powerful technique: allowing the weight on a constraint/edge to be a piecewise-linear function, and then modifying the STNU constraint-propagation rules to accommodate PLFs.

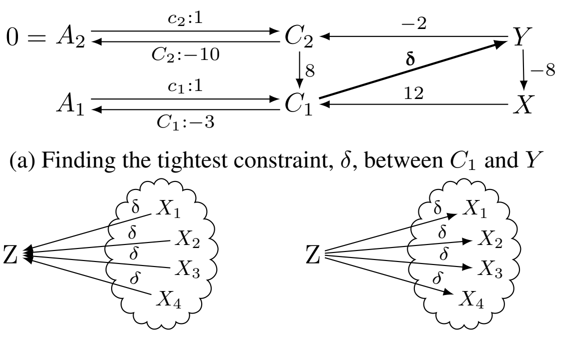
Modifying STNU propagation rules for PLFs. The rules in Table 1 can be modified to accommodate weights that are PLFs, as follows, where, now, f and g are the respective PLFs for the lefthand and righthand pre-existing (ordinary or upper-case) edges in the rules. (A numerical weight is a special case of a PLF with only one piece.)

- No Case: $sum(f, g)$;
- Upper Case: $sum(f, g)$, generated edge labeled by C;
- Lower Case: $sum(g|^{<0}, x)$;
- Cross Case: $sum(g|^{<0}, x)$, generated edge labeled by C;
- Label Removal/General Unordered Red'n: $max(f, -x)$.

In addition, whenever a rule generates an edge, say, from X to Y whose weight is the PLF h, the appropriate update is determined as follows. First, h is replaced by $merge(h)$ (i.e., adjacent pieces of h that represent the same linear function are merged). Next, if there is no pre-existing edge, then the (merged) h becomes the weight of a new edge from X to Y. However, if there is a pre-existing edge from X to Y with weight H (another PLF), then the new weight for that edge is $min(h, H)$ (i.e., the minimum of the new and old PLFs). Finally, if a self loop with weight h is generated, then any values δ for which $h(\delta) < 0$ must be excluded from the domain of *every* PLF appearing in the network. In other words, $\delta_{\text{min}}^{h \geq 0}$ becomes the new lower bound for *every* PLF in the network (i.e., each f is replaced by $f|_{\delta \geq \delta_{\text{min}}^{h \geq 0}}$).

⇒ Restricting the global domain for δ is what determines the minimum value of δ that preserves the DC property.

How much can an edge be tightened? Consider the edge from C_1 to Y. To compute the minimum weight for this



(a) Finding the tightest constraint, δ, between C_1 and Y

(b) Finding the maximum delay (c) Finding the tightest horizon

Figure 4: Solving different problems for a single STNU

edge that will preserve the DC property, we replace the numerical weight by the variable δ, as shown in Fig. 4a. Exhaustively propagating the rules from Table 1 generates the *semi-reducible loop* shown in Fig. 5. In the figure, the edges generated by the rules are dashed, and the name of the rule appears to the left of the corresponding labeled value. For example, the last propagation, which generates the loop from A_2 to A_2, is annotated by UC: $\max\{2\delta - 3, \delta - 12\}$, for $\delta < 2$, indicating that the Upper-Case rule generated this edge. Since the weight on a self-loop must be non-negative, we must have $2\delta - 3 \geq 0$ (i.e., $\delta \geq \frac{3}{2}$) to preserve the DC property. The reason that a network with only integer weights could lead to a fractional answer is that, in an STNU graph, a shortest *semi-reducible path* can involve multiple occurrences of a single edge (Hunsberger 2013). The worst case is a *magic loop,* in which a single edge can appear 2^{k-1} times, where k is the number of contingent links. In our case, the edge from C_1 to Y appears twice in the shown semi-reducible loop. As a result, each unit decrease in its weight leads to two units of tightening of the semi-reducible loop.

Delaying initial executions. Consider the problem of determining how much the execution of the other time-points in the network can be delayed after Z = 0. To find out, we insert an edge from each executable time-point to Z, as illustrated in Fig. 4b, and then exhaustively apply the propagation rules from Table 1. For the sample STNU from Fig. 3, A_2 plays the role of Z, and exhaustive propagation leads to the result that $\delta \geq -3$ (i.e., once A_2 has been executed, some executable time-point—here, X—must execute *at or before* time 3).

Computing maximum reaction time. Given some $\epsilon > 0$, an execution strategy is called ϵ*-dynamic* if it is only able to react to an observation of a contingent execution after a delay of ϵ; and an STNU is called ϵ*-dynamically controllable* (ϵ-DC) if it admits an ϵ-dynamic strategy that guarantees the satisfaction of all relevant constraints no matter how the contingent durations turn out.[3] A sound-and-complete ϵ-DC-checking algorithm can be obtained by making slight changes to the LC and CC rules from Table 1, as shown in Table 2.[4] Using those rules, it can be shown that the sample STNU from Fig. 3 is ϵ-DC for $\epsilon = 2$, but not for $\epsilon = 8$.

LC$^\epsilon$:	$A \xrightarrow{c:x} C \xrightarrow{g} Y$ $x+g$	$g < \epsilon$
CC$^\epsilon$:	$A \xrightarrow{c:x} C \xrightarrow{C':g} A'$ $C'{:}x+g$	$C \not\equiv C'$ $g < \epsilon$

Table 2: STNU propagation rules for ϵ-DC checking

For any STNU $\mathcal{S}$, let $\hat{\epsilon}(\mathcal{S})$ denote the *maximum* value of ϵ for which $\mathcal{S}$ is ϵ-DC.[5] Our approach can be used to compute $\hat{\epsilon}(\mathcal{S})$ for any STNU, as follows. First, let $\epsilon \in (0, \infty)$ be the sole parameter, and view the (initial) weight of each edge as a *constant function* of ϵ. Now, consider an application of the LC rule from Table 2 where $x = 2$ and $g = 4$. The applicability condition $g < \epsilon$ (i.e., $4 < \epsilon$) stipulates that the generated edge of length $2 + 4 = 6$ is valid only for $\epsilon > 4$. Therefore, if the pre-existing weight on that edge was, say, 9, then the updated weight for this edge would be the step function, $f(\epsilon) = 9$, for $\epsilon \in (0, 4]; 6$, otherwise. Similarly, applications of the CC$^\epsilon$ rule from Table 2 can result in step functions. It follows that, since all edge weights were initially constants, and the only use of ϵ is by the LC$^\epsilon$ and CC$^\epsilon$ rules to restrict the domain of an edge weight, *all* generated edges will necessarily be step functions (i.e., PLFs for which each piece is a constant function). As always, should any rule application generate a self-loop, then the values of ϵ for which the weight of that loop is negative must be excluded from the global domain. For example, applying our approach to the STNU from Fig. 3 determines that the maximum reaction time that will preserve the DC property is $\hat{\epsilon} = 5$. Fig. 6 shows the negative loop

[3] Bhargava et al. (2018) defined *delay controllability,* where contingent links can have different delays. Our ϵ-DC for STNUs assumes a fixed lower bound ϵ for a strategy's reaction time.

[4] The rules are from Bhargava et al. (2018), but with fixed delays.

[5] Our definition of $\hat{\epsilon}(\mathcal{S})$ for STNUs mirrors the definition of $\hat{\epsilon}(\mathcal{S})$ for CSTNs (Comin and Rizzi 2015).

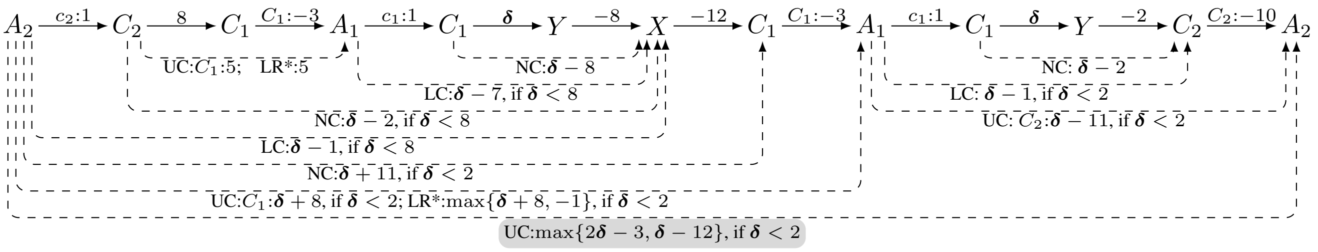

A_2
$c_2{:}1$
C_2
8
C_1
$C_1{:}-3$
A_1
$c_1{:}1$
C_1
δ
Y
-8
X
-12
C_1
$C_1{:}-3$
A_1
$c_1{:}1$
C_1
δ
Y
-2
C_2
$C_2{:}-10$
A_2
UC:C_1:5; LR*:5
NC:$\delta - 8$
LC:$\delta - 7$, if $\delta < 8$
NC:$\delta - 2$, if $\delta < 8$
LC:$\delta - 1$, if $\delta < 8$
NC:$\delta + 11$, if $\delta < 2$
UC:C_1:$\delta + 8$, if $\delta < 2$; LR*:$\max\{\delta + 8, -1\}$, if $\delta < 2$
NC: $\delta - 2$
LC: $\delta - 1$, if $\delta < 2$
UC: C_2:$\delta - 11$, if $\delta < 2$
UC:$\max\{2\delta - 3, \delta - 12\}$, if $\delta < 2$

Figure 5: A propagation on the sample STNU from Fig. 4

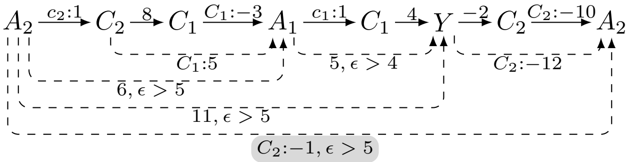


Figure 6: A negative loop from A_2 to A_2 that arises if $\epsilon > 5$

that results for any $\epsilon > 5$. (To ensure that all PLFs are non-decreasing, which simplifies our implementation, we let $\delta = -\epsilon$.) Finally, we observe a new result: since the only use of ϵ is to restrict the global domain, it follows that if the original network $\mathcal{S}$ contains only integer edge weights, then $\hat{\epsilon}(\mathcal{S})$ must also be an integer. As will be seen in the next section, this is a very different result from that for CSTNs.

Relaxing bounds on contingent links. Another previously open problem that our approach can solve is that of computing how much the bounds on a contingent link can be relaxed while preserving the DC property. For example, to determine how much the upper bound on $(A_1, 1, 3, C_1)$ can be relaxed, we first replace the labeled value on the upper-case edge from C_1 to A_1 by $C_1{:}\delta$, and then propagate using the rules from Table 1. Doing so yields the answer $\frac{11}{2}$. Similarly, if we want to determine how much the lower bound on that contingent link can be relaxed, we instead replace the labeled value on the lower-case edge from A_1 to C_1 by $c_1{:}\delta$, and then propagate.[6] For the sample STNU, the lower bound on A_1C_1 can be reduced to 0 without threatening the DC property.

Determining the tightest horizon. Our approach could also be used to compute the tightest *horizon* (equivalently, the shortest *makespan*) that can be applied to the network while preserving the DC property. In other words, if every time-point is constrained to occur before time h, as illustrated in Fig. 4c, then what is the smallest value of h that will preserve the DC property? However, for this particular problem, there is a more direct approach based on lower-bound values drawn from the *All-Pairs Shortest-Semi-Reducible-Paths* matrix (Morris and Muscettola 2005; Hunsberger 2015), as explicated by the following new result.

Theorem 1. *Let $\mathcal{S}$ be a DC STNU with time-points in $\mathcal{T}$; and let $\mathcal{D}^*$ be its All-Pairs Shortest-Semi-Reducible-Paths matrix. Then $h^* = \max\{-\mathcal{D}^*(X, \mathrm{Z}) \mid X \in \mathcal{T}\}$ is the tightest horizon for $\mathcal{S}$ that will preserve the DC property.*

Proof. Let H be the actual tightest horizon. Choose $X \in \mathcal{T}$ such that $-\mathcal{D}^*(X, \mathrm{Z}) = h^*$. Then in the situation where each contingent duration takes its maximum, $X \geq h^*$ must hold. Hence, $H \geq h^*$. Next, suppose $H > h^*$. Then inserting an edge of length h^* from Z to some Y must create a negative semi-reducible (SR) loop $\mathcal{L}$. If $\mathcal{L}_Y$ is the sub-path of $\mathcal{L}$ from Y to Z, then $|\mathcal{L}_Y| < -h^*$. Since $\mathcal{D}^*(Y, Z) \geq -h^*$, $\mathcal{L}_Y$ must not be SR. Let AC be the *first* lower-case edge in $\mathcal{L}_Y$ that cannot be reduced away in $\mathcal{L}_Y$. Since $\mathcal{L}$ *is* SR, AC must be reducible using a negative-length sub-path of $\mathcal{L}$ from C to Z to Y to some W. Since any suffix of an SR path is SR, the suffix of $\mathcal{L}_{CW}$ from Y to W, whose length is less than $-h^*$, is SR. But then appending the pre-existing zero-length edge from W to Z onto that suffix would create an SR path from Y to Z of length less than $-h^*$, contradicting $\mathcal{D}^*(Y, \mathrm{Z}) \geq -h^*$. Thus, $H = h^*$. □

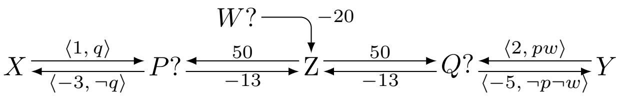


Figure 7: A sample CSTN graph

For the STNU in Fig. 3, the tightest horizon is $h^* = 12 = -\mathcal{D}^*(Y, Z)$, because C_2 might be 10, and $Y \geq C_2 + 2$. Interestingly, this computed value is much smaller than the theoretically determined upper bound, $h = M(n-1) = 50$, where M is the maximum absolute value of any negative edge in the *original* network and n is the number of time-points.[7]

Conditional Simple Temporal Networks

Tsamardinos et al. (2003) introduced Conditional Simple Temporal Networks (CSTNs) that extend STNs to include *observation time-points (OTPs)* and their associated *propositional letters*. In a CSTN, the execution of an OTP $P?$ generates a truth value for its associated proposition p. For example, $P?$ might represent the time at which a patient's blood pressure was measured, and $p = \mathit{true}$ might represent that the patient's blood pressure was higher than normal. Each constraint in a CSTN can be labeled by a conjunction of propositional literals that specify the scenarios in which that constraint must be satisfied.[8] (A *scenario* specifies a truth value for each propositional letter.) For example, the labeled constraint $(Y - X \leq 5, p\neg q)$ represents that $Y - X \leq 5$ must hold in scenarios where $p = \mathit{true}$ and $q = \mathit{false}$.

Definition 1 (CSTN). A *Conditional Simple Temporal Network* (CSTN) is a tuple, $\langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O}\rangle$, where: $\mathcal{T}$ is a finite set of real-valued variables (time-points); $\mathcal{P}$ is a finite set of propositional letters, and $\mathcal{P}^*$ is the set of *consistent* conjunctions of literals from $\mathcal{P}$; $\mathcal{C}$ is a set of *labeled* constraints, each having the form, $(Y - X \leq \delta,\ \alpha)$, where $X, Y \in \mathcal{T}$, $\delta \in \mathbb{R}$, and $\alpha \in \mathcal{P}^*$; $\mathcal{OT} \subseteq \mathcal{T}$ is a set of observation time-points (OTPs); and $\mathcal{O}\colon \mathcal{P} \to \mathcal{OT}$ is a bijection that associates a unique OTP $P?$ to each propositional letter p.

Each constraint, $(Y - X \leq \delta, \alpha)$, is represented by the *labeled value* $\langle \delta, \alpha\rangle$ on the edge from X to Y; and each edge may have many labeled values. Fig. 7 shows a sample CSTN.

An *execution strategy* for a CSTN specifies when time-points will be executed, but cannot affect which truth values

[6] Theorem 5 in Hunsberger (2015) ensures that, for the problem of relaxing the lower bound of (A, x, y, C), the LR$^+$ rule can use $\max\{f, 0\}$ instead of $\max\{f, -\delta\}$ when applied to upper-case edges labeled by C, thereby avoiding introducing *decreasing* PLFs.

[7] Cairo et al. (2017) proved the $M(n-1)$ bound for the horizon of a DC CSTN. However, since any STNU can be translated into a DC-equivalent CSTN (Hunsberger and Posenato 2018c), the $M(n-1)$ bound must also hold for a DC STNU.

[8] Time-points in CSTNs may also have propositional labels, but Cairo et al. (2017) showed that no loss of generality results from restricting attention to CSTNs whose time-points are unlabeled. Therefore, this paper does not consider labels on time-points.

LP:	$\begin{tikzcd} X \arrow[r, "\langle u, \alpha\rangle"] \arrow[rr, dashed, bend right, "\langle u+v, \alpha\beta\rangle"'] & Y \arrow[r, "\langle v, \beta\rangle"] & \mathrm{Z} \end{tikzcd}$	if $\alpha\beta \in \mathcal{P}^*, u+v<0$
qR_0:	$\begin{tikzcd} P? \arrow[r, "\langle w, \alpha\tilde{p}\rangle"] \arrow[r, dashed, shift right=2, "\langle w, \alpha\rangle"'] & \mathrm{Z} \end{tikzcd}$	if $w < 0;\ \alpha \in \mathcal{Q}^*$, $\tilde{p} \in \{p, \neg p, ?p\}$
qR_3^*:	$\begin{tikzcd} P? \arrow[r, "\langle w, \alpha\rangle"] & \mathrm{Z} & Y \arrow[l, "\langle v, \beta\tilde{p}\rangle"'] \arrow[l, dashed, shift left=2, "\langle m, \alpha\star\beta\rangle"] \end{tikzcd}$	if $w < 0, \alpha, \beta \in \mathcal{Q}^*$, $\tilde{p} \in \{p, \neg p, ?p\}$
In qR_0/qR_3^*, p does not appear in α or β. In qR_3^*, $m = \max\{v, w\}$.		

Table 3: Propagation rules for the π-DC-checking algorithm

qLP^+:	$\begin{tikzcd} X \arrow[r, "\langle u, \alpha\rangle"] \arrow[rr, dashed, bend right, "\langle u+v, \alpha\star\beta\rangle"'] & Y \arrow[r, "\langle v, \beta\rangle"] & T \end{tikzcd}$	Only if $u+v<0$ and $u<0$ or $\alpha\star\beta \in \mathcal{P}^*$
qR_0^+:	$\begin{tikzcd} P? \arrow[r, "\langle w, \alpha\tilde{p}\rangle"] \arrow[r, dashed, shift right=2, "\langle w, \alpha\rangle"'] & T \end{tikzcd}$	(same as for qR_0)
qR_3^+:	$\begin{tikzcd} P? \arrow[r, "\langle w, \alpha\rangle"] & T & Y \arrow[l, "\langle v, \beta\tilde{p}\rangle"'] \arrow[l, dashed, shift left=2, "\langle m, \alpha\star\beta\rangle"] \end{tikzcd}$	(same as for qR_3^*)
qInf:	$\begin{tikzcd} X \arrow[loop left, "\langle v, \alpha\rangle"] \arrow[loop right, dashed, "\langle -\infty, \alpha\rangle"] \end{tikzcd}$	Only if $v<0$; $\alpha \in \mathcal{Q}^* \backslash \mathcal{P}^*$
In qR_0^+/qR_3^+, p does not appear in α or β; and $m = \max\{v, w\}$. Generated labels $\langle v, \alpha\rangle$, where $v \geq 0$ and $\alpha \notin \mathcal{P}^*$, are discarded.		

Table 4: An alternative set of rules for π-DC checking

will be generated for propositional letters. However, a *dynamic* strategy can *react* to observations. A CSTN is *dynamically consistent* (DC) if it admits a dynamic execution strategy that guarantees the satisfaction of all relevant constraints no matter which outcomes are observed during execution.

Different versions of DC make different assumptions about how reactive a dynamic strategy can be. Cairo et al. (2016) defined π-DC, which assumes that a strategy can react instantaneously. Hunsberger and Posenato (2018a) presented a sound-and-complete π-DC-checking algorithm based on the propagation rules in Table 3, where Z is the zero time-point. Note that the expression $\alpha \star \beta$ in the qR_3^* rule can generate a new kind of label, called a *q-label.* Each q-label is a conjunction of *q-literals;* each q-literal has the form $p, \neg p$ or $?p$. A constraint labeled by $?p$ need only hold as long as the truth value of p is unknown (i.e., while $P?$ is unexecuted). The $\star$ operator extends conjunction to q-labels. Intuitively, if constraint C_1 is labeled by p, and C_2 by $\neg p$, then both C_1 and C_2 must hold as long as the value of p is unknown, represented by the q-label, $p{\star}\neg p = ?p$. The set of all q-labels is notated $\mathcal{Q}^* \supseteq \mathcal{P}^*$. For any $p \in \mathcal{P}$, $?p \models p$ and $?p \models \neg p$; and for any scenario s, $s \not\models ?p$.

The π-DC-checking algorithm is sound and complete, and guaranteed to terminate, but anecdotal evidence suggests that incorporating more rules can speed it up, particularly for CSTNs with *negative q-loops* (i.e., negative loops with mutually inconsistent edge-labels). For example, the loop from $P?$ to X to $P?$ in Fig. 7 is a negative q-loop. To avoid excessive propagations due to negative q-loops, Table 4 introduces a new set of propagation rules. Its first three rules are more general than their counterparts from Table 3 in that each can generate edges pointing at any $T \in \mathcal{T}$, not just Z. The qLP^+ rule is also more general because, when $u < 0$, it can generate q-labeled edges. Finally, the qInf rule avoids needless cycles of propagation by replacing the weight on a negative self-loop with $-\infty$ in cases where the label $\alpha \in \mathcal{Q}^* \backslash \mathcal{P}^*$. (If $\alpha \in \mathcal{P}^*$, then the network would necessarily be non-DC.) The labeled value $\langle -\infty, \alpha\rangle$ can then be propagated by the other rules—another way that they are more general.[9]

For numerical weights, qR_0^+ and qR_3^+ can be proven to be sound by replacing Z by any $T \in \mathcal{T}$ in the soundness proofs for qR_0 and qR_3^* (Hunsberger and Posenato 2018a).

Theorem 2. *The* qLP^+ *rule is sound for numerical weights.*

Proof. The case, $\alpha{\star}\beta = \alpha\beta \in \mathcal{P}^*$, is trivial. Hence, suppose that $u < 0$ and $\alpha \star \beta \in \mathcal{Q}^*$. Let σ be a dynamic and valid strategy that satisfies the pre-existing edges from X to Y, and Y to T, but *not* the generated edge from X to T.

Case 1: $v < 0$. In this case, u, v and $u + v$ are all negative. Therefore, the semantics for satisfying a q-labeled constraint applies (Hunsberger and Posenato 2018a). Hence, for *every* scenario s, conditions (1) and (2) below must hold, while for *some* scenario $\hat{s}$, condition (3) must hold.[10]

$$
\begin{array}{lr}
(1)\ \ (X_s \geq Y_s - u) & \text{(1a)}\\
\quad\ \ \textit{or}\ \ (\exists \tilde{a} \in \alpha \text{ such that } A? \prec_s X \text{ and } s \not\models \tilde{a}) & \text{(1b)}\\
(2)\ \ (Y_s \geq T_s - v) & \text{(2a)}\\
\quad\ \ \textit{or}\ \ (\exists \tilde{b} \in \beta \text{ such that } B? \prec_s Y \text{ and } s \not\models \tilde{b}) & \text{(2b)}\\
(3)\ \ (X_{\hat{s}} < T_{\hat{s}} - u - v) & \text{(3a)}\\
\quad\ \ \textit{and}\ \ (\forall \tilde{c} \in \alpha \star \beta\text{: } X \prec_{\hat{s}} C? \text{ or } \hat{s} \models \tilde{c}) & \text{(3b)}
\end{array}
$$

Henceforth, fix $s = \hat{s}$. Thus, (1), (2) and (3) hold for $s = \hat{s}$. Henceforth, the subscript s (or $\hat{s}$) shall be dropped.

If $a \in \alpha$, then a or $?a$ must be in $\alpha \star \beta$; if $\neg a \in \alpha$, then $\neg a$ or $?a$ must be in $\alpha \star \beta$; and if $?a \in \alpha$, then $?a \in \alpha \star \beta$. Since $s \not\models ?p$ for any $?p$, (1b) contradicts (3b). So (1a) holds.

Since (1a) and (2a) imply $X \geq Y - u \geq T - u - v$, which contradicts (3a), (2b) must hold. Let $\tilde{b} \in \beta$ such that $B? \prec Y$ and $s \not\models \tilde{b}$. Then $B? \prec Y \leq X + u < X$, whence $B? \prec X$. But $\tilde{b} \in \beta$ implies that either $\tilde{b}$ or $?b$ is in $\alpha \star \beta$, which contradicts (3b), since $s \not\models ?b$.

Case 2: $v \geq 0$. Here, $\beta \in \mathcal{P}^*$ and satisfying the constraint $(T - Y \leq v, \beta)$ implies that for every scenario s, either (2a) above holds, or (2b†) $s \not\models \beta$ holds. As in Case 1, for $s = \hat{s}$, (1a) must hold, but (2a) must *not* hold. Hence, (2b†) must hold. Then, (1a) yields: $Y < Y - u \leq X$. Next, for any $p \in \beta$ for which $s(p) = \textit{false}$ (resp., any $\neg p \in \beta$ for which $s(p) = \textit{true}$) (3b) implies that $X \prec P?$. Let s' be the same scenario as s, except that $s'(p) = \textit{true}$ for every $p \in \beta$ for which $s(p) = \textit{false}$, and $s'(p) = \textit{false}$ for every $\neg p \in \beta$ for which $s(p) = \textit{true}$. By construction, $s' \models \beta$ and $Y < X \prec_s P?$ for all such p; hence, $Y_{s'} = Y$ and $X_{s'} = X$. And since $s' \models \beta$, (2a) must hold for s', which, together with $Y' = Y$, $u + v < 0$ and (1a), yield: $T_{s'} \leq Y_{s'} + v = Y + v < Y - u \leq X \prec P?$. But then $T = T_{s'} \leq X$. And then (3a) and $u + v < 0$ yield the contradiction, $T \leq X < T - u - v < T$. □

[9] It is trivial for the implementation to accommodate $\langle -\infty, \alpha\rangle$.

[10] X_s denotes the value σ assigns to X in the scenario s; for any $p \in \mathcal{P}, \tilde{p} \in \{p, \neg p, ?p\}$; and $A? \prec_s X$ represents that $A?$ *preceded* X in the scenario s in the sense of the π-DC semantics (Cairo, Comin, and Rizzi 2016; Hunsberger and Posenato 2018a).

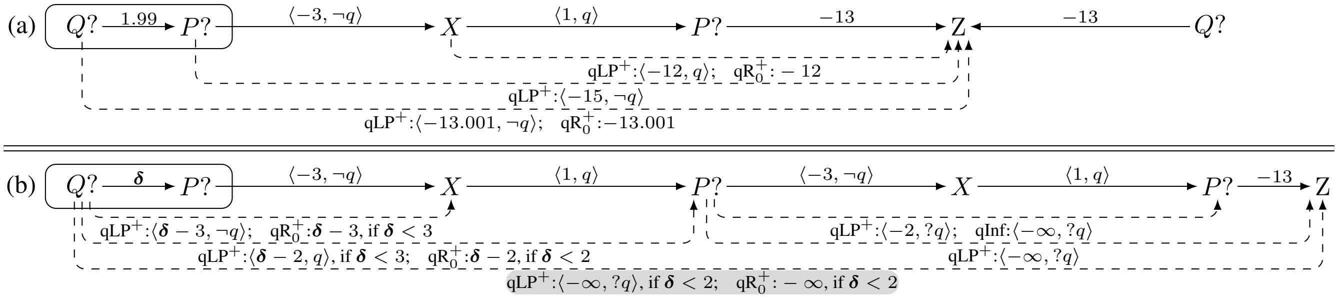

(a)
$Q?$
1.99
$P?$
$\langle -3, \neg q\rangle$
X
$\langle 1, q\rangle$
$P?$
-13
Z
-13
$Q?$
qLP$^+$:$\langle -12, q\rangle$; qR$_0^+$: -12
qLP$^+$:$\langle -15, \neg q\rangle$
qLP$^+$:$\langle -13.001, \neg q\rangle$; qR$_0^+$:$-13.001$
(b)
$Q?$
$\boldsymbol{\delta}$
$P?$
$\langle -3, \neg q\rangle$
X
$\langle 1, q\rangle$
$P?$
$\langle -3, \neg q\rangle$
X
$\langle 1, q\rangle$
$P?$
-13
Z
qLP$^+$:$\langle \boldsymbol{\delta} - 3, \neg q\rangle$; qR$_0^+$:$\boldsymbol{\delta} - 3$, if $\boldsymbol{\delta} < 3$
qLP$^+$:$\langle -2, ?q\rangle$; qInf:$\langle -\infty, ?q\rangle$
qLP$^+$:$\langle \boldsymbol{\delta} - 2, q\rangle$, if $\boldsymbol{\delta} < 3$; qR$_0^+$:$\boldsymbol{\delta} - 2$, if $\boldsymbol{\delta} < 2$
qLP$^+$:$\langle -\infty, ?q\rangle$
qLP$^+$:$\langle -\infty, ?q\rangle$, if $\boldsymbol{\delta} < 2$; qR$_0^+$: $-\infty$, if $\boldsymbol{\delta} < 2$

Figure 8: Propagations for the CSTN from Fig. 7 starting with an edge from $Q?$ to $P?$ labeled by: (a) 1.99; and (b) δ

To prove the soundness of rules that manipulate labeled values such as $\langle -\infty, \alpha\rangle$ requires the following definition.

Definition 2. *σ satisfies* the constraint, $(T - X \leq -\infty, \alpha)$, if for each scenario s: $(\exists \tilde{a} \in \alpha)$ s.t. $A? \prec_s X$ and $s \not\models \tilde{a}$.

Such a constraint is necessarily unsatisfiable if $\alpha \in \mathcal{P}^*$. Note that the satisfaction conditions depend only on X and α.

Theorem 3. *The* qInf *rule is sound.*

Proof. Suppose σ satisfies the pre-existing loop in the qInf rule (i.e., $(X - X \leq v, \alpha)$, where $v < 0$). Then, for each scenario s, either $X - X \leq v$ or $(\exists \tilde{a} \in \alpha)$ such that $A? \prec_s X$ and $s \not\models \tilde{a}$. Since $0 = X - X \leq v < 0$ is unsatisfiable, the existential clause must hold. □

Theorem 4. *The* qLP$^+$ *rule is sound when* $-\infty \in \{u, v\}$.

Proof. Case 1: $u = -\infty$. Suppose σ satisfies the constraint, $(Y - X \leq -\infty, \alpha)$. Then for each scenario s, $(\exists \tilde{a} \in \alpha)$ such that $A? \prec_s X$ and $s \not\models \tilde{a}$. Then $\tilde{a} \in \alpha \star \beta$ or $?a \in \alpha \star \beta$ (even if $\tilde{a} = ?a$). Since $s \not\models \tilde{a}$ and $s \not\models ?a$, it follows that σ satisfies the generated constraint, $(T - X \leq -\infty, \alpha \star \beta)$.

Case 2: $u < 0$ *and* $v = -\infty$. Suppose σ satisfies $(Y - X \leq u, \alpha)$ and $(T - Y \leq -\infty, \beta)$, but *not* the generated edge $(T - X \leq -\infty, \alpha \star \beta)$. Then, for *all* scenarios s, (1) and (2) below hold, while for *some* $\hat{s}$, (3) below holds.

(1) $X \geq Y - u$ (1a)
or $(\exists \tilde{a} \in \alpha$ such that $A? \prec_s X$ and $s \not\models \tilde{a})$ (1b)

(2) $(\exists \tilde{b} \in \beta$ such that $B? \prec_s Y$ and $s \not\models \tilde{b})$

(3) $(\forall \tilde{c} \in \alpha \star \beta,\ X \prec_{\hat{s}} C?$ or $s \models \tilde{c})$.

Henceforth, let $s = \hat{s}$. Let $\tilde{b}$ be such that (2) holds. Then $\tilde{b} \in \alpha \star \beta$ or $?b \in \alpha \star \beta$ (even if $\tilde{b} = ?b$), $s \not\models \tilde{b}$, $s \not\models ?b$, and $B? \prec_s Y$. Then (3) implies $X \prec_s B?$. Thus, $X \prec_s B \prec_s Y$. Hence, (1a) cannot hold, since $X \geq Y - u > Y$ contradicts $X \prec_s Y$. Thus, (1b) holds, whence $\tilde{a} \in \alpha \star \beta$ or $?a \in \alpha \star \beta$, $s \not\models \tilde{a}$, $s \not\models ?a$, and $A? \prec_s X$ together contradict (3). □

Soundness proofs for qR$_0^+$ and qR$_3^+$ involving $-\infty$ are similar, but space limitations preclude giving them here.

Storing labeled values. Since each CSTN edge could have up to $4^{|\mathcal{P}|}$ different labeled values, it is important to store them efficiently. With numerical weights, labeled values can be stored in a *subsumption hierarchy,* where $\langle v, \alpha\rangle$ subsumes (i.e., is more general than) $\langle w, \beta\rangle$ iff $\beta \models \alpha$ (i.e., the literals in α are a subset of those in β) and $w < v$. For example, $\langle 3, p\neg qr\rangle$ subsumes $\langle 2, p\neg q(?r)s\rangle$. (Recall that $?r \models r$.) The root of the hierarchy is a labeled value $\langle w, \boxdot\rangle$, where $w \in (-\infty, \infty]$. The insertion point for a new labeled value is found by descending from the root node, looking for *most specific subsumers,* then updating parent/child links as needed.

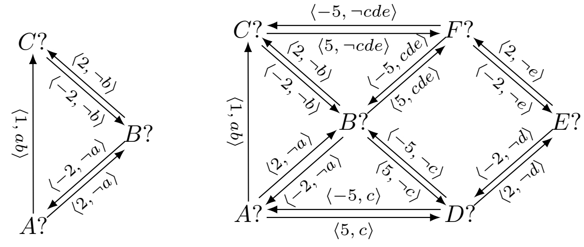
For PLFs f and g, it may happen that $g(\delta) < f(\delta)$ for some, but not all, values of δ. Therefore, for PLFs, we say that $\langle f, \alpha\rangle$ subsumes $\langle g, \beta\rangle$ if $\beta \models \alpha$ and $g(\delta) < f(\delta)$ for at least some values of δ. Therefore, our implementation for CSTNs defines a predicate, $\mathit{ltSomewhere?}$, that returns true if its first input is less than its second input on some non-empty portion of the domain.

How much can a given edge weight be tightened? For the CSTN in Fig. 7, the strongest weight that can be put onto a new edge from $P?$ to $Q?$ can be computed by inserting an edge from $P?$ to $Q?$ with a weight of δ, and then propagating. The answer is -37, because a more negative weight would create a negative loop from $P?$ to $Q?$ to Z to $P?$.

In contrast, the strongest weight that can be put onto a new edge from $Q?$ to $P?$ is 2. However, for this version of the problem, the negative q-loop between X and $P?$ causes the three-rule algorithm to cycle indefinitely, its lower bound for δ approaching, but never reaching, the limit, 2.[11] To see why, consider the simpler case where the edge from $Q?$ to $P?$ has the numerical weight, 1.99. As shown in the top of Fig. 8, it takes nine rule applications to decrease the weight on the edge from $Q?$ to Z from its initial value of -13 to -13.01. Hence, 900 rule applications would be required to decrease the weight on that edge from -13 to -14. Similarly, if the initial weight on the edge from $Q?$ to $P?$ were 1.9999, it would take $90,000$ rule applications to achieve the same result. When using PLFs involving values of δ that can be arbitrarily close to 2, the 3-rule algorithm will cycle indefinitely. To avoid the non-termination problem, the four rules from Table 4 can be used instead. They find the answer in nine rule applications, as shown in the bottom half of Fig. 8.

What is the maximum delay after Z? Consider the CSTN in Fig. 7. Suppose that Z has been executed at 0. The maximum delay before some other time-point must be executed can be found by inserting edges from each time-point to Z

[11] The π-DC-checking algorithm is guaranteed to terminate for CSTNs with numerical weights (Hunsberger and Posenato 2018b); however, with PLF weights, it can fail to terminate. In contrast, the rule-set in Table 4 is guaranteed terminate even for PLF weights.


C?
B?
A?
$\langle 1, ab\rangle$
$\langle 2, \neg b\rangle$
$\langle -2, \neg b\rangle$
$\langle -2, \neg a\rangle$
$\langle 2, \neg a\rangle$
C?
F?
B?
E?
A?
D?
$\langle -5, \neg cde\rangle$
$\langle 5, \neg cde\rangle$
$\langle 2, \neg b\rangle$
$\langle -2, \neg b\rangle$
$\langle -5, cde\rangle$
$\langle 5, cde\rangle$
$\langle 2, \neg e\rangle$
$\langle -2, \neg e\rangle$
$\langle 1, ab\rangle$
$\langle 2, \neg a\rangle$
$\langle -2, \neg a\rangle$
$\langle -5, \neg c\rangle$
$\langle 5, \neg c\rangle$
$\langle -2, \neg d\rangle$
$\langle 2, \neg d\rangle$
$\langle -5, c\rangle$
$\langle 5, c\rangle$

(a) A CSTN, $\hat{\epsilon} = \frac{1}{2}$

(b) A more complex CSTN, $\hat{\epsilon} = \frac{1}{4}$

Figure 9: Sample CSTN graphs ($A? \equiv Z$)

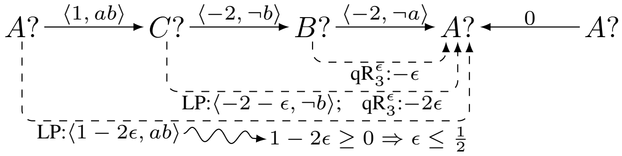

A?
$\langle 1, ab\rangle$
C?
$\langle -2, \neg b\rangle$
B?
$\langle -2, \neg a\rangle$
A?
0
A?
qR_3^ϵ:$-\epsilon$
LP:$\langle -2-\epsilon, \neg b\rangle$; qR_3^ϵ:-2ϵ
LP:$\langle 1-2\epsilon, ab\rangle$
$1-2\epsilon \geq 0 \Rightarrow \epsilon \leq \frac{1}{2}$

Figure 10: An upper-bound for ϵ for the CSTN in Fig. 9a

with weight δ, as shown in Fig. 4b. After ten rounds of propagation, involving a complex interplay among the LP, qR_0 and qR_3^* rules, it is discovered that $\delta \geq -43$ (i.e., the maximum delay before another time-point must be executed is 43).

CSTNs with Bounded Reaction Time. For any CSTN $\mathcal{S}$, Comin and Rizzi (2015) defined $\hat{\epsilon}(\mathcal{S})$ to be the maximum value of ϵ for which $\mathcal{S}$ is ϵ-DC (i.e., for any $\epsilon \in [0, \hat{\epsilon}(\mathcal{S})]$, $\mathcal{S}$ is ϵ-DC). They proved that for a DC CSTN with integer weights, $\hat{\epsilon}(\mathcal{S})$ cannot be smaller than $\frac{1}{n2^k}$ (i.e., the agent need not be infinitesimally reactive), where n is the number of time-points and k is the number of observation time-points. They also showed that the bound is "almost tight" by presenting CSTNs for which $\hat{\epsilon}(\mathcal{S}) = \frac{1}{2^{k/3}}$. However, the actual value of $\hat{\epsilon}(\mathcal{S})$ is typically much larger than this worst-case bound.

The ϵ-DC-checking algorithm due to Hunsberger and Posenato (2018a) uses three rules that are identical to the rules in Table 3 except that the qR_3^* rule can be applied whenever $w < \epsilon$, and the value of m is given by $\max\{v, w - \epsilon\} = \max\{v, w + \delta\}$, where $\delta = -\epsilon$. (The resulting rule is called qR_3^ϵ.) As with the ϵ-DC-checking algorithm for STNUs, it suffices to implement a function that can compute this maximum where v and w can be any combination of numbers, $+\infty$ or PLFs. As with STNUs, there is no need to insert PLF edges into the initial CSTN; instead, occurrences of ϵ are inserted automatically when the qR_3^ϵ rule is applied. Applying the PLF-version of the ϵ-DC-checking algorithm to the CSTN from Fig. 7 computes $\hat{\epsilon}(\mathcal{S})$ to be 12.5, which is *much greater* than the theoretically determined bound of $\frac{1}{n2^k} = \frac{1}{6 \cdot 2^3} = \frac{1}{48}$.

The CSTNs in Fig. 9, all of whose time-points are observation time-points (including Z $\equiv A$), are adapted from Cairo and Rizzi (2016). For the CSTN in Fig. 9a, the ϵ-DC rules correctly determine that $\hat{\epsilon} = \frac{1}{2}$. The key propagation sequence is shown in Fig. 10. Since the final generated edge is a self-loop, its weight must be non-negative (i.e., $1 - 2\epsilon \geq 0$), which leads to the restriction, $\epsilon \leq \frac{1}{2}$. For the CSTN in Fig. 9b the ϵ-DC rules compute $\hat{\epsilon} = \frac{1}{4}$. These examples illustrate that CSTNs with integer weights can have a fractional value of $\hat{\epsilon}$.

Computing the tightest horizon. As with STNUs, our approach could also be used to compute the tightest horizon that could be imposed on a DC CSTN—namely, by inserting upper-bound constraints for all time-points, as shown in Fig. 4c, and then propagating. However, the following new result offers a more direct approach. It is based on computing for each time-point X its *effective lower bound,* $ELB(X)$ (Hunsberger and Posenato 2018b). This value equals the absolute value of the length of the most negative edge from X to Z in the *fully propagated* network and, hence, represents the greatest lower bound for X implied by the labeled constraints in the original network.

Theorem 5. *Let $\mathcal{S}$ be a DC CSTN, and $\mathcal{T}$ its set of time-points. Then $h^* = \max\{ELB(X) \mid X \in \mathcal{T}\}$ is the tightest horizon that can be imposed on $\mathcal{S}$ preserving the DC property.*

Proof. Let X be any time-point in $\mathcal{T}$ for which $ELB(X) = h^*$. Then there exists an edge from X to Z in the *fully propagated* network, with length h^* and labeled by some $\alpha \in \mathcal{P}^*$. Therefore, in any scenario where α is true, $X \geq h^*$ must hold. Hence, the tightest horizon cannot be smaller than h^*. On the other hand, the *earliest-execution strategy,* which executes each time-point X at or before its ELB value, is guaranteed to be valid and dynamic (Hunsberger and Posenato 2018b). Therefore, the tightest horizon cannot be larger than h^*. □

For the CSTN in Fig. 7, the computed tightest horizon is 25, which is not at all obvious from the structure of the graph. It derives from the complex interplay among the propagation rules. Note that this value is much smaller than the theoretical bound $M(n-1) = 20 \cdot 5 = 100$ due to Cairo et al. (2017), where M is the maximum absolute value of any negative edge in the *original* network, and n is the number of time-points.

Conclusions and Related Work

This paper presented a novel technique of using piecewise-linear functions as weights on STNU and CSTN edges to solve previously open problems. Generalizing propagation rules to accommodate PLF weights required implementing a small handful of functions. Most problems were solved by inserting one or more edges with an initial weight of δ, and then using the modified rules to propagate PLF weights. For ϵ-DC checking, no new edges were added; instead, the modified rules automatically introduced PLF weights, which could then be propagated. In the case of CSTNs with negative q-loops, a new set of propagation rules was introduced to avoid needless cycles of propagation caused by the PLFs. The paper also proved two new results for computing the tightest horizon for a DC STNU or CSTN.

Yu et al. (2017) address *tightening* contingent links in over-constrained temporal problems with choice nodes (CCTPU) and contingent links with probabilistic durations (cc-pCCTP). Their *Best-First Conflict-Directed Relaxation* (BCDR) algorithm computes continuous relaxations for over-constrained temporal problems having different types of constraints.

Cui et al. (2015) address the problem of finding the bounds on the ordinary edges in a DC STNU that optimize a given objective function while preserving the DC property. They formulate the DC constraints as a disjunctive linear model having up to $O(n^3)$ constraints and $O(kn^2)$ variables. Although their approach could be used to solve some of the STNU problems presented in this paper, it is not clear whether their technique of reducing the number of variables in their model would yield an advantage given that "any link with bounds constrained in the input STNU must be represented". In contrast, our approach uses only one variable, δ. In addition, their approach has not been applied to CSTNs.

Currently, we are working on techniques for speeding up our algorithm based on lazy computation of PLFs.

References

Bhargava, N.; Muise, C.; Vaquero, T.; and Williams, B. 2018. Delay controllability: Multi-agent coordination under communication delay. Technical Report MIT-CSAIL-TR-2018-02, MIT.

Cairo, M., and Rizzi, R. 2016. Dynamic controllability of conditional simple temporal networks is pspace-complete. In *23rd Int. Symp. on Temporal Representation and Reasoning (TIME-2016)*, 90–99.

Cairo, M.; Hunsberger, L.; Posenato, R.; and Rizzi, R. 2017. A Streamlined Model of Conditional Simple Temporal Networks - Semantics and Equivalence Results. In *24th Int. Symp. on Temporal Representation and Reasoning (TIME 2017)*, volume 90 of *LIPIcs*, 10:1–10:19.

Cairo, M.; Comin, C.; and Rizzi, R. 2016. Instantaneous reaction-time in dynamic-consistency checking of conditional simple temporal networks. In *23rd Int. Symp. on Temporal Representation and Reasoning (TIME 2016)*, 80–89.

Cairo, M.; Hunsberger, L.; and Rizzi, R. 2018. Faster dynamic controllablity checking for simple temporal networks with uncertainty. In *25th Int. Symp. on Temporal Representation and Reasoning (TIME 2018)*, volume 120 of *LIPIcs*, 8:1–8:16.

Casanova, G.; Pralet, C.; and Lesire, C. 2015. Managing dynamic multi-agent simple temporal network. In *2015 Int. Conf. on Autonomous Agents and Multiagent Systems, AAMAS 2015*, 1171–1179. ACM.

Comin, C., and Rizzi, R. 2015. Dynamic consistency of conditional simple temporal networks via mean payoff games: a singly-exponential time dc-checking. In *22st Int. Symp. on Temporal Representation and Reasoning (TIME 2015)*, 19–28.

Cui, J.; Yu, P.; Fang, C.; Haslum, P.; and Williams, B. C. 2015. Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *25th Int. Conf. on Automated Planning and Scheduling (ICAPS-2015)*.

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

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predictable exogenous events. *J. Artif. Intell. Res.* 25:187–231.

Hunsberger, L., and Posenato, R. 2018a. Reducing ϵ-DC Checking for Conditional Simple Temporal Networks to DC Checking. In *25th Int. Symp. on Temporal Representation and Reasoning (TIME 2018)*, volume 120 of *LIPIcs*, 15:1–15:15.

Hunsberger, L., and Posenato, R. 2018b. Simpler and faster algorithm for checking the dynamic consistency of conditional simple temporal networks. In *26th Int. Joint Conf. on Artificial Intelligence, (IJCAI-2018)*, 1324–1330.

Hunsberger, L., and Posenato, R. 2018c. Sound-and-Complete Algorithms for Checking the Dynamic Controllability of Conditional Simple Temporal Networks with Uncertainty. In *25th Int. Symp. on Temporal Representation and Reasoning (TIME 2018)*, volume 120 of *LIPIcs*, 14:1–14:17.

Hunsberger, L. 2013. Magic Loops in Simple Temporal Networks with Uncertainty–Exploiting Structure to Speed Up Dynamic Controllability Checking. In *5th Int. Conf. on Agents and Artificial Intelligence, ICAART 2013*, volume 2, 157–170.

Hunsberger, L. 2015. Efficient execution of dynamically controllable simple temporal networks with uncertainty. *Acta Informatica* 53(2):89–147.

Kumar, T. K. S. 2004. A polynomial-time algorithm for simple temporal problems with piecewise constant domain preference functions. In *19th National Conf. on Artificial Intelligence (AAAI-04)*, 67–72.

Morris, P. H., and Muscettola, N. 2005. Temporal dynamic controllability revisited. In *20th National Conf. on Artificial Intelligence (AAAI-05)*, 1193–1198.

Morris, R.; Morris, P.; Khatib, L.; and Yorke-Smith, N. 2005. Temporal constraint reasoning with preferences and probabilities. In *IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling*, 150–155.

Morris, P. H.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI-01)*, 494–502.

Morris, P. 2006. A Structural Characterization of Temporal Dynamic Controllability. In *Principles and Practice of Constraint Programming (CP 2006)*, volume 4204, 375–389.

Morris, P. 2014. Dynamic controllability and dispatchability relationships. In *Integration of AI and OR Techniques in Constraint Programming*, volume 8451 of *LNCS*. 464–479.

Rossi, F.; Venable, K.; and Yorke-Smith, N. 2006. Uncertainty in soft temporal constraint problems: a general framework and controllability algorithms for the fuzzy case. *Journal of Artificial Intelligence Research* 27:617–674.

Tsamardinos, I.; Vidal, T.; and Pollack, M. E. 2003. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints* 8:365–388.

Yu, P.; Williams, B.; Fang, C.; Cui, J.; and Haslum, P. 2017. Resolving over-constrained temporal problems with uncertainty through conflict-directed relaxation. *Journal of Artificial Intelligence Research* 60:425–490.