Converting Simple Temporal Networks with Uncertainty into Minimal Equivalent Dispatchable Form

Luke Hunsberger[1], **Roberto Posenato**[2]
[1]Vassar College, Poughkeepsie, NY 12604 USA
[2]Università di Verona, Verona, Italy
hunsberger@vassar.edu, roberto.posenato@univr.it

Abstract

A *Simple Temporal Network with Uncertainty* (STNU) is a structure for representing and reasoning about time constraints on actions with uncertain durations. An STNU is *dynamically controllable* (DC) if there exists a dynamic strategy for executing the network that guarantees that all of its constraints will be satisfied no matter how the uncertain durations turn out—within their specified bounds. However, such strategies typically require exponential space. Therefore, it is essential to convert a DC STNU into a so-called *dispatchable* form for practical applications. For a dispatchable STNU, the relevant portions of a real-time execution strategy can be incrementally constructed during execution, requiring only $O(n^2)$ space while also providing maximum flexibility but requiring only minimal computation during execution. Existing algorithms can generate equivalent dispatchable STNUs, but with no guarantee about the number of edges in the output STNU. Since that number directly impacts the computations during execution, this paper presents a novel algorithm for converting any dispatchable STNU into an equivalent dispatchable network with minimal edges. The complexity of the algorithm is $O(kn^3)$, where k is the number of actions with uncertain durations, and n is the number of timepoints. The paper also provides an empirical evaluation of the order-of-magnitude reduction of edges obtained by the new algorithm.

Introduction and Related Work

Temporal constraint networks facilitate representation and reasoning about temporal constraints on activities. Among the many kinds of temporal constraint networks in the literature, *Simple Temporal Networks with Uncertainty* (STNUs) are one of the most important because they allow the explicit representation of actions with uncertain durations (Morris, Muscettola, and Vidal 2001). In an STNU, an action with uncertain duration is represented by a *contingent link* that specifies bounds on the duration's uncertainty. A contingent link also represents that a scheduler cannot *decide* the action's duration but can only *observe* it at runtime. Therefore, for STNUs, it is important to know whether a strategy exists for executing the network that guarantees that all of its constraints will be satisfied no matter how the uncertain durations play out. If such a strategy exists, the STNU is said to be *dynamically controllable* (DC). The literature includes several polynomial-time algorithms for checking the DC property that differ in their approaches to finding possible inconsistencies, characterized by different kinds of negative cycles (Stedl and Williams 2005; Morris 2006, 2014; Nilsson, Kvarnstrom, and Doherty 2014; Cairo, Hunsberger, and Rizzi 2018). However, the algorithms are not constructive (i.e., they do not output actual execution strategies in positive instances).

For DC STNUs, an execution strategy can be specified by pre-computing all possible incremental schedules for all combinations of uncertain action durations. However, such a strategy requires exponential space and, therefore, is impractical. So, for practical applications, Morris (2014, 2016) proposed converting DC STNUs into an equivalent *dispatchable* form. Hunsberger and Posenato (2024) showed that for a dispatchable STNU, a dynamic execution strategy can be generated incrementally during execution, using only $O(n^2)$ space, where n is the number of timepoints. Moreover, such a strategy provides maximum flexibility for the action scheduler but requires minimal real-time computation.

Most DC-checking algorithms do not guarantee a dispatchable output, but Morris (2014) indicated that his DC-checking algorithm could be modified to do so. We call that version of his algorithm *Morris14*. More recently, Hunsberger and Posenato (2023) presented an algorithm for creating an equivalent dispatchable STNU called $\mathit{FD}_{\text{STNU}}$. Although these are the only known algorithms that guarantee an equivalent dispatchable output, they do not make any claims about the *number of edges* in the output. Since the number of edges directly impacts computations during execution, this paper presents a novel algorithm for converting any dispatchable STNU into an equivalent dispatchable network with a *minimal* number of edges. The complexity of the algorithm is $O(kn^3)$, where k is the number of actions with uncertain durations, and n is the number of nodes in the network. The paper provides an empirical evaluation of the *order-of-magnitude* reduction in the number of edges obtained by the new algorithm.

Background

This section summarizes relevant definitions and results for the dispatchability of STNs and STNUs.

Definition 1 (STN)**.** A *Simple Temporal Network* (STN) is a pair $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set of real-valued variables called *timepoints* (TPs) and $\mathcal{C}$ is a set of constraints of the form $(Y - X \leq \delta)$, for some $X, Y \in \mathcal{T}$

and $\delta \in \mathbb{R}$ (Dechter, Meiri, and Pearl 1991). Each STN $(\mathcal{T}, \mathcal{C})$ has a corresponding graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, where the TPs in $\mathcal{T}$ serve as the graph's nodes, and the constraints in $\mathcal{C}$ correspond to labeled, directed edges in $\mathcal{E}$. In particular: $\mathcal{E} = \{X \xrightarrow{\delta} Y \mid (Y - X \leq \delta) \in \mathcal{C}\}$. For convenience, an edge $X \xrightarrow{\delta} Y$ may be notated as (X, δ, Y).

An STN is *consistent* if it has a solution as a constraint satisfaction problem. Although solutions for consistent STNs can be computed in advance of execution (e.g., by the *Bellman-Ford* algorithm (Cormen et al. 2022)), it is often desirable to preserve as much flexibility as possible during execution (e.g., to enable reacting to unanticipated events). Toward that end, Tsamardinos, Muscettola, and Morris (1998) first specified a real-time execution algorithm for STNs, called the *Time Dispatching* (TD) algorithm, and then defined an STN to be *dispatchable* if every run of the TD algorithm was guaranteed to generate a solution. The TD algorithm provides maximum flexibility during execution by maintaining *time windows* for each timepoint. It requires minimal computation during execution by propagating the effects of each real-time execution, $X = v$, only *locally* (i.e., to X's *neighbors*; that is, timepoints connected to X by a single edge). Morris (2016) subsequently provided a graphical characterization of STN dispatchability in terms of *vee-paths.*

Definition 2. A *vee-path* is a path consisting of zero or more negative edges followed by zero or more non-negative edges. If $\mathcal{P}$ is a vee-path from X to Y that is also a shortest path from X to Y, then $\mathcal{P}$ is called a *shortest vee-path* (SVP).

Theorem 1. *An STN is dispatchable if and only if whenever there is a path from any X to any Y, there is an SVP from X to Y (Morris 2016).*

A *Simple Temporal Network with Uncertainty* (STNU) augments an STN to include actions with uncertain durations.

Definition 3 (STNU). An STNU is a triple, $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, where $(\mathcal{T}, \mathcal{C})$ is an STN, and $\mathcal{L}$ is a set of *contingent links* (CLs), each of the form (A, x, y, C), where $A, C \in \mathcal{T}$ and $0 < x < y < \infty$ (Morris, Muscettola, and Vidal 2001).

Once the *activation timepoint* A is *executed* (i.e., assigned a value during execution), the *contingent timepoint* C is guaranteed to execute such that $C - A \in [x, y]$, but the particular time chosen for C is not controlled by the agent executing the network; it is only *observed* in real-time.

With no loss of generality, we assume that no contingent TP can serve as the activation TP for another contingent link.

Each STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ has a corresponding graph, $\mathcal{G} = (\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}_{\text{lc}} \cup \mathcal{E}_{\text{uc}})$, where $(\mathcal{T}, \mathcal{E}_{\text{o}})$ is the graph for the STN $(\mathcal{T}, \mathcal{C})$, and $\mathcal{E}_{\text{lc}}$ and $\mathcal{E}_{\text{uc}}$ contain labeled, directed edges derived from the contingent links in $\mathcal{L}$. In particular: $\mathcal{E}_{\text{lc}} = \{A \xrightarrow{c:x} C \mid (A, x, y, C) \in \mathcal{L}\}$, and $\mathcal{E}_{\text{uc}} = \{C \xrightarrow{C:-y} A \mid (A, x, y, C) \in \mathcal{L}\}$. The *lower-case* (LC) edge $A \xrightarrow{c:x} C$ represents the uncontrollable possibility that the duration $C - A$ might take on its minimum value x, while the *upper-case* (UC) edge $C \xrightarrow{C:-y} A$ represents the uncontrollable possibility that $C - A$ might take on its maximum value y. These *labeled* edges may be respectively notated as $(A, c{:}x, C)$ and $(C, C{:}-y, A)$. In contrast, the constraints in $\mathcal{C}$ and edges in $\mathcal{E}_{\text{o}}$ may be called *ordinary* constraints and edges, respectively, to distinguish them from the labeled LC and UC edges. For convenience, we frequently blur the distinction between an STNU and its graph, and between edges and constraints.

Definition 4. An STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is *dynamically controllable* (DC) if there exists a *dynamic strategy* for executing its timepoints such that all constraints in $\mathcal{C}$ are guaranteed to be satisfied no matter how the durations of the CLs in $\mathcal{L}$ turn out (Morris, Muscettola, and Vidal 2001). The strategy is *dynamic* in that its execution decisions cannot depend on advance knowledge of future contingent executions.

Several polynomial-time *DC-checking* algorithms have been presented in the literature (Morris 2006, 2014; Cairo, Hunsberger, and Rizzi 2018; Hunsberger and Posenato 2022). However, in positive instances, such algorithms only confirm the *existence* of a dynamic execution strategy; they do not output one. Since such strategies typically require exponential space, Morris (2016) extended the concept of *dispatchability* from STNs to *extended* STNUs (ESTNUs), as follows.

First, we must backtrack. Some DC-checking algorithms for STNUs generate a new kind of *conditional* constraint called a *wait* (Morris 2006). A typical wait can be glossed as *"If the contingent timepoint C has not yet executed, then V must wait until at least w after the activation timepoint A."* Its graphical representation is the *generated* UC edge, $V \xrightarrow{C:-v} A$, where A and C are the activation and contingent timepoints for some CL (A, x, y, C). Intuitively, since the execution of C cannot be directly controlled and might occur as late as y after A, V must wait until v after A; but if C executes earlier than v after A, then the wait is automatically satisfied, and V may be executed immediately.

Generating wait edges is not required to determine the DC property (e.g., as seen in the algorithms of Morris (2014) and Cairo, Hunsberger, and Rizzi (2018)), but waits turn out to be necessary for enforcing the dispatchability of STNUs. Anticipating this, Morris (2016) defined *extended* STNUs and the dispatchability of ESTNUs, as follows.

Definition 5. An *extended STNU* (ESTNU) is an STNU augmented to include zero or more waits (equivalently, an STNU graph together with a set $\mathcal{E}_{\text{ucg}}$ of *generated* UC edges).

Definition 6 (Situation). Let $\mathcal{S}$ be an ESTNU with k contingent links whose duration ranges are $[x_1, y_1], \ldots, [x_k, y_k]$. A *situation* for $\mathcal{S}$ is a k-tuple $\omega = (\omega_1, \omega_2, \ldots, \omega_k)$ where $\omega_i \in [x_i, y_i]$ for each $i \in \{1, 2, \ldots, k\}$. The space of all situations is denoted by $\Omega = [x_1, y_1] \times \cdots \times [x_k, y_k]$. If C is the contingent timepoint for a link (A, x, y, C), then the duration $C - A$ in the situation ω may alternatively be notated as ω_c.

A situation not only specifies the duration of each contingent link, it also determines the impact of each wait. This information is captured by the *projection* of the ESTNU.

Definition 7. For an ESTNU graph $\mathcal{G} = (\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}_{\text{lc}} \cup \mathcal{E}_{\text{uc}} \cup \mathcal{E}_{\text{ucg}})$, and a situation ω, the *projection* of $\mathcal{G}$ onto ω is the STN graph $\mathcal{G}_\omega = (\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}^\omega_{\text{lc}} \cup \mathcal{E}^\omega_{\text{uc}} \cup \mathcal{E}^\omega_{\text{ucg}})$, where:

$$\begin{array}{ll} \mathcal{E}^\omega_{\text{lc}} = \{(A_i, \omega_i, C_i) & \mid \exists (A_i, c_i{:}x_i, C_i) \in \mathcal{E}_{\text{lc}}\} \\ \mathcal{E}^\omega_{\text{uc}} = \{(C_i, -\omega_i, A_i) & \mid \exists (C_i, C_i{:}-y_i, A_i) \in \mathcal{E}_{\text{uc}}\} \\ \mathcal{E}^\omega_{\text{ucg}} = \{(V, \delta_i, A_i) & \mid \exists (V, C_i{:}-v, A_i) \in \mathcal{E}_{\text{ucg}}\} \end{array}$$
$$(\delta_i \text{ abbreviates } \max\{-\omega_i, -v\})$$

The edges in $\mathcal{E}^{\omega}_{\mathrm{lc}}$ and $\mathcal{E}^{\omega}_{\mathrm{uc}}$ fix each duration $C_i - A_i = \omega_i$. The edges in $\mathcal{E}^{\omega}_{\mathrm{ucg}}$ reflect the *effective* constraints imposed by the waits in $\mathcal{E}_{\mathrm{ucg}}$ in the situation ω. For example, if $\omega_i = 3$, then the wait $(V, C_i{:}{-7}, A_i)$ projects onto $(V, -3, A_i)$, reflecting that the wait vanishes when C_i executes at $A_i + 3$. (Note: $\max\{-3, -7\} = -3$.) But if $\omega_i = 8$, then the wait projects onto $(V, -7, A_i)$, reflecting that V must wait the full 7 units, since C_i did not execute early. (Note: $\max\{-8, -7\} = -7$.)

Definition 8 (Path length)**.** The length of a path $\mathcal{P}$ in a projection $\mathcal{G}_\omega$ is notated as $|\mathcal{P}|_\omega$. For any V and W, $\mathrm{d}_\omega(V, W)$ denotes the length of the shortest path from V to W in $\mathcal{G}_\omega$; and $d^*(V, W) = \max_\omega\{\mathrm{d}_\omega(V, W)\}$ denotes the maximum such length *across all projections.* If $\mathcal{P}$ has only ordinary edges, then $|\mathcal{P}|$ denotes its length; and $\mathrm{d}(V, W)$ denotes the length of the shortest path from V to W that has only *ordinary* edges. If context allows, we may notate a path by listing its timepoints. For example, a path from V to A to C may be notated as VAC, and its length as $|VAC|_\omega$.

Following Morris (2014), Hunsberger and Posenato (2024) defined a real-time execution algorithm RTE* for ESTNUs that, like the TD algorithm for STNs, provides maximum flexibility but requires minimal computation. They then defined an ESTNU $\mathcal{S}$ to be dispatchable if every execution of the RTE* algorithm is guaranteed to satisfy the constraints in $\mathcal{S}$. Next, they proved the following claim due to Morris.

Theorem 2. *An ESTNU is* dispatchable *if and only if all of its STN projections are dispatchable (as STNs).*

This theorem is invaluable in proving the correctness of algorithms related to ESTNU dispatchability since, together with Theorem 1, it connects ESTNU dispatchability to the existence of shortest vee-paths in STN projections.

Most DC-checking algorithms do not output a dispatchable ESTNU. However, as Morris (2014) noted, his $O(n^3)$-time DC-checking algorithm can be modified to generate waits and, in doing so, guarantee an equivalent dispatchable output. More recently, Hunsberger and Posenato (2023) presented a faster algorithm for generating equivalent dispatchable ESTNUs. However, neither algorithm makes any claim about the number of edges in the dispatchable output. Since that number impacts the performance of the real-time execution algorithm, producing an equivalent dispatchable ESTNU with a minimal number of edges is of practical importance.

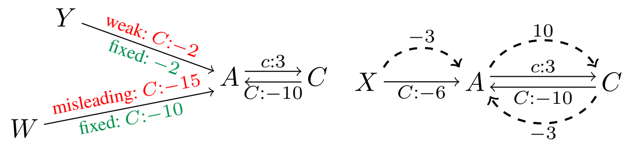
Definition 9. Given an ESTNU graph $\mathcal{G}$, if $\mathcal{G}'$ is an equivalent dispatchable ESTNU having a minimal number of edges, then $\mathcal{G}'$ is called a μESTNU for $\mathcal{G}$.

This paper presents the first algorithm for computing μESTNUs. It runs in $O(kn^3)$ worst-case time.

Preliminary Observations

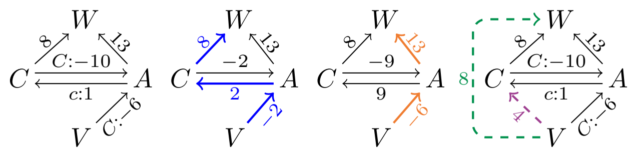
This section introduces ESTNU structures that play important roles in computing μESTNUs. It also previews cases of wait edges that can be removed while preserving dispatchability.

As noted earlier, an ESTNU is dispatchable if and only if all of its STN projections are dispatchable—as STNs. This implies that if there is a path from some V to some W in an ESTNU, then in each projection, there must be a shortest vee-path from V to W. However, the shortest vee-paths in different projections need not follow the same routes from V to W. As a result, a combination of shortest vee-paths across different projections may entail an (implicit) ordinary edge from V to W that is stronger than any pre-existing edge. Our new algorithm begins by generating what we call *stand-in* edges—that is, *ordinary* edges that explicitly represent implicit constraints entailed by combinations of ESTNU edges across different STN projections. The stand-in edges can be used to determine what other edges can be removed from the network without threatening its dispatchability. After playing this role, the stand-in edges are eventually removed—because other ESTNU paths or edges entail them. Before considering stand-in edges, we must fix any *weak* or *misleading* waits.



(a) *Weak* and *misleading* waits (b) (Dashed) stand-in edges

Figure 1: Managing labeled ESTNU edges



(a) ESTNU (b) $\omega_c = 2$ (c) $\omega_c = 9$ (d) stand-ins

Figure 2: A sample ESTNU, two of its projections with (colored) shortest *vee-paths*, and two (dashed) stand-in edges

Fixing "weak" and "misleading" waits. In Figure 1a, $(Y, C{:}{-2}, A)$ is a *weak wait,* since C cannot execute before the wait time of 2 expires. So this wait is effectively unconditional and hence is *replaced* by the ordinary edge $(Y, -2, A)$, as per the *Unconditional Unordered Reduction* rule of Morris, Muscettola, and Vidal (2001). In contrast, $(W, C{:}{-15}, A)$ is a *misleading wait* since C *must* execute no later than 10 after A. This wait is fixed by changing the wait time to the maximum of 10: $(W, C{:}{-10}, A)$.

Stand-in edges for LC, UC, and wait edges. Figure 1b shows (dashed) stand-in edges that represent the strongest ordinary constraints that are entailed by labeled ESTNU edges associated with the contingent link $(A, 3, 10, C)$. The stand-in edge for the LC edge $(A, c{:}3, C)$ is $(A, 10, C)$, representing that $C - A \leq 10$ in every projection. The stand-in edge for the UC edge $(A, C{:}{-10}, C)$ is $(C, -3, A)$, representing that $A - C \leq -3$ (i.e., $C - A \geq 3$) in every projection. For the wait edge $(X, C{:}{-6}, A)$, the stand-in edge is $(X, -3, A)$, representing that X must *unconditionally* wait at least 3 after A, since C cannot execute sooner than that.

Stand-in edges for diamond structures. Consider the ESTNU in Figure 2a. Since it has only one contingent link, each projection is determined by the value ω_c that it assigns to $C - A$. In the projection where $\omega_c = 2$, shown in Figure 2b, the shortest vee-path from V to W, indi-

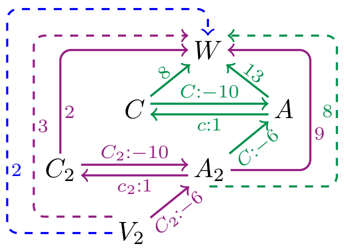
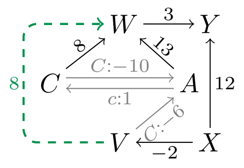


Figure 3: Stand-in edges entailed by nested diamonds (left); using a stand-in edge to dominate the edge $(X, 12, Y)$ (right)

cated by thick blue edges, has length 8, whereas in the projection shown in Figure 2c, where $\omega_c = 9$, the shortest vee-path from V to W, indicated by thick orange edges, has length 7. It is not hard to check that for this ESTNU, the length of the shortest vee-path from V to W is at most 8 in every projection. For example, in projections where $\omega_c \leq 5$, $|VACW|_\omega = \max\{-\omega_c, -6\} + \omega_c + 8 = \max\{8, \omega_c + 2\} \leq 8$. But in projections where $\omega_c \geq 5$, $|VAW|_\omega = \max\{-\omega_c, -6\} + 13 = \max\{13 - \omega_c, 7\} \leq 8$. Hence, this combination of ordinary and labeled edges entails the (green, dashed) stand-in edge $(V, 8, W)$ in Figure 2d.

The VAC rule. The edges in Figure 2a also entail the stand-in edge $(V, 4, C)$, shown as purple and dashed in Figure 2d, since in each projection, $|VAC|_\omega = \max\{-6, -\omega_c\} + \omega_c = \max\{\omega_c - 6, 0\} \leq 4$, since $\omega_c \leq 10$.

Stand-in edges for nested diamonds. The left of Figure 3 shows a more complicated ESTNU, where the diamond structure involving the solid green edges is nested inside the diamond structure involving the solid purple edges. Ignoring the green edges, for now, the solid purple edges can be shown to entail the (purple, dashed) stand-in edge $(V_2, 3, W)$. In particular, in projections where $\omega_2 = C_2 - A_2 \leq 7$, the length of the path $V_2A_2C_2W$ is: $\max\{-\omega_2, -6\} + \omega_2 + 2 = \max\{2, \omega_2 - 4\} \leq 3$. In contrast, if $\omega_2 \geq 7$, the length of the alternative path V_2A_2W is: $\max\{-\omega_2, -6\} + 9 = \max\{9 - \omega_2, 3\} \leq 3$.

Next, since the green diamond is isomorphic to the diamond from Figure 2a, it entails the (green, dashed) stand-in edge $(A_2, 8, W)$. But now, using that stand-in edge instead of the purple edge $(A_2, 9, W)$, a new analysis of the purple structure shows that it now entails a stronger (blue, dashed) stand-in edge $(V_2, 2, W)$. In other words, nested diamond structures can sometimes combine to entail stronger stand-in edges. Not only that, the order in which nested diamonds are analyzed can affect the overall computational effort required.

The importance of stand-in edges. The network in the right of Figure 3 includes the diamond structure from Figure 2a (with its labeled edges drawn in gray), its (dashed) stand-in edge $(V, 8, W)$, and some additional edges. Assuming that the stand-in edge is present in the network, then the *vee-path* $XVWY$, which contains only ordinary edges, has length $-2 + 8 + 3 = 9$, which is less than the length of the edge $(X, 12, Y)$. That implies that the edge $(X, 12, Y)$ can be removed from the network without affecting its dispatchability. Had the pre-existing edge been $(X, 9, Y)$, it could still be removed, given the alternative vee-path. Crucially,

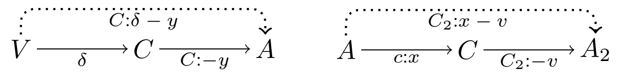

(a) Upper-Case Rule (b) Cross-Case Rule

Figure 4: Rules for generating wait edges

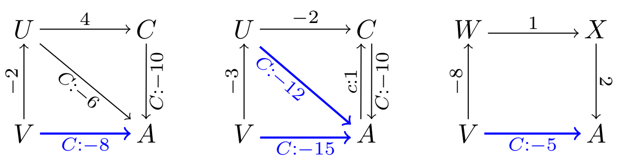

(a) Dominated by another wait (b) Dominated by a UC edge (c) Dominated by an ordinary path

Figure 5: Eamples of (blue) wait edges that can be removed without threatening ESTNU dispatcability

this conclusion depended solely on ordinary edges. Once the edge $(X, 12, Y)$ has been removed, the stand-in edge has played its role and can be discarded, leaving behind the SVPs $XVAWY$ and $XVACWY$ from X to Y.

Removing unneeded wait edges. Computing a μESTNU also requires removing three cases of wait edges that are not needed for dispatchability: wait edges that are dominated by (1) other wait edges, (2) UC edges; or (3) ordinary paths.

First, we review the rules of Morris and Muscettola (2005) that are used to generate wait edges. The waits generated by these rules *must* be satisfied by *every* valid execution strategy.

The *Upper Case* (UC) rule in Figure 4a combines an ordinary edge (V, δ, C) and a UC edge $(C, C{:}{-y}, A)$ to generate a (dotted) wait edge $(V, C{:}\delta - y, A)$. Intuitively, since $C - A$ might be as big as y, if we want $C - V \leq \delta$, then V must *wait* $(y - \delta)$ after A (unless C happens to execute early).

Similarly, the *Cross Case* (CC) rule in Figure 4b combines the LC edge $(A, c{:}x, C)$ for one contingent link with a wait edge $(C, C_2{:}{-v}, A_2)$ labeled by a *different* contingent timepoint C_2 to generate a (dotted) wait edge $(C, C_2{:}x - v, A_2)$. Intuitively, since C might be as little as x after A, but C must wait at least v after A_2 (unless C_2 executes early), then A must *wait* at least $(v - x)$ after A_2 (unless C_2 executes early).

Waits dominated by other waits. Figure 5a shows an ESTNU having a (blue) wait edge $(V, C{:}{-8}, A)$ that is dominated by another wait edge $(U, C{:}{-6}, A)$. The key feature is that the path from V to U (in this case, a single edge) has a negative length. As a result, in any situation ω_c, $|VUA|_{\omega_c} = -2 + \max\{-6, -\omega_c\} = \max\{-8, -2 - \omega_c\} \leq \max\{-8, -\omega_c\} = |(V, C{:}{-8}, A)|_{\omega_c}$.

Waits dominated by UC edges. In Figure 5b, two (blue) wait edges, $(V, C{:}{-15}, A)$ and $(U, C{:}{-12}, A)$, are dominated by the UC edge $(C, C{:}{-10}, A)$ via the paths, $VUCA$ and UCA, respectively. For each $\omega_c \in [1, 10]$: $|VUCA|_{\omega_c} = -5 + |(C{:}{-10}, A)|_{\omega_c} = -5 - \omega_c \leq \max\{-15, -\omega_c\} = |(V, C{:}{-15}, A)|_{\omega_c}$. Crucially, $\mathrm{d}(V, C) = -5 < 0$.

Waits dominated by ordinary paths. In Figure 5c, the (blue) wait $(V, C{:}{-5}, A)$ is dominated by the ordinary path $VWXA$, since for each ω_c, $|VWXA|_{\omega_c} = -5 \leq \max\{-5, -\omega_c\} = |(V, C{:}{-5}, A)|_{\omega_c}$.

Algorithm 1: *minDisp*$_{\text{ESTNU}}$

```
Input: $\mathcal{G} = (\mathcal{T},\ \mathcal{E}_{\rm o} \cup \mathcal{E}_{\rm lc} \cup \mathcal{E}_{\rm uc} \cup \mathcal{E}_{\rm ucg})$, dispatchable ESTNU
Output: A $\mu$ESTNU for $\mathcal{G}$
$(\mathcal{E}_o^{\rm si}, d) :=$ genStandIns$(\mathcal{T},\ \mathcal{E}_{\rm o} \cup \mathcal{E}_{\rm lc} \cup \mathcal{E}_{\rm uc} \cup \mathcal{E}_{\rm ucg})$
// STN dispatchability on ord. edges, reorienting labeled edges
$(\mathcal{T}, \mathcal{E}_o^*, \hat{\mathcal{E}}_{\rm lc}, \hat{\mathcal{E}}_{\rm uc}, \hat{\mathcal{E}}_{\rm ucg}) := disp_{stn}(\mathcal{T}, \mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si}, \mathcal{E}_{\rm lc}, \mathcal{E}_{\rm uc}, \mathcal{E}_{\rm ucg})$
$\hat{\mathcal{E}}_o^* := \mathcal{E}_o^* \backslash \mathcal{E}_o^{\rm si}$ // Remove any remaining stand-in edges from $\mathcal{E}_o^*$
$\hat{\mathcal{E}}_{\rm ucg} := \hat{\mathcal{E}}_{\rm ucg} \backslash$markWaits$(\mathcal{T}_c, \hat{\mathcal{E}}_{\rm ucg}, d)$// Remove domin. waits
return $\mathcal{G} = (\mathcal{T}, \hat{\mathcal{E}}_o^* \cup \hat{\mathcal{E}}_{\rm lc} \cup \hat{\mathcal{E}}_{\rm uc} \cup \hat{\mathcal{E}}_{\rm ucg})$
```

The *minDisp*$_{\text{ESTNU}}$ Algorithm

This section introduces our new *minDisp*$_{\text{ESTNU}}$ algorithm. It takes a dispatchable ESTNU $\mathcal{G} = (\mathcal{T}, \mathcal{E}_{\rm o}, \mathcal{E}_{\rm lc}, \mathcal{E}_{\rm uc}, \mathcal{E}_{\rm ucg})$ as input, and generates as output a μESTNU (i.e., an equivalent dispatchable ESTNU with a minimal number of edges). Since the input is dispatchable, the algorithm must only determine which edges can be removed while preserving dispatchability.

The algorithm has four steps. First, it computes $\mathcal{E}_o^{\rm si}$, the set of stand-in edges entailed by combinations of ESTNU edges. These stand-in edges only play a supporting role and, hence, are removed before the end of the algorithm. Second, it uses the STN-based dispatchability algorithm from Tsamardinos, Muscettola, and Morris (1998) to transform the STN $(\mathcal{T}, \mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si})$ into an equivalent dispatchable STN $(\mathcal{T}, \mathcal{E}_o^*)$ with a minimal number of edges (for the STN). Third, if any stand-in edges remain in $\mathcal{E}_o^*$, it removes them. Fourth, it removes from $\mathcal{E}_{\rm ucg}$ all wait edges that are dominated by (1) ordinary paths; (2) UC edges; or (3) other waits, resulting in a set $\hat{\mathcal{E}}_{\rm ucg}$. Finally, *minDisp*$_{\text{ESTNU}}$ outputs the μESTNU determined by the ordinary edges in $\hat{\mathcal{E}}_o^*$ and the waits in $\hat{\mathcal{E}}_{\rm ucg}$.

Algorithm pseudocode. Algorithm 1 shows the pseudocode for *minDisp*$_{\text{ESTNU}}$. At Line 1, it calls *genStandIns* (Algorithm 2), which computes the set $\mathcal{E}_o^{\rm si}$ of all stand-in edges. The edges in $\mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si}$ encode the strongest *ordinary* constraints entailed by the input ESTNU. *genStandIns* also *fixes* all "weak" or "misleading" wait edges (Lines 5-9), and computes the distance function d for the STN $(\mathcal{T}, \mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si})$.

At Line 2, *minDisp*$_{\text{ESTNU}}$ applies the *STN* dispatchability algorithm $disp_{stn}$ to the STN $(\mathcal{T}, \mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si})$ to compute an equivalent, dispatchable STN $(\mathcal{T}, \mathcal{E}_o^*)$, typically by inserting some new ordinary edges and deleting others. The $disp_{stn}$ algorithm identifies any *rigid components* in the input STN and then *reorients* its edges to make them incident only to the *representative* timepoints of those rigid components.[1] So we modified $disp_{stn}$ to similarly reorient the *labeled* edges from the ESTNU. That is why $disp_{stn}$ takes the labeled edge sets, $\mathcal{E}_{\rm lc}, \mathcal{E}_{\rm uc}$ and $\mathcal{E}_{\rm ucg}$, as extra inputs and returns the corresponding sets of reoriented edges, $\hat{\mathcal{E}}_{\rm lc}, \hat{\mathcal{E}}_{\rm uc}$ and $\hat{\mathcal{E}}_{\rm ucg}$, as extra outputs. (The labeled edges are only reoriented; none are lost or added in the process.) After applying $disp_{stn}$, the stand-in edges in $\mathcal{E}_o^{\rm si}$ have served their purpose and, therefore, any stand-ins happening to remain in $\mathcal{E}_o^*$ are removed (Line 3).

Finally, at Line 4, *minDisp*$_{\text{ESTNU}}$ calls the *markWaits* function (Algorithm 3) that collects the wait edges that are not needed for dispatchability and hence can be removed from the ESTNU. The μESTNU is output at Line 5.

[1]A rigid component is a set of timepoints whose values relative to one another are constrained to be fixed.

Algorithm 2: *genStandIns*

```
Input: $(\mathcal{T}_x \cup \mathcal{T}_c,\ \mathcal{E}_{\rm o} \cup \mathcal{E}_{\rm lc} \cup \mathcal{E}_{\rm uc} \cup \mathcal{E}_{\rm ucg})$, dispatchable ESTNU
Output: $(\mathcal{E}_o^{\rm si}, d)$, $\mathcal{E}_o^{\rm si}$ = set of stand-in ord. edges, and $d$ =
        distance function for $\mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si}$.
$\mathcal{L} :=$ the contingent links associated with $\mathcal{G}$;  $\mathcal{E}_o^{\rm si} := \emptyset$
foreach $(A, x, y, C) \in \mathcal{L}$ do
    // Collect stand-in edges for LC and UC edges
    $\mathcal{E}_o^{\rm si} := \mathcal{E}_o^{\rm si} \cup \{(A, y, C), (C, -x, A)\}$
    foreach $(V, C{:}{-v}, A) \in \mathcal{E}_{\rm ucg}$ do
        if $-v \geq -x$ then // Replace weak wait by ord edge
            $\mathcal{E}_{\rm ucg} := \mathcal{E}_{\rm ucg} \backslash \{(V, C{:}{-v}, A)\}$
            $\mathcal{E}_{\rm o} := \mathcal{E}_{\rm o} \cup \{(V, -v, A)\}$
        else if $-v < -y$ then // Adjust misleading wait
            $\mathcal{E}_{\rm ucg} := \mathcal{E}_{\rm ucg} \backslash \{(V, C{:}{-v}, A)\} \cup \{V, C{:}{-y}, A)\}$
        // Collect stand-in edge for wait edge
        if $-v < -x$ then $\mathcal{E}_o^{\rm si} := \mathcal{E}_o^{\rm si} \cup \{(V, -x, A)\}$
for $i := 1$ to $k$ do // $k$ = max depth of nested diamond struct.
    $d := \text{Johnson}(\mathcal{T}, \mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si})$// Shortest ord path lengths
    $edgeAdded := \bot$
    foreach $(A, x, y, C) \in \mathcal{L}$ do
        foreach $(V, C{:}{-q}, A) \in \mathcal{E}_{\rm ucg}$ do
            if $y - v \leq d(V, C)$ then    // Apply the VAC rule
                $\mathcal{E}_o^{\rm si} := \mathcal{E}_o^{\rm si} \cup \{(V, y - v, C)\}$
                $edgeAdded := \top$
            foreach $W \in \mathcal{T} \backslash \{A, C, V\}$ do
                // Explore $VACW$ "diamond" structures
                $\gamma := d(C, W)$;   $\delta := d(A, W)$
                if $\gamma < \infty$ and $\delta < \infty$ then
                    $\hat{\omega} := \delta - \gamma$
                    if $x < \hat{\omega} < y$ then
                        $\theta := \max\{-\hat{\omega}, -q\} + \delta$
                        if $\theta \leq d(V, W)$ then // stand-in edge!
                            $\mathcal{E}_o^{\rm si} := \mathcal{E}_o^{\rm si} \cup \{(V, \theta, W)\}$
                            $edgeAdded := \top$
    if $edgeAdded == \bot$ then break      // Exit from the for
if $edgeAdded$ then $d := \text{Johnson}(\mathcal{T}, \mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si})$   // Update $d$
return $(\mathcal{E}_o^{\rm si}, d)$                    // At this point, $d = d^*$
```

The next section covers *genStandIns* and *markWaits* in detail and proves important properties of Algorithms 1-3.

Formal Analysis

At Lines 5-9, *genStandIns* replaces any "weak waits" with ordinary edges and adjusts the wait times of any "misleading waits". At Lines 3 and 10, it collects stand-in edges entailed by the LC, UC, and (fixed) wait edges from the input ESTNU.

Lemma 1. *Each weak wait is equivalent to its ordinary replacement. Each misleading wait is equivalent to its fixed version. For each labeled edge e (fixed if needed), its stand-in edge ê satisfies* $\max_\omega\{|e|_\omega\} = |\hat{e}|$. *Hence, ê is entailed by e.*

Proof. For the contingent link (A, x, y, C), $\omega_c \in [x, y]$. For

Algorithm 3: *markWaits*

```
    Input: $\mathcal{T}_c$, contingent TPs; $\hat{\mathcal{E}}_{\text{ucg}}$, wait edges; $d$, distance fn.
    Output: A set $\mathcal{E}^m_{\text{ucg}} \subseteq \hat{\mathcal{E}}_{\text{ucg}}$ of wait edges marked for removal
1   $\mathcal{E}^m_{\text{ucg}} := \emptyset$
    // Collect waits dominated by ord paths,UC edges,or other waits
2   foreach $(V, C{:}{-v}, A) \in \hat{\mathcal{E}}_{\text{ucg}}$ do
3       if $d(V,A) \leq -v$ or $d(V,C) < 0$ then
4           $\mathcal{E}^m_{\text{ucg}} := \mathcal{E}^m_{\text{ucg}} \cup \{(V, C{:}{-v}, A)\}$
5       else
6           foreach $U \in \mathcal{T} \mid \exists(U, C{:}{-u}, A) \in \hat{\mathcal{E}}_{\text{ucg}}$ do
7               if $d(V,U) < 0$ and $d(V,U) - u \leq -v$ then
8                   $\mathcal{E}^m_{\text{ucg}} := \mathcal{E}^m_{\text{ucg}} \cup \{(V, C{:}{-v}, A)\}$
9   return $\mathcal{E}^m_{\text{ucg}}$
```

a weak wait, $G = (V, C{:}{-v}, A)$, $-v \geq -x$. Hence, $|G|_{\omega_c} = \max\{-v, -\omega_c\} = -v = |(V, -v, A)|$. For a misleading wait, $H = (W, C{:}{-v}, A)$, $-v < -y$. Hence, $|H|_{\omega_c} = \max\{-v, -\omega_c\} \leq \max\{-y, -\omega_c\} = |(V, C{:}{-y}, A)|_{\omega_c}$. The stand-in for $e = (A, c{:}x, C)$ is $\hat{e} = (A, y, C)$. Hence, $d^*(A,C) \leq \max_{\omega_c}\{|e|_{\omega_c}\} = \max_{\omega_c}\{\omega_c\} = y = |\hat{e}|$. The stand-in for $E = (C, C{:}{-y}, A)$ is $\hat{E} = (C, -x, A)$. Hence, $d^*(C,A) \leq \max_{\omega_c}\{|E|_{\omega_c}\} = \max_{\omega_c}\{-\omega_c\} = -x = |\hat{E}|$. The stand-in for a (fixed) wait, $F = (V, C{:}{-v}, A)$ is $\hat{F} = (V, -x, A)$. Hence, $d^*(V,A) \leq \max_{\omega_c}\{|F|_{\omega_c}\} = \max_{\omega_c}\{\max\{-v, -\omega_c\}\} = -x = |\hat{F}|_{\omega_c}$. □

Henceforth, this section assumes that $\mathcal{G}$ includes these initial stand-in edges and that all waits are fixed. It also restricts attention to *simple* paths (i.e., paths with no subsidiary cycles).

Next, the main loop of *genStandIns* (Lines 11-28) does at most k iterations, aiming to generate: (1) the stand-in edges derived from all possible applications of the VAC rule (Lines 16-18; recall the dashed purple stand-in edge from Figure 2d); and (2) the stand-in edges entailed by all possible $VACW$ "diamond" structures (Lines 20-27; recall the dashed green stand-in edge from Figure 2d). Theorems 3 and 4, and Corollary 1, below, ensure that *genStandIns* correctly generates all stand-in edges and computes all d^* values.

Definition 10 (Needed)**.** A contingent link (A, x, y, C) is *needed for* $d^*(U,W)$ in an ESTNU $\mathcal{G}$ if removing *all* of the labeled edges associated with that contingent link from $\mathcal{G}$ would change the value of $d^*(U,W)$.

Note that if no contingent links are needed for $d^*(U,W)$, then its value is determined solely by ordinary edges.

Lemma 2. *Suppose that* (A, x, y, C) *is the only contingent link needed for* $d^*(U,W)$*. Then the LC edge* $(A, c{:}x, C)$ *and some wait edge* $(V, C{:}{-v}, A)$ *are both needed for* $d^*(U,W)$*.*

Proof. Since only one contingent link is needed, the shortest vee-paths relevant to $d^*(U,W)$ in any projection derive from ESTNU paths involving zero or more ordinary edges and *at least one* edge labeled by c or C. Therefore, regarding situations, it suffices to restrict attention to the value of $\omega_c = C - A$. The result follows from showing that each of the following cases generates a contradiction.

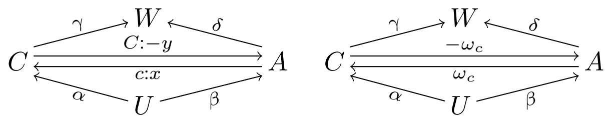
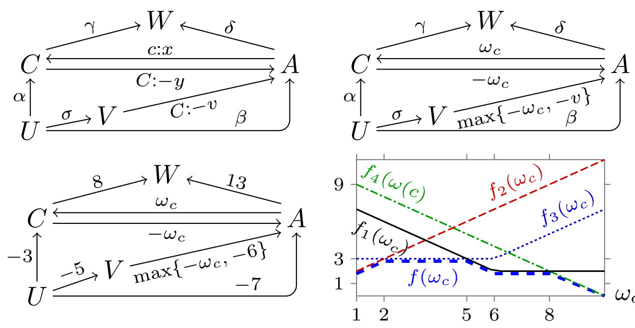


Figure 6: The diamond considered in Case 3 of Lemma 2



$$
\begin{aligned}
f_1(\omega_c) &= |UVAW|_\omega &&= \sigma + \max\{-v, -\omega_c\} + \delta \\
f_2(\omega_c) &= |UACW|_\omega &&= \beta + \omega_c + \gamma \\
f_3(\omega_c) &= |UVACW|_\omega &&= \sigma + \max\{-v, -\omega_c\} + \omega_c + \gamma \\
f_4(\omega_c) &= |UCAW|_\omega &&= \alpha - \omega_c + \delta \\
f(\omega_c) &= \min\{f_i(\omega_c)\}
\end{aligned}
$$

Figure 7: ESTNU paths and their projections (cf. Theorem 3)

Case 1: The LC edge $e = (A, c{:}x, C)$ *is the* **only** *needed edge.* Let $\mathcal{P}$ be an ESTNU path from U to W such that $|\mathcal{P}|_y \leq d^*(U,W)$. ($|\mathcal{P}|_y$ notates the length of $\mathcal{P}$'s projection in the situation where $\omega_c = y$.) Next, let $\mathcal{P}^{\text{si}}$ be the *ordinary* path that is the same as $\mathcal{P}$ except that e has been replaced by its stand-in edge (A, y, C). Then for *any* ω, we have that $|\mathcal{P}^{\text{si}}| = |\mathcal{P}|_y \leq d^*(U,W)$, contradicting the need for e.

Case 2: The LC edge $e = (A, c{:}x, C)$ *is* **not** *needed.* Let $\mathcal{P}$ be any ESTNU path from U to W, not containing e, such that $|\mathcal{P}|_x \leq d^*(U,W)$. Next, let $\mathcal{P}^{\text{si}}$ be the *ordinary* path that is the same as $\mathcal{P}$ except that the UC edge and any wait edges have been replaced by their stand-in edges. Then $|\mathcal{P}^{\text{si}}| = |\mathcal{P}|_x \leq d^*(U,W)$, contradicting the need for labeled edges.

Case 3: The **only** *needed edges are the LC edge* $(A, c{:}x, C)$ *and UC edge* $(C, C{:}{-y}, A)$. Then, each SVP derives from paths in Figure 6 (the arrows may denote ordinary *paths)*; $|UACW|_{\omega_c} = \beta + \omega_c + \gamma$ (increasing with ω_c); and $|UCAW|_{\omega_c} = \alpha - \omega_c + \delta$ (decreasing with ω_c). Since the LC and UC edges are both needed, we must have: $|UCW| > d^*(U,W)$ and $|UAW| > d^*(U,W)$; and the SVPs must be $UACW$ for smaller values of ω_c; and $UCAW$ for larger values. The *maximum* shortest path length occurs when $|UCAW|_{\omega_c} = |UACW|_{\omega_c}$, which occurs when $\omega_c = \frac{1}{2}(\alpha + \delta - \beta - \gamma)$. But that max value is $\frac{(\alpha+\gamma)+(\beta+\delta)}{2} = \frac{|UCW|+|UAW|}{2} > d^*(U,W)$, a contradiction. □

Theorem 3. *Suppose that* (A, x, y, C) *is the only contingent link needed for* $d^*(U,W)$*. Then,* $d^*(U,W)$ *equals* $\min\{\text{d}(U,V) + d^*(V,W) \mid \exists(V, C{:}-v, A) \in \mathcal{E}_{\text{ucg}}\}$.

Proof. By Lemma 2, the edges needed for $d^*(U,W)$ include the LC edge (A, x, y, C) and some wait edge $(V, C{:}{-v}, A)$.

As in the proof of Lemma 2, we refer to situations by the value of $\omega_c = C - A$. As illustrated in Figure 7, where the arrows labeled by Greek letters represent shortest ordinary paths, each shortest vee-path from U to W in any $\mathcal{G}_{\omega_c}$ must, by construction, be one of the four paths, $UVAW$, $UACW$, $UVACW$ or $UCAW$, whose lengths are functions of ω_c, as shown at the bottom of the figure. These length functions are plotted for the sample instance in the lower right of the figure. In addition, since the LC and wait edges are needed, we must have: $d^*(U,W) < |UCW|$ and $d^*(U,W) < |UAW|$.

Each of the length functions is continuous, piecewise-linear and monotone (i.e., non-decreasing or non-increasing). Thus, the *minimum* of the length functions, $f(\omega_c) = \min_{1\le i\le 4}\{f_i(\omega_c)\}$ (blue and dashed in Figure 7) must be continuous and piecewise linear. So the *maximum* value of $f(\omega_c)$ (i.e., $d^*(U,W)$) must occur at an endpoint of one of the piecewise-linear segments (i.e., where two of the length functions intersect). If $\mathcal{I}$ is the set of values of ω_c at all points of intersection, then $d^*(U,W) = \max\{f(\omega_c) \mid \omega_c \in \mathcal{I}\}$. In the sample plot, $d^*(U,W) = 3$.

Let τ_{ij} denote the value of ω_c where $f_i(\omega_c) = f_j(\omega_c)$. For example, τ_{13} denotes the value of ω_c at which $f_1(\omega_c) = f_3(\omega_c)$ (i.e., where $|UVAW|_{\omega_c} = |UVACW|_{\omega_c}$). Intersection points that depend on the projected length of the wait edge (i.e., $\max\{-v, -\omega_c\}$) are given the superscript v (for the case where $-v \ge -\omega_c$) or ω (for $-\omega_c \ge -v$). It is easy but tedious to compute all possible points of intersection:

$$\begin{aligned} \tau^v_{12} &= \sigma+\delta-v-\beta-\gamma & \tau^\omega_{12} &= \tfrac{\sigma+\delta-\beta-\gamma}{2} \\ \tau_{13} &= \delta-\gamma & \tau^v_{14} &= \alpha-\sigma+v \\ \tau^\omega_{23} &= \sigma-\beta & \tau_{24} &= \tfrac{\alpha+\delta-\beta-\gamma}{2} \\ \tau^v_{34} &= \tfrac{\alpha+\delta-\sigma+v-\gamma}{2} & \tau^\omega_{34} &= \alpha+\delta-\sigma-\gamma \end{aligned}$$

Claim: For each $\tau \in \mathcal{I}$, $f(\tau) = \min\{f_1(\tau), f_3(\tau)\} = \min\{|UVAW|_{\omega_c}, |UVACW|_{\omega_c}\}$. In other words, the two paths $UVAW$ and $UVACW$ suffice to determine $d^*(U,W)$, which requires computing only their *single* point of intersection: $\tau_{13} = \delta - \gamma$. At that point, $|UVAW|_{\omega_c} = |UVACW|_{\omega_c} = \sigma + \max\{\delta - v, \gamma\}$. In the sample plot, $\delta - \gamma = 13 - 8 = 5$ and $d^*(U,W) = -5 + \max\{7, 8\} = 3$.

Proof of claim. For lack of space, we only show the proof for τ^v_{12} (i.e., where $f_1 = f_2$ and $-v \ge -\omega_c$). If the minimum is $f_1(\tau^v_{12}) = f_2(\tau^v_{12})$ or $f_3(\tau^v_{12})$, then the claim holds. It only remains to show that the min is *not* $f_4(\tau^w_{12}) = \frac{\alpha-\sigma}{2} + \frac{\alpha+\gamma+\delta+\beta}{2}$. Now $(\alpha - \sigma) \ge 0$, since otherwise $|UCA|_{\omega_c} = \alpha - \omega_c < \sigma - \omega_c \le \sigma + \max\{-\omega_c, -v\} = |UVA|_{\omega_c}$, contradicting the need for the wait edge. Hence, $f_4(\tau^w_{12}) \ge \frac{(\alpha+\gamma)+(\delta+\beta)}{2} = \frac{|UCW|+|UAW|}{2} > d^*(U,W)$. □

Finally, note that $d^*(V,W) < \max\{\delta - v, \gamma\}$ would yield a contradiction since it would entail the existence of alternative *shorter* pathways from U to V to W in every projection. Therefore, $d^*(V,W) = \max\{\delta - v, \gamma\}$; and, assuming $(V, C{:}{-v}, A)$ is the needed wait edge, $d^*(U,W) = \sigma + d^*(V,W) = \mathrm{d}(U,V) + d^*(V,W)$.

So far, the proof assumed that $W \notin \{A, C\}$. However, (A,x,y,C) cannot be needed for $d^*(V,A)$ since the LC edge cannot be in any simple SVP ending in A. And the only way (A,x,y,C) can be needed for any $d^*(V,C)$ is if the SVP is the concatenation of $(V, C{:}{-v}, A)$ and the LC edge, which yields the stand-in edge generated by the VAC rule: $|VAC|_{\omega_c} = \max\{-v, -\omega_c\} + \omega_c = \max\{\omega_c - v, 0\} = y - v$. □

Definition 11. For any U and W, let $\|(U,W)\|$ denote the minimum number of contingent links needed for $d^*(U,W)$.

Theorem 4. *For any U and W, if $\|(U,W)\| = h > 0$, then $d^*(U,W) = \min\{\mathrm{d}(U,V) + d^*(V,W) \mid \exists (V, C{:}-v, A) \in \mathcal{E}_{\rm ucg}$ and $\|(A,W)\| < h\}$.*

Proof. The case $h = 1$ is given by Theorem 3. For the recursive case, suppose $h > 1$. Among the set $\mathcal{L}^*$ of contingent links (CLs) whose edges are needed for $d^*(U,W)$, let (A,x,y,C) be one whose activation timepoint (ATP) is *not* constrained to occur *before* the ATP for any other CL in $\mathcal{L}^*$; and let $(V, C{:}{-v}, A)$ be any wait edge needed for $d^*(U,W)$.

By the definition of vee-path, since LC edges are non-negative, no LC edge can *precede* either the UC edge $(C, C{:}{-y}, A)$ or the wait edge $(V, C{:}{-v}, A)$ in any SVP; nor, by the choice of (A,x,y,C), can any UC or wait edge from CLs in $\mathcal{L}^*$ *precede* $(C, C{:}{-y}, A)$ or $(V, C{:}{-v}, A)$. In addition, since UC and wait edges are negative, no UC or wait edge can *follow* the LC edge $(A, c{:}x, C)$ in any SVP. Therefore, by Lemma 2, no CLs from $\mathcal{L}^*$ can be needed for the paths, UC, UV or CW, shown in Figure 7; so without loss of generality, they can be assumed to be ordinary paths. In contrast, the path AW *may* use labeled edges from CLs in $\mathcal{L}^*$. (Recall the *nested diamonds* in Figure 3.) In addition, since $\|(U,W)\| = h$, it follows that $\|(A,W)\| < h$.

Let ω^p denote a *partial* situation that specifies durations for all contingent links *except* (A,x,y,C). Choose ω^p such that $|AW|_{\omega^p} = d^*(A,W)$. Then for any $\omega_c \in [x,y]$, let ω^p_c be the (full) situation that extends ω^p to include $C - A = \omega_c$. So, across all such situations ω^p_c, the path lengths, $\gamma = |CW|$ and $\delta = |AW|_{\omega^p} = d^*(A,W)$, are fixed. Hence, by Theorem 3, for any wait edge $(V, C{:}{-v}, A)$, we get that $\max_{\omega_c}\{d_{\omega^p_c}(V,W)\} = \max\{\delta - v, \gamma\}$. Furthermore, since for any ω, $|AW|_\omega \le d^*(A,W)$, it follows that $|VAW|_\omega \le |VAW|_{\omega^p}$, implying that $d^*(V,W) = \max_{\omega_c}\{d_{\omega^p_c}(V,W)\} = \max\{\delta - v, \gamma\}$. Since (A,x,y,C) is needed for $d^*(U,W)$, any SVP from U to W, in any situation ω, must pass through some wait $(V, C{:}{-v}, A)$, from which the result follows. □

Corollary 1. *Algorithm 2 (*genStandIns*) correctly computes $d^*(U,W)$ for each pair of timepoints U and W* **(i.e., at the end of the algorithm,** *$d = d^*$).*

Proof. The *genStandIns* algorithm initially computes the stand-in values entailed by all LC, UC, and wait edges. In a DC network, there cannot be any further squeezing of LC or UC edges. However, alternative paths may subsequently be found that dominate wait edges.

If $\|(U,W)\| = 0$, then $d^*(U,W)$ is obtained by the initial call to Johnson's algorithm. Otherwise, by Theorem 4, computing $d^*(U,W)$ depends only on the value of $d^*(V,W)$ corresponding to some wait edge $(V, C{:}{-v}, A)$. After the first iteration of the main loop, *genStandIns* has correctly computed the $d^*(U,W)$ values for all U and W such that

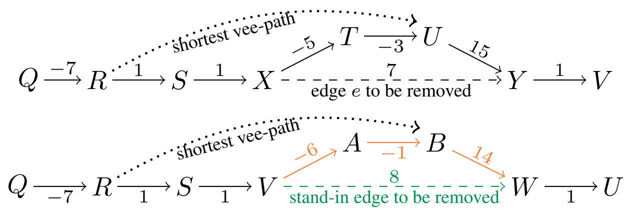


Figure 8: Paths RE: Lemma 3 (top) and Theorem 5 (bottom)

$\|(U,W)\| = 1$. Those values are incorporated as ordinary stand-in edges for the next round. In general, after the i^{th} round, it has correctly computed the $d^*(U,W)$ values for all U and W such that $\|(U,W)\| \le i$. Since the labeled edges from different contingent links can only be nested to a maximum depth of k (because each level of nesting involves a precedence relation among the participating activation timepoints), at most k iterations of the algorithm are needed. □

In addition to $disp_{stn}$ typically removing some ordinary edges, *markWaits* collects different classes of wait edges to be removed from the ESTNU. At Line 3, it collects waits dominated by ordinary paths (see the condition, $\mathrm{d}(V,A) \le -v$) or UC edges (see the condition, $\mathrm{d}(V,C) < 0$). Then, at Lines 6-8, it collects all waits dominated by other waits. Theorem 5, below, ensures that removing these edges cannot threaten the dispatchability of the ESTNU. But first, we present an important result about *STN* dispatchability.

Lemma 3. *Suppose that $e = (X,\delta,Y)$ is an (ordinary) edge in a dispatchable* STN *$\mathcal{G}_o$, and $\mathcal{P}_e$ is a shortest vee-path from X to Y in $\mathcal{G}_o$ that does* not *use e. Then removing e from $\mathcal{G}_o$ preserves the dispatchability of $\mathcal{G}_o$.*

Proof. Let $\mathcal{P}$ be any shortest vee-path (SVP) from some Q to some V in $\mathcal{G}_o$ that includes e. Let $\mathcal{P}'$ be the path obtained from $\mathcal{P}$ by replacing e by $\mathcal{P}_e$. Since $\mathcal{P}_e$ is an SVP, it follows that $|\mathcal{P}_e| \le \delta$, and $|\mathcal{P}'| \le |\mathcal{P}|$, which implies that $\mathcal{P}'$ is a shortest path. Suppose, however, that $\mathcal{P}'$ is not a vee-path.

Case 1: Replacing e by $\mathcal{P}_e$ inserts a negative edge after a non-negative edge. This scenario is illustrated in the top of Figure 8, where $e = (X,7,Y)$, $\mathcal{P}$ is the path $QRSXYV$, $\mathcal{P}_e$ is the path $XTUY$, and replacing e by $\mathcal{P}_e$ inserts the negative edge XT after the non-negative edge SX. In this scenario, $\delta = |e|$ must be non-negative since the edge e follows the non-negative edge SX in a vee-path. In addition, U terminates the last negative edge in $\mathcal{P}_e$; hence, the subpath from X to U is negative. However, since there is a path from R to U in $\mathcal{G}_o$, the dispatchability of $\mathcal{G}_o$ ensures that there is an SVP from R to U, shown as dotted in the figure. Substituting this (dotted) vee-path for the sub-path $RSXTU$ in $\mathcal{P}'$, provides an SVP from Q to V, despite the removal of e. (If e is used in any shortest path $\mathcal{P}^\dagger$ from R to U, then the suffix of $\mathcal{P}^\dagger$ from X to U must be negative, given that XTU is a shortest path with $|XTU| < 0$. But then $\mathcal{P}^\dagger$ cannot be a vee-path.) The same argument works in general where R is the last timepoint in the subpath from Q to X that terminates a negative edge, and U is the last timepoint in $\mathcal{P}_e$ that terminates a negative edge.

Case 2: Replacing e by $\mathcal{P}_e$ inserts a non-negative edge before a negative edge. Handled similarly. □

Theorem 5. *Algorithm 1 (*minDisp$_{\text{ESTNU}}$*) preserves the dispatchability of its input ESTNU.*

Proof. The first step of *minDisp*$_{\text{ESTNU}}$ inserts the ordinary edges collected by *genStandIns* (Algorithm 2). Since they are entailed by the network, they cannot disturb dispatchability. The second step runs the $disp_{stn}$ algorithm on the ordinary edges, which can insert some edges while removing others. With no loss of generality, suppose it first inserts all the new edges and then removes edges. Let $e = (X,\delta,Y)$ be the first edge whose removal thwarts the ESTNU's dispatchability. Since $disp_{stn}$ preserves dispatchability among the ordinary edges, there must be an alternative shortest (ordinary) vee-path $\mathcal{P}_e$ from X to Y with $|\mathcal{P}_e| \le |e|$. But e might also be used by an SVP $\mathcal{P}$ from some Q to some V in a projection $\mathcal{G}_\omega$ such that replacing e by $\mathcal{P}_e$ creates a non-vee-path. Since $\mathcal{G}_\omega$ is a dispatchable STN, Lemma 3 ensures the existence of an SVP from Q to V in $\mathcal{G}_\omega$ that does not use e, a contradiction.

The third step of *minDisp*$_{\text{ESTNU}}$ is to remove any remaining stand-in edges. The same kind of argument ensures that their removal cannot thwart the ESTNU's dispatchability. For example, recall the stand-in edge $(V,8,W)$ derived from the $VACW$ diamond structure in Figure 2. In the projection where $\omega_c = 9$ (cf. Figure 2c), the relevant SVP is VAW of length 7. In any diamond structure, the path CW must be non-negative since it follows the LC edge in an SVP. Hence, the path AW must also be non-negative. (Otherwise, the intersection point $\omega_c = \delta - \gamma$ could not lie within (x,y).) However, the SVP from A to W might begin with a negative edge, say $(A,-1,B)$, as illustrated in the bottom of Figure 8. (In general, B would be chosen as the terminus of the last negative edge in the SVP from A to W.) As in Figure 8 and Lemma 3, since there is a path from R to B in this projection, there must be an SVP from R to B, shown as dotted in Figure 8, which ensures the existence of an SVP from Q to U that does not use the stand-in edge $(V,8,W)$.

Finally, the fourth step of *minDisp*$_{\text{ESTNU}}$ removes three classes of waits collected by *markWaits* (Algorithm 3, Lines 3-8). Let $e = (V, C{:}{-v}, A)$ be the first wait edge whose removal thwarts the ESTNU's dispatchability; and let (A,x,y,C) be the corresponding contingent link.

Case 1: $d^*(V,A) \le -v$. Since *genStandIns* removed all weak waits (Lines 5-7), it follows that $-v < -x$, where $-x$ is the length of this wait's stand-in edge. So, prior to the third step's removal of the remaining stand-in edges, there must have been an *alternative* ordinary (simple) path $\mathcal{P}_e$ from V to A of some length $\delta \le -v \le \max\{-v, -\omega_c\} = |(V, C{:}{-v}, A)|_{\omega_c}$, where some of $\mathcal{P}_e$'s edges were stand-in edges. However, since $-v < -x$, $\mathcal{P}_e$ cannot be *just* the stand-in edge $(V,-x,A)$. Now, in the third step, the stand-in edges in $\mathcal{P}_e$ were, in effect, replaced by alternative SVPs but, as shown above, Lemma 3 ensures that this preserves ESTNU dispatchability, providing in each projection an alternative SVP from V to A of length at most $-v \le |(V, C{:}{-v}, A)|_{\omega_c}$.

Case 2: $d^*(V,C) < 0$. Prior to the third step's removal of stand-in edges, there must be an ordinary SVP $\mathcal{P}$ from V

to C. As argued above, removing all stand-in edges from $\mathcal{P}$ cannot disturb the ESTNU dispatchability. Hence, in any projection $\mathcal{G}_\omega$, there must be an SVP $\mathcal{P}_\omega$ from V to C, where $|\mathcal{P}_\omega|_\omega \leq d^*(V,C) < 0$. Let R be the terminus of the last negative edge in $\mathcal{P}_\omega$. Since there is a path from R to A in $\mathcal{G}_\omega$ (using the suffix of $\mathcal{P}_\omega$ beginning at R together with the projected UC edge), there must be an SVP $\mathcal{P}_{\text{ra}}$ from R to A. Let $\mathcal{P}_{vr}$ be the prefix of $\mathcal{P}_\omega$ from V to R; and let $\mathcal{P}_{\text{vra}}$ be the vee-path obtained by concatenating $\mathcal{P}_{\text{vr}}$ and $\mathcal{P}_{\text{ra}}$. Then $|\mathcal{P}_{\text{vra}}|_\omega = |\mathcal{P}_{\text{vr}}|_\omega + |\mathcal{P}_{\text{ra}}|_\omega \leq |\mathcal{P}_\omega|_\omega + |(C,C{:}{-y},A)|_{\omega_c} < |(C,C{:}{-y},A)|_\omega = -\omega_c \leq \max\{-v,-\omega_c\} = |(V,C{:}{-v},A)|_{\omega_c}$, contradicting the need for $(V,C{:}{-v},A)$. By Lemma 3, even if the projection of $(V,C{:}{-v},A)$ were used in an SVP in some $\mathcal{G}_\omega$, replacing it by $\mathcal{P}_{\text{vra}}$ would not disturb the ESTNU's dispatchability.

Case 3: There is a wait $(U,C{:}{-u},A)$ with $d^*(V,U) < 0$ and $d^*(V,U) - u \leq -v$. As in Case 2, in any projection $\mathcal{G}_\omega$, there must be an SVP $\mathcal{P}_\omega$ from V to U with $|\mathcal{P}_\omega|_\omega \leq d^*(V,U) < 0$. Let R be the terminus of the last negative edge in $\mathcal{P}_\omega$; $\mathcal{P}_{\text{vr}}$, the prefix of $\mathcal{P}_\omega$ from V to R; $\mathcal{P}_{\text{ra}}$, an SVP from R to A; and $\mathcal{P}_{\text{vra}}$, the concatenation of $\mathcal{P}_{\text{vr}}$ and $\mathcal{P}_{\text{ra}}$. Then $|\mathcal{P}_{\text{vra}}|_\omega = |\mathcal{P}_{\text{vr}}|_\omega + |\mathcal{P}_{\text{ra}}|_\omega \leq |\mathcal{P}_\omega|_\omega + |(U,C{:}{-u},A)|_{\omega_c} \leq d^*(V,U) + \max\{-u,-\omega_c\} \leq \max\{d^*(V,U)-u,-\omega_c\} \leq \max\{-v,-\omega_c\} = |(V,C{:}{-v},A)|_{\omega_c}$. □

Theorem 6. *When given a dispatchable ESTNU as input, the* minDisp$_{\text{ESTNU}}$ *algorithm outputs an* equivalent *dispatchable ESTNU having a* minimal *number of edges (i.e., a μESTNU).*

Proof. The ESTNU output by *minDisp*$_{\text{ESTNU}}$ is equivalent to the input network since: (1) all fixed waits and stand-in edges are entailed by alternative edges or paths, (2) the $disp_{stn}$ algorithm generates an equivalent set of ordinary edges, and (3) each removed wait edge is entailed by an alternative path.

Minimality. Since Theorem 5 ensures that the output ESTNU is dispatchable, it suffices to show that each edge in the output ESTNU is needed for dispatchability (i.e., its removal would thwart dispatchability). To the contrary, suppose e is some edge *not* needed for dispatchability but belongs to the output ESTNU and, hence, was not removed. For reference, let $\mathcal{G}_o = (\mathcal{T}, \mathcal{E}_o^*)$ be the STN output by $disp_{stn}$.

Case 1: e is an ordinary edge (U,δ,W). Since all stand-in edges are eventually removed, e cannot be a stand-in edge; it must be an original ordinary edge or one inserted by $disp_{stn}$. Since $disp_{stn}$ outputs a minimal dispatchable STN, and it kept or inserted e, it follows that any vee-path $\mathcal{P}$ from U to W in $\mathcal{G}_o$ that does not use e must satisfy: $|\mathcal{P}| > \delta$. But that implies that removing e from the ESTNU would yield $d^*(U,V) > \delta = |e|$, contradicting that e is not needed.

Case 2: e is a wait edge $(V,C{:}{-v},A)$. Since e was not removed, the conditions in Lines 3-8 of *markWaits* must all be false. So $d^*(V,A) > -v$; $d^*(V,C) \geq 0$; and for all *other* waits $(U,C{:}{-u},A)$, $d^*(V,U) \geq 0$ or $d^*(V,U) - u > -v$. Since e is not needed, then for each ω with $\omega_c = y$, there must be a simple SVP $\mathcal{P}_{\text{va}}$ from V to A that does not use e, with $|\mathcal{P}_{\text{va}}|_\omega \leq |e|_y = -v$. Being simple, $\mathcal{P}_{\text{va}}$ cannot include an edge derived from the LC edge $(A,c{:}x,C)$; and if it includes an edge derived from the UC edge $(C,C{:}{-y},A)$ or some wait $(U,C{:}{-u},A)$, then it must be the last edge in $\mathcal{P}_{\text{va}}$.

Case 2a: For each ω, there is an SVP $\mathcal{P}_{\text{va}}$ from V to A that does not include any edge derived from edges labeled by C. Then for some ω, $d^*(V,A) = |\mathcal{P}_{\text{va}}|_\omega \leq -v$, a contradiction.

Case 2b: For some ω, an SVP $\mathcal{P}_{\text{va}}$ from V to A terminates in an edge derived from the UC edge $(C,C{:}{-y},A)$. Since $\mathcal{P}_{\text{va}}$ is an SVP ending in a negative edge, *all* of its edges, including the subpath from V to C, must be negative. Since negative edges in a projection derive only from negative edges in the ESTNU, it follows that $d^*(V,C) < 0$, a contradiction.

Case 2c: For some ω, a simple SVP $\mathcal{P}_{\text{va}}$ from V to A terminates in an edge derived from another wait $E_U = (U,C{:}{-u},A)$. Let $\mathcal{P}_{\text{vu}}$ be the prefix of $\mathcal{P}_{\text{va}}$ from V to U. Since the last edge in the SVP $\mathcal{P}_{\text{va}}$ is negative, all of the edges in $\mathcal{P}_{\text{vu}}$ must also be negative. Let ω^* be the same as ω except that $C - A = y$. If the edges in $\mathcal{P}_{\text{vu}}$ all derive from ordinary edges in the ESTNU, then $|\mathcal{P}_{\text{vu}}| = d^*(V,U)$ and, hence, $-v \geq |\mathcal{P}_{\text{va}}|_{\omega^*} = d^*(V,U) - u$, a contradiction. Therefore, $\mathcal{P}_{\text{vu}}$ must contain *one or more* waits $(W_i,C_i{:}{-w_i},A_i)$ or UC edges $(C_i,C_i{:}{-y_i},A_i)$ labeled by contingent TPs *other than* C. For each such edge, the path from A_i to A is an SVP with all negative edges; hence $d^*(A_i,A) < 0$ and so the LC edge $(A_i,c_i{:}x_i,C_i)$ cannot be part of *any* SVP ending at A.

Let $\mathcal{A}_U$ be the set of contingent links whose activation timepoints A_i appear in a proper prefix of *any* SVP in *any* projection whose terminal edge derives from E_U; and let ω^m be any situation where $C_i - A_i = x_i$ for each $(A_i,x_i,y_i,C_i) \in \mathcal{A}_U$, but $C - A = y$. Let $\mathcal{P}_{\text{va}}^U$ be any simple SVP from V to A whose terminal edge derives from E_U in $\mathcal{G}_{\omega^m}$, and let $\mathcal{P}_{\text{vu}}^U$ be the prefix of $\mathcal{P}_{\text{va}}^U$ from V to U. Since $\mathcal{P}_{\text{vu}}^U$ is simple, it cannot contain any edges labeled by C or c. By construction, $|\mathcal{P}_{\text{va}}^U|_{\omega^m} \leq -v$, but the only labeled edges associated with links in $\mathcal{A}_U$ that can appear in $\mathcal{P}_{\text{va}}^U$ are UC or wait edges. But then for *any* ω_o that is the same as ω^m except possibly for durations of links in $\mathcal{A}_U$, $-v \geq |\mathcal{P}_{\text{va}}^U|_{\omega^m} = |\mathcal{P}_{\text{vu}}^U|_{\omega^m} - u \geq |\mathcal{P}_{\text{vu}}^U|_{\omega_o} - u$. Since the durations of all *other* contingent links were arbitrary (except for $C - A = y$), we get $-v \geq d^*(V,U) - u$, a contradiction. So E_U can't be the last edge of an SVP from V to U in ω_o.

Finally, let $\mathcal{A}$ be the union of the sets $\mathcal{A}_U$ associated with all wait edges $(U,C{:}{-u},A)$ that terminate SVPs from V to A in some situations; let ω^M be *any* situation where $C_i - A_i = x_i$ for all of the links in $\mathcal{A}$, but $C - A = y$; and let $\mathcal{P}^\star$ be any simple SVP from V to A in ω^M. By the preceding argument, *none* of the wait edges E_U can be the terminal edge of $\mathcal{P}^\star$; $\mathcal{P}^\star$ cannot contain *any* edges labeled by C or c; and $-v \geq |\mathcal{P}^\star|_{\omega^M}$. But since the only kind of labeled edges associated with links in $\mathcal{A}$ that can appear in $\mathcal{P}^\star$ are UC or wait edges, it follows that in *any* situation $\omega^\dagger$ that is the same as ω^M except possibly for durations of links in $\mathcal{A}$, $-v \geq |\mathcal{P}^\star|_{\omega^M} \geq |\mathcal{P}^\star|_{\omega^\dagger}$. And since the durations of all other links were arbitrary, and $\mathcal{P}^\star$ cannot contain any edges labeled by C or c, we get: $-v \geq d^*(V,A)$, a contradiction. □

Complexity. The time complexity of *minDisp*$_{\text{ESTNU}}$ is dominated by the (at most) $k+1$ calls to Johnson's algorithm whose complexity is $O(mn + n^2 \log n)$, where $m \leq n^2$ is the maximum number of edges in $\mathcal{G}_o$. Therefore, the worst-case complexity of *minDisp*$_{\text{ESTNU}}$ is $O(kn^3)$.

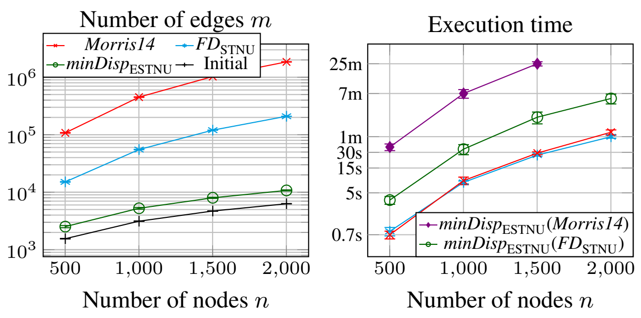


Figure 9: *minDisp*$_{\text{ESTNU}}$ performance vs. network size

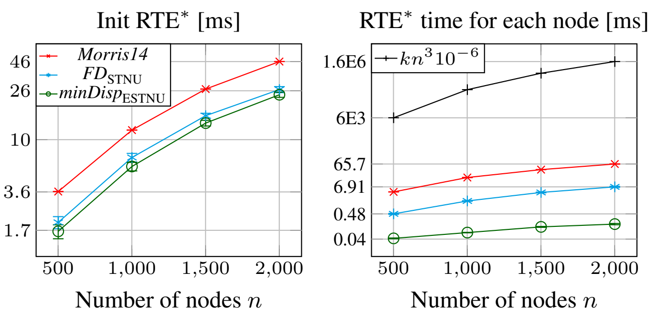


Figure 10: RTE* performance vs. network size

Empirical Evaluation

We evaluated the performance of *minDisp*$_{\text{ESTNU}}$ against *Morris14* and *FD*$_{\text{STNU}}$, aiming to demonstrate: (1) the increase in computational cost required to generate a μESTNU, and (2) the reduction in the number of edges in the μESTNU.

Morris14 is the DC-checking algorithm proposed by Morris (2014), which we modified according to his high-level description to enable it to generate equivalent dispatchable networks. *FD*$_{\text{STNU}}$ is the fast dispatchable STNU algorithm presented by Hunsberger and Posenato (2023).

We implemented all algorithms in Java, available in version 4.12 of the CSTNU Tool of Posenato (2022), and ran them on a JVM 21 with 8 GB of heap memory on a Linux computer with two AMD Opteron™ 4334@3.1 GHz (6200 BogoMIPS).

For our evaluation, we used the benchmark published by Posenato (2020). For each $n \in \{500, 1000, 1500, 2000\}$, the benchmark contains 30 randomly generated DC STNUs, each having n nodes, $n/10$ contingent links, and about six incident edges per node (for a total of $m \approx 3n$ edges).

Each DC STNU was separately fed to *Morris14* and *FD*$_{\text{STNU}}$ to generate a pair of equivalent dispatchable ESTNUs. Both were then fed to the *minDisp*$_{\text{ESTNU}}$ algorithm to check that the resulting μESTNUs were the same. In the large majority of cases, they were, but for a few instances, the two minimized ESTNUs had some different edges. We discovered that simultaneity constraints among pairs of timepoints can result in trivially different but equivalent minimized ESTNUs.

The diagram on the left side of Figure 9 plots the average numbers of edges in the original STNU (black); the dispatchable ESTNUs generated by *Morris14* (red) and *FD*$_{\text{STNU}}$ (blue); and the minimal dispatchable ESTNU generated by *minDisp*$_{\text{ESTNU}}$ (green). The error bars show 95% confidence intervals (difficult to see because the standard deviations are small). The average number of edges in the minimized networks is *about two orders of magnitude smaller* than in the ESTNUs generated by *Morris14*, and *about one order of magnitude smaller* than in the ESTNUs generated by *FD*$_{\text{STNU}}$, confirming the importance of *minDisp*$_{\text{ESTNU}}$ for providing dispatchable networks that can be more efficiently executed.

The righthand plot of Figure 9 shows the computational cost of generating minimal dispatchable ESTNUs. The results highlight the influence of the number of edges on the computing time, which is $O(kn^3)$ in the worst case. When the input instances are generated by *Morris14*, the average execution time of *minDisp*$_{\text{ESTNU}}$ (violet line) is an order of magnitude higher than when the input instances are generated by *FD*$_{\text{STNU}}$ (green line). (For $n = 2000$, the execution time of *minDisp*$_{\text{ESTNU}}$(*Morris14*) was above the 30-minute timeout.) This difference is similar to the difference in the average numbers of edges between instances generated by *Morris14* and *FD*$_{\text{STNU}}$ shown in the lefthand plot.

RTE* is a real-time executor that can execute dispatchable ESTNUs efficiently (Hunsberger and Posenato 2024); an open-source Java implementation is available in CSTNU Tool v. 1.42 (Posenato 2022). Figure 10 demonstrates the impact of the number of edges on the performance of the ESTNU executor RTE*. For each benchmark instance, equivalent dispatchable ESTNUs were generated by *Morris14*, *FD*$_{\text{STNU}}$, and *minDisp*$_{\text{ESTNU}}$*FD*$_{\text{STNU}}$. We configured RTE* to schedule timepoints using an early execution strategy for controllable timepoints and set up the environment to return the middle value for each contingent duration. The lefthand plot of Figure 10 shows the average execution time needed to initialize the executor. The time depends in part on the number of edges in the network. The execution time when the input is from *Morris14* is around double that when input is from *minDisp*$_{\text{ESTNU}}$*FD*$_{\text{STNU}}$. In contrast, the number of edges of the *Morris14* instances is around two orders of magnitude greater than the number of edges of the *minDisp*$_{\text{ESTNU}}$*FD*$_{\text{STNU}}$ ones. This is because the time necessary to initialize some general data structures dominates the time to initialize the data structure relative to the edges. The righthand plot of Figure 10 shows the average time required by RTE* to schedule a single timepoint or to manage the occurrence of a contingent one. On average, the time RTE* takes to a value to a timepoint of a *minDisp*$_{\text{ESTNU}}$*FD*$_{\text{STNU}}$ instance is smaller by two orders of magnitude than the time it takes to assign a value to a timepoint of a *Morris14* instance.

Conclusions

This paper presented a new algorithm for generating equivalent dispatchable ESTNUs with a minimal number of edges. The empirical evaluation demonstrated its effectiveness both in reducing the number of edges and in improving the performance during real-time execution. The formal analysis proved the algorithm's correctness.

References

Cairo, M.; Hunsberger, L.; and Rizzi, R. 2018. Faster Dynamic Controllablity Checking for Simple Temporal Networks with Uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME-2018)*, volume 120 of *LIPIcs*, 8:1–8:16.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2022. *Introduction to Algorithms, 4th Edition*. MIT Press. ISBN 978-0-262-04630-5.

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

Hunsberger, L.; and Posenato, R. 2022. Speeding up the RUL$^-$ Dynamic-Controllability-Checking Algorithm for Simple Temporal Networks with Uncertainty. In *36th AAAI Conference on Artificial Intelligence (AAAI-22)*, volume 36-9, 9776–9785. AAAI Pres.

Hunsberger, L.; and Posenato, R. 2023. A Faster Algorithm for Converting Simple Temporal Networks with Uncertainty into Dispatchable Form. *Information and Computation*, 293(105063): 1–21.

Hunsberger, L.; and Posenato, R. 2024. Foundations of Dispatchability for Simple Temporal Networks with Uncertainty. In *16th International Conference on Agents and Artificial Intelligence (ICAART 2024)*, volume 2, 253–263. SCITEPRESS. ISBN 978-989-758-680-4.

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 *Int. Conf. on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR-2014)*, volume 8451 of *LNCS*, 464–479. Springer.

Morris, P. 2016. The Mathematics of Dispatchability Revisited. In *26th International Conference on Automated Planning and Scheduling (ICAPS-2016)*, 244–252.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001)*, volume 1, 494–499.

Morris, P. H.; and Muscettola, N. 2005. Temporal Dynamic Controllability Revisited. In *20th National Conference on Artificial Intelligence (AAAI-2005)*, 1193–1198.

Nilsson, M.; Kvarnstrom, J.; and Doherty, P. 2014. EfficientIDC: A faster incremental dynamic controllability algorithm. In *24th International Conference on Automated Planning and Scheduling (ICAPS-14)*, 199–207.

Posenato, R. 2020. STNU Benchmark version 2020. Last access 2022-12-01.

Posenato, R. 2022. CSTNU Tool: A Java library for checking temporal networks. *SoftwareX*, 17: 100905.

Stedl, J.; and Williams, B. 2005. A fast incremental dynamic controllability algorithm. In *ICAPS Workshop on Plan Execution: A Reality Check*, 69–75.

Tsamardinos, I.; Muscettola, N.; and Morris, P. 1998. Fast Transformation of Temporal Plans for Efficient Execution. In *15th National Conf. on Artificial Intelligence (AAAI-1998)*, 254–261.