Acta Informatica (2016) 53:89–147
DOI 10.1007/s00236-015-0227-0

ORIGINAL ARTICLE

Efficient execution of dynamically controllable simple temporal networks with uncertainty

Luke Hunsberger[1]

Received: 10 February 2014 / Accepted: 18 March 2015 / Published online: 1 May 2015

Abstract A simple temporal network with uncertainty (STNU) is a data structure for representing and reasoning about temporal constraints where the durations of certain temporal intervals—the contingent links—are only discovered during execution. The most important property of an STNU is whether it is dynamically controllable (DC)—that is, whether there exists a strategy for executing its time-points that will guarantee that all of its constraints will be satisfied no matter how the durations of the contingent links turn out. The literature on STNUs includes a variety of DC-checking algorithms and execution algorithms. The fastest DC-checking algorithm reported so far is the $O(N^3)$-time algorithm due to Morris (Integration of AI and OR techniques in constraint programming—11th international conference, CPAIOR 2014, volume 8451 of Lecture Notes in Computer Science. Springer, Berlin, pp 464–479, 2014). The fastest *execution algorithm* for dynamically controllable STNUs is the $O(N^3)$-time algorithm due to Hunsberger (Proceedings of the 20th international symposium on temporal representation and reasoning (TIME-2013). IEEE Computer Society, Washington, 2013). This paper begins by providing the first comprehensive, rigorous, and yet streamlined treatment of the theoretical foundations of STNUs, including execution semantics, dynamic controllability, and a set of results that have been collected into what has recently been called the fundamental theorem of STNUs. The paper carefully argues from basic definitions to proofs of the major theorems on which all of the important algorithmic work on STNUs depends. Although many parts of this presentation have appeared in various forms, in various papers, the scattered nature of the STNU literature has allowed too many holes in the theory to persist, and has relied all too often on proof sketches that leave important details unexamined. The presentation combines results from many sources, while also introducing novel approaches and proofs. The paper concludes by presenting a modified version of a recent algorithm for managing the execution of dynamically controllable STNUs, the fastest reported so far in the literature. The modified version organizes its computations more efficiently and corrects an oversight in the original algorithm.

✉ Luke Hunsberger
hunsberg@cs.vassar.edu

[1] Computer Science Department, Vassar College, Poughkeepsie, NY, USA

1 Introduction

A *simple temporal network with uncertainty* (STNU) is a data structure for representing and reasoning about temporal constraints in scenarios where the durations of some temporal intervals are beyond the control of the planning agent (or executor) [18]. The most important property of an STNU is whether it is *dynamically controllable*—that is, whether there exists a strategy for executing the time-points in the network such that all constraints are guaranteed to be satisfied no matter how the uncontrollable durations turn out over time. Algorithms for determining whether arbitrary STNUs are dynamically controllable are called DC-checking algorithms.

In 2001, Morris, Muscettola and Vidal [18] presented the most widely used semantics for dynamic controllability, along with a *pseudo-polynomial* DC-checking algorithm, hereinafter called the MMV-01 algorithm. However, the completeness proof for their algorithm depended on an informally sketched dynamic strategy for generating execution decisions.

In 2005, Morris and Muscettola [19] presented the first truly polynomial DC-checking algorithm, an $O(N^5)$-time algorithm, hereinafter called the MM-05 algorithm. (N is the number of time-points in the network.) Their algorithm demonstrated that edge generation (equivalently, constraint propagation) for STNUs could be bounded; however, they assumed—without proof—that *non-shortest* labeled edges in an STNU could be discarded, which turns out to be far from trivial to prove. In addition, the completeness proof for the MM-05 algorithm depended on the completeness of the earlier, MMV-01 algorithm.

In 2006, Morris [16] presented a new approach to analyzing the graphical properties of STNUs that made substantial contributions to the theoretical foundations of STNUs while also enabling a faster, $O(N^4)$-time DC-checking algorithm, hereinafter called the M-06 algorithm. However, Morris used a non-standard execution semantics that allows a form of instantaneous reactivity. As in the MM-05 work, the proof of the main result—that an STNU is DC if and only if its graph has no *semi-reducible negative loops* (cf. Sect. 2.4)—relied on the completeness of the earlier, MMV-01 algorithm.

In 2009, this author discovered a minor technical flaw in the semantics for dynamic controllability (cf. the discussion of *prehistories* in Sect. 2.3) that opened the door to a counter-example [10]. Fixing that flaw vaporized the counter-example, while also enabling a more practical characterization of *dynamic execution strategies* in terms of *real-time execution decisions.* This author also presented the first polynomial algorithms for managing the execution of dynamically controllable networks, while also translating Morris' definitions and techniques into a form applicable to the standard execution semantics [11,12].

Meanwhile, in 2005, Stedl and Williams [28] introduced an *incremental* version of the DC-checking problem. They defined the *Incremental Dynamic Controllability* (IDC) problem as that of determining whether an incremental change (e.g., adding a new constraint) to a dynamically controllable STNU preserves its dynamic controllability. They presented an algorithm, called FastIDC, for solving the IDC problem, but did not prove its completeness.[1]

In 2013, Nilsson et al. [22] showed that the FastIDC algorithm was, in fact, not complete by presenting a counter-example. They fixed the problem and proved that the modified algorithm was complete by showing that it was equivalent to the MMV-01 algorithm. They subsequently made further improvements to the FastIDC algorithm yielding an $O(N^4)$-time algorithm called EIDC [24], and an $O(N^3)$-time algorithm called EIDC2 [25].

[1] They also highlighted interesting connections between the *dispatchability* of simple temporal networks (STNs) [21,29] and the dynamic controllability of STNUs, but did not formally define dispatchability for STNUs. Similar remarks apply to a follow-up paper [27].

Finally, in 2014, Morris [17] presented the fastest DC-checking algorithm reported so far in the literature, an $O(N^3)$-time algorithm, hereinafter called the M-14 algorithm. The M-14 algorithm solves the full DC-checking problem in the same worst-case time in which the EIDC2 algorithm solves the incremental (IDC) problem.[2] However, as with all of its predecessors, the completeness proof for the M-14 algorithm ultimately depends on the completeness of the original MMV-01 algorithm.

As the preceding overview suggests, there have been numerous incremental and substantial contributions to the theoretical foundations of STNUs, which have led to important advances in DC-checking algorithms. However, these contributions have never before been composed into a comprehensive and rigorous treatment that connects important results into a coherent framework, discarding unnecessary diversions, while filling in important details. The presentation in this paper provides such a treatment.

Structure of the rest of the paper Section 2 provides the relevant background on STNUs and dynamic controllability. It begins with an overview of simple temporal networks. It then defines STNUs and, for completeness, presents both the MMV-01 semantics for dynamic controllability and the equivalent semantics based on real-time execution decisions. It continues with STNU graphs and the commonly used edge-generation (constraint-propagation) rules from Morris and Muscettola [19], before defining the important concept of a semi-reducible negative loop (i.e., SRN loop).

Section 3 tackles the first part of what has recently been called the *fundamental theorem of STNUs* [13]: that an STNU whose graph has an SRN loop must *not* be dynamically controllable. It first formally defines what it means for an execution strategy to *satisfy* each kind of edge in an STNU graph. It then defines a notion of *soundness* for the edge-generation rules. Finally, it proves that each of the edge-generation rules is sound.

Section 4 addresses the second part of the fundamental theorem: that an STNU with no SRN loops must be dynamically controllable. It first carefully defines *edge-generation rounds,* and then proves that if an STNU has no SRN loops, then non-shortest edges can be discarded during edge-generation, something that is commonly assumed in the literature without proof, although it turns out to be surprisingly tricky, requiring a novel application of techniques from Morris' graphical analysis of STNUs [16]. It also shows that the notion of shortest semi-reducible paths is well defined for STNUs that do not have any SRN loops. Finally, it introduces an execution strategy that generates decisions based on incrementally updated STNU graphs and the *all-pairs, shortest-semi-reducible-paths* (APSSRP) matrix, and proves that for any STNU having no SRN loops, following that strategy will guarantee that all constraints in the network will necessarily be satisfied no matter how the uncontrollable durations turn out.

Section 5 then presents a modified version of the FAST-EX execution algorithm for dynamically controllable STNUs. It is the fastest execution algorithm reported in the literature so far: at most N updates at $O(N^2)$ time per update, for a total computation time of $O(N^3)$ time. The modifications include streamlining the constraints that are added to the network, efficiently interleaving updates, and correcting an error of omission in the original presentation of the algorithm [12].

Section 6 presents conclusions and discusses the many avenues for future work on STNUs and dynamic controllability.

[2] Nilsson et al. [25] have observed that the M-14 and EIDC2 algorithms, although derived independently, employ many similar techniques. They conjecture that the algorithms are, in fact, equivalent.

2 Background

Temporal networks are data structures for representing and reasoning about temporal constraints on activities. The most basic kind of temporal network is a Simple Temporal Network (STN) which can accommodate such constraints as release times, deadlines, precedence constraints, and duration constraints [5]. The fundamental computational tasks associated with STNs—checking consistency and managing execution—can be done in polynomial time [5,9,29]. A simple temporal network with uncertainty (STNU) augments an STN to include *contingent links* that can be used to represent actions with uncertain durations [18]. Despite the increase in expressiveness, the analogous computational tasks—checking *dynamic controllability* and managing execution—can still be done in polynomial time [11,16,19]. This section summarizes relevant background information from the literature on STNs and STNUs that will be used in the rest of the paper.

2.1 Simple temporal networks

Definition 1 (*STN* [5]) A *Simple Temporal Network* (STN) is a pair, $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a finite and non-empty set of real-valued variables called *time-points* and $\mathcal{C}$ is a set of binary constraints, each having the form, $Y - X \leq \delta$, for some $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$.

A *solution* for an STN is a set of values for its time-points that together satisfy all of its constraints. An STN that has a solution is called *consistent*. The problem of determining whether an STN is consistent is often called the *Simple Temporal Problem* (STP) [5]. Thus, the STP is a kind of *constraint satisfaction problem* [4].

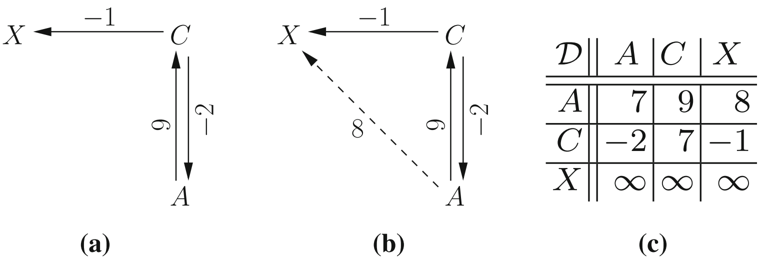
For example, consider the STN defined by:

$$\mathcal{T} = \{A, C, X\} \quad \text{and} \quad \mathcal{C} = \{C - A \leq 9,\ A - C \leq -2,\ X - C \leq -1\}.$$

It is consistent because it has a solution—for example: $\{A = 0, X = 5, C = 7\}$.

Definition 2 (*STN graph* [5]) The graph for an STN, $(\mathcal{T}, \mathcal{C})$, is a pair, $\langle \mathcal{T}, \mathcal{E} \rangle$, where the time-points in $\mathcal{T}$ serve as the nodes in the graph, and the constraints in $\mathcal{C}$ correspond one-to-one to the edges in $\mathcal{E}$. In particular, each constraint, $Y - X \leq \delta$ in $\mathcal{C}$, corresponds to an edge, $X \xrightarrow{\delta} Y$ in $\mathcal{E}$.

Figure 1a shows the graph for the sample STN seen above. To facilitate subsequent contrast with STNUs, constraints and edges in an STN are called *ordinary* constraints and edges.



$\mathcal{D}$	A	C	X
A	7	9	8
C	−2	7	−1
X	∞	∞	∞

Fig. 1 **a** A sample STN graph. **b** The same graph with a new edge. **c** The distance matrix, $\mathcal{D}$

Paths in an STN graph Paths in an STN graph correspond to constraints that must be satisfied by any solution for the associated STN. In general, if $\mathcal{P}$ is a path from X to Y of length ℓ, then the constraint, $Y - X \leq \ell$, must be satisfied by any solution. For example, the path from A to C to X of length 8 in Fig. 1a corresponds to the constraint, $X - A \leq 8$. This constraint is made explicit by the dashed edge from A to X in Fig. 1b. It is easy to check that this constraint is satisfied by the solution given earlier. Since shorter paths correspond to stronger constraints, the *all-pairs, shortest-paths* (APSP) matrix for an STN graph plays an important role in the theory of STNs.

Definition 3 (*Distance matrix* [5]) The *distance matrix* for an STN is the all-pairs, shortest-paths matrix, $\mathcal{D}$, for the associated STN graph.

For any time-points X and Y, $\mathcal{D}(X, Y)$ equals the length of the shortest path from X to Y in the STN graph. Thus, the constraint, $Y - X \leq \mathcal{D}(X, Y)$, must be satisfied by any solution for the STN. Figure 1c shows the distance matrix for the STN from Fig. 1a.

The following theorem specifies the important relationships between an STN, its graph, and its distance matrix [5]. For this reason, it has recently been called the *fundamental theorem of STNs* [13].

Theorem 1 (Fundamental theorem of STNs [5,13]) *For any STN $\mathcal{S}$, with graph $\mathcal{G}$, and distance matrix $\mathcal{D}$, the following are equivalent: (1) $\mathcal{S}$ is consistent; (2) every loop in $\mathcal{G}$ has non-negative length; (3) the notion of shortest path is well defined for $\mathcal{G}$; and (4) $\mathcal{D}$ has only non-negative values down its main diagonal.*[3]

The zero time-point In many applications, it is useful to include a special reference time-point, Z, whose value is fixed at 0. Z is frequently called the *zero time-point.* Binary constraints involving Z are equivalent to unary constraints. For example, given that $Z = 0$, the constraints, $Y - Z \leq b$ and $Z - Y \leq -a$, are respectively equivalent to the unary constraints, $Y \leq b$ and $a \leq Y$, which in turn can be abbreviated as $Y \in [a, b]$.

Given the observations about shortest paths and the definition of the distance matrix $\mathcal{D}$, it follows that for any X, the constraints, $X - Z \leq \mathcal{D}(Z, X)$ and $Z - X \leq \mathcal{D}(X, Z)$, must be satisfied (i.e., $X \in [-\mathcal{D}(X, Z),\ \mathcal{D}(Z, X)]$). The interval $[-\mathcal{D}(X, Z),\ \mathcal{D}(Z, X)]$ is often called the *execution window* for X. To *execute* X at some time t means to assign the value t to X, which can be explicitly represented in the network by adding the constraints, $X - Z \leq t$ and $Z - X \leq -t$ (i.e., $X = t$). This is equivalent to inserting the edges, $Z \xrightarrow{t} X$ and $X \xrightarrow{-t} Z$, into the STN graph. Once executed, the value of X is fixed.

Although most STNs will already have a zero time-point to accommodate unary constraints such as deadlines or release times, the following lemma ensures that each *consistent* STN has a time-point that can play the role of Z; it also shows how to find such a time-point.

Lemma 1 *Let $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ be a consistent STN. For each time-point $X \in \mathcal{T}$, let $\mu_X = \min\{\mathcal{D}(X, X_i) \mid X_i \in \mathcal{T}\}$; and let $\mu = \max\{\mu_X \mid X \in \mathcal{T}\}$. Then $\mu \geq 0$. In addition, if X_0 is any time-point for which $\mu_{X_0} = \mu$, then constraining X_0 to equal 0, and constraining every other time-point to occur at or after X_0 will not affect the consistency of the network.*

Proof Suppose that $\mu < 0$. Let X_1 be any time-point in $\mathcal{T}$. Since $\mu_{X_1} \leq \mu < 0$, there must be some $X_2 \in \mathcal{T}$, such that $\mathcal{D}(X_1, X_2) < 0$. Similarly, there must be some $X_3 \in \mathcal{T}$ such that

[3] Many researchers prefer to automatically include trivial *self-loops* of length 0 at each time-point in an STN, corresponding to constraints of the form, $X - X \leq 0$, in which case, the diagonal entries of a consistent distance matrix will all necessarily be zero, instead of being merely non-negative.

$\mathcal{D}(X_2, X_3) < 0$. And so on. Since there are only finitely many time-points in $\mathcal{T}$, this process must eventually yield a sequence of shortest paths that, when strung together, form a loop of negative length, contradicting the fundamental theorem. Thus, $\mu \geq 0$.

Next, let X_0 be any time-point for which $\mu_{X_0} = \mu$. For each X_i, it follows that $\mathcal{D}(X_0, X_i) \geq \mu_{X_0} = \mu \geq 0$. That is, the shortest path from X_0 to X_i has non-negative length. Thus, constraining X_i to occur at or after X_0—which is equivalent to inserting the edge, $X_i \xrightarrow{0} X_0$, into the graph)—cannot introduce a negative loop. Thus, the resulting network must still be consistent.

Now, adding that edge, which *terminates* at X_0, cannot affect the lengths of any shortest paths *emanating* from X_0 (cf. Fact 1 and Corollary 1, below); and hence cannot change any of the $\mathcal{D}(X_0, X_j)$ values. In this way, one after the other, every X_j can be constrained to occur at or after X_0 without introducing any negative loops. Thus, the resulting network, $\mathcal{S}' = (\mathcal{T},\ \mathcal{C} \cup \{X_0 - X_i \leq 0 \mid X_i \in \mathcal{T}\})$, must be consistent. Thus, there must a set of variable assignments, Σ, that is a solution for $\mathcal{S}'$. Let v be the value assigned to X_0 by Σ; and let Σ' be the same as Σ except that each value is decremented by v. Since all constraints in the network are binary difference constraints, Σ' must also be a solution for $\mathcal{S}'$; and in that solution, X_0 has the value 0. □

The following elementary results are used not only in the above proof, but also in several places in the rest of the paper.

Fact 1 *Suppose that U and V are distinct time-points in an STN, and that $\mathcal{P}$ is a shortest path from U to V. Then there exists a shortest path, $\mathcal{P}'$, from U to V that does not include any edges of the form, $V \xrightarrow{\delta} Y$, for any time-point Y.*

Proof Suppose that $\mathcal{P}$ is a shortest path from U to V that includes an edge of the form, $V \xrightarrow{\delta} Y$. Let E be the first such edge in $\mathcal{P}$. Now, by construction, the suffix of $\mathcal{P}$ whose first edge is E must be a loop, $\mathcal{P}_L$, from V back to V. If $\mathcal{P}_L$ had negative length, then splicing an extra copy of $\mathcal{P}_L$ into $\mathcal{P}$ would generate a shorter path from U to V, contradicting that $\mathcal{P}$ is a shortest path. But if $\mathcal{P}_L$ had positive length, then extracting it from $\mathcal{P}$ would similarly lead to a shorter path, and thus a contradiction. Therefore, $\mathcal{P}_L$ must have length 0. Let $\mathcal{P}'$ be the path from U to V obtained by extracting the loop $\mathcal{P}_L$ from $\mathcal{P}$. Since $\mathcal{P}_L$ has length 0, the length of $\mathcal{P}'$ is the same as that of $\mathcal{P}$. Thus, $\mathcal{P}'$ is a shortest path from U to V. Furthermore, since U and V are distinct time-points, $\mathcal{P}'$ contains at least one edge. □

Corollary 1 *Suppose $\mathcal{S}$ is a consistent STN, and $\mathcal{G}$ its graph. If inserting the edge, $V \xrightarrow{\delta} Y$, into $\mathcal{G}$ preserves its consistency, then doing so cannot affect the lengths of any shortest paths terminating at V.*

2.2 Simple temporal networks with uncertainty

Consider the following example of taking a taxi to the airport. Suppose that I want to arrive at the airport sometime between 9:45 and 10:00, but that I do not control the duration of the taxi ride, which may be anywhere from 15 to 25 min. It is not hard to verify that I can ensure that my arrival falls between 9:45 and 10:00 simply by making sure that I get into the taxi sometime between 9:30 and 9:35. That is, although I only directly control the time at which I get into the taxi, I can nonetheless satisfy constraints involving a time I do not directly control (i.e., the time I arrive at the airport). Although this example is quite simple, the uncertain duration of the taxi ride cannot be properly represented within an STN.

A STNU augments an STN to include a set of *contingent links*, where each contingent link represents a bounded temporal interval whose duration is not controlled by the planning agent and is thus uncertain [18].[4] Contingent links are typically used to represent actions whose durations are uncertain—like the taxi ride in the above example. The agent may control when an action starts, but not how long it will take to complete; the agent only discovers the actual duration in real time. The most important property of an STNU is whether it is *dynamically controllable*—that is, whether there exists a strategy for executing the controllable time-points such that all constraints will be satisfied no matter how the durations of the contingent links turn out in real time—within their specified bounds. As will be seen, the dynamic controllability of STNUs can be determined in polynomial time.

Definition 4 (*STNU* [18]) A STNU is a triple, $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, where $(\mathcal{T}, \mathcal{C})$ is an STN, and $\mathcal{L}$ is a set of contingent links.[5] Each contingent link has the form, (A, x, y, C), where $A, C \in \mathcal{T}$, and $0 < x < y < \infty$. In addition, if (A_1, x_1, y_1, C_1) and (A_2, x_2, y_2, C_2) are distinct contingent links, then C_1 and C_2 must be distinct time-points in $\mathcal{T}$.

For any contingent link (A, x, y, C), A and C are called its *activation* and *contingent* time-points, respectively; and x and y respectively specify lower and upper bounds on the link's duration, $C - A$. In most cases, an agent will control the execution of A, but not C. Instead, C can be thought of as being controlled by the *environment.* The execution semantics, defined later on, will ensure that, from the agent's perspective, the duration, $C - A$, is uncontrollable, but nonetheless guaranteed to lie within the interval, $[x, y]$. Thus, the agent may be able to indirectly exert some control over C through its control over A.

For example, consider the STNU, $\mathcal{S}^\dagger = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, where:

$$\begin{aligned}
\mathcal{T} &= \{X, A_1, C_1, A_2, C_2\};\\
\mathcal{C} &= \{C_1 - C_2 \leq 2,\ X - C_1 \leq -1\};\ \text{and}\\
\mathcal{L} &= \{(A_1, 2, 9, C_1), (A_2, 3, 7, C_2)\}.
\end{aligned}$$

This STNU will be used as a running example.

Contingent versus executable time-points Any time-point in an STNU that is not a contingent time-point for some contingent link is said to be *executable.* Thus, the time-points in an STNU are partitioned into two sets: $\mathcal{T}_c$, the contingent time-points; and $\mathcal{T}_{ex}$, the executable time-points. For example, for the STNU $\mathcal{S}^\dagger$, $\mathcal{T}_c = \{C_1, C_2\}$ and $\mathcal{T}_{ex} = \{A_1, A_2, X\}$. The agent is presumed to directly control the execution of the executable time-points, but not the contingent time-points. For example, with regard to the contingent link $(A_1, 2, 9, C_1)$, the agent directly controls only the execution of A_1. Once A_1 has been executed (i.e., once the contingent link has been *activated),* the execution of C_1 is out of the agent's control. Although C_1 is guaranteed to be executed such that $C_1 - A_1 \in [2, 9]$, the agent does not get to choose the particular time, but only *observes* the execution of C_1 when it happens. Similar remarks apply to a chain (or even a tree) of contingent links with an executable time-point at its root.[6]

[4] Agents are not part of the semantics of STNUs. They are used here for expository convenience.

[5] The notation presented in Definition 4 is equivalent to the original notation introduced by Morris et al. [18]. It is used in this paper primarily to facilitate access to the constituents of the contingent links—for example, (A, x, y, C) instead of the more cumbersome $(\mathit{start}(e), \ell(e), u(e), \mathit{finish}(e))$.

[6] Note that there is no prohibition against a contingent time-point C for one contingent link serving as the activation time-point for some other contingent link. In this way, contingent links may form chains or trees.

2.3 The semantics of execution and dynamic controllability

As will be seen, an STNU is *dynamically controllable* (DC) if there exists a dynamic strategy for executing the executable time-points that guarantees that all constraints will be satisfied no matter how the durations of the contingent links turn out—within their specified bounds. Crucially, the decisions constituting a *dynamic execution strategy* can depend only on *past* execution events, including past observations of contingent durations. Thus, the semantics of dynamic controllability is necessarily related to the semantics of execution in an STNU.

In 2001, Morris, Muscettola and Vidal introduced the most widely used semantics for dynamic controllability, hereinafter called the MMV-01 semantics [18]. Although widely cited, its definitions do not clearly articulate some important concepts. For example, execution decisions are not explicitly defined; instead, they are only implicitly captured by the definitions of *execution strategies* and *prehistories.* Perhaps for this reason, an error in the MMV-01 specification involving the crucial requirement that execution decisions can depend only on *already past* execution events contained an error that was not caught until 2009 (cf. the discussion of *prehistories* in Sect. 2.3.1) [10]. Fixing that flaw not only enabled the semantics to properly capture the desired requirement, it also enabled an equivalent characterization of dynamic controllability in terms of explicit *real-time execution decisions* (RTEDs). Unlike the MMV-01 semantics, the RTED-based semantics focuses squarely on the evolution of execution events over time. Insodoing, it clearly distinguishes the execution semantics from the definition of dynamic controllability. For these reasons, the theoretical contributions of this paper are expressed in terms of the RTED-based semantics. Nonetheless, for the sake of comparison, the (corrected) MMV-01 semantics is summarized first, in Sect. 2.3.1, followed by the RTED-based semantics in Sect. 2.3.2.[7]

2.3.1 The MMV-01 semantics for dynamic controllability

Definition 5 (*Situations* [32]) If $(A_1, x_1, y_1, C_1), \ldots, (A_K, x_K, y_K, C_K)$ are the K contingent links in an STNU $\mathcal{S}$, then the corresponding *space of situations for* $\mathcal{S}$, denoted by Ω, is given by: $\Omega = [x_1, y_1] \times \cdots \times [x_K, y_K]$.[8]

Each *situation,* $\omega = (\omega_1, \ldots, \omega_K) \in \Omega$, represents one possible (complete) set of values for the durations of the contingent links in a given STNU. For example, the space of situations for the sample STNU $\mathcal{S}^\dagger$ is $[2, 9] \times [3, 7]$; and $(3, 6)$ is the situation where $C_1 - A_1 = 3$ and $C_2 - A_2 = 6$. The particular situation that actually obtains is, of course, not known in advance, but only gradually discovered, over time, as each contingent time-point executes.

Definition 6 (*Projection* [18,32]) The *projection* of an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ onto a situation ω is defined to be the STN—not STNU—that results from, in effect, forcing each contingent link to take on the duration specified in ω. Thus, the projection of $\mathcal{S}$ onto ω, denoted by $\mathcal{S}_\omega$, is given by: $\mathcal{S}_\omega = (\mathcal{T},\ \mathcal{C} \cup \{C_i - A_i = \omega_i \mid 1 \leq i \leq K\})$.[9]

For example, the projection of $\mathcal{S}^\dagger$ onto the situation, $(3, 6)$, is the STN $(\mathcal{T}, \mathcal{C}')$ where:

$$\mathcal{T} = \{X, A_1, C_1, A_2, C_2\};\ \text{and}$$
$$\mathcal{C}' = \{C_1 - C_2 \leq 2,\ X - C_1 \leq -1,\ C_1 - A_1 = 3,\ C_2 - A_2 = 6\}$$

[7] The equivalence of the MMV-01 and RTED-based semantics is addressed in detail elsewhere [10].

[8] Context usually makes clear to which STNU a given space of situations applies.

[9] Note that $C_i - A_i = \omega_i$ is shorthand for the pair of constraints, $C_i - A_i \leq \omega_i$ and $A_i - C_i \leq -\omega_i$.

Definition 7 (*Schedule* [18]) A *schedule* for an STNU $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ is a mapping $\xi : \mathcal{T} \longrightarrow \mathbb{R}$.

Notice that a schedule does not distinguish between contingent and executable time-points. For example, $\xi^{\dagger} = \{A_1 \mapsto 0,\ X \mapsto 1,\ C_1 \mapsto 3,\ A_2 \mapsto 3.5,\ C_2 \mapsto 9.5\}$ is a schedule for the sample STNU $\mathcal{S}^{\dagger}$.

For convenience, the *execution time* of X in a schedule ξ may be denoted by either $\xi(X)$ or $[\xi]_X$. In addition, when context allows, the set of all schedules for a given STNU may be denoted simply by Ξ.

Definition 8 (*Execution strategy* [10,18]) An *execution strategy* for an STNU $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ is a mapping $S : \Omega \longrightarrow \Xi$ (i.e., a mapping from situations to schedules).[10]

For example, consider the following informally expressed execution strategy for $\mathcal{S}^{\dagger}$:

- Execute A_1 at time 0.
- Execute X at time 1.
- If C_1 is observed to execute *before* time 4, then execute A_2 at time $(C_1 + 4)/2$; otherwise, execute A_2 at time 4.

The rather tedious task of transforming this informal description into a formal mapping from situations to schedules, as prescribed by Definition 8, is left to the reader.

Definition 9 (*Viable* [18]) An execution strategy S for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is called *viable* if for each situation $\omega \in \Omega$, the schedule $S(\omega)$ is consistent with (i.e., is a solution for) the projection $\mathcal{S}_{\omega}$.

Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be any STNU. Recall that, for any situation ω, the constraints in the projection $\mathcal{S}_{\omega}$ are precisely the constraints from $\mathcal{C}$ together with constraints that the contingent links in $\mathcal{L}$ have the durations specified in ω. Therefore, an execution strategy S for $\mathcal{S}$ is viable if and only if for each situation ω, the execution times specified by the schedule $S(\omega)$ satisfy the constraints in $\mathcal{C}$, while also being consistent with the durations specified by ω.

Regarding the informally expressed strategy for the STNU $\mathcal{S}^{\dagger}$ seen above, it is not hard to verify that in each situation $\omega \in [2, 9] \times [3, 7]$, the schedules generated by this strategy invariably satisfy all of the constraints in $\mathcal{C}$, while also being consistent with the durations in ω. Therefore, the strategy is viable. For example, in the situation $\omega = (3, 6)$, the strategy generates the schedule, $\xi^{\dagger}$, seen earlier. That schedule satisfies the constraints in $\mathcal{C}$ (i.e., $C_1 - C_2 \leq 2$ and $X - C_1 \leq -1$), while also being consistent with the contingent durations in ω (since $C_1 - A_1 = 3$ and $C_2 - A_2 = 6$).

In the MMV-01 semantics, execution strategies are represented as mappings from (complete) situations to (complete) schedules; and execution decisions are not explicitly represented at all. As a result, the crucial requirement that execution decisions can depend only on *past* execution events is specified indirectly, in terms of *prehistories*. Given a schedule ξ, and a time-point (i.e., *variable*) $X \in \mathcal{T}$, the original MMV-01 semantics defined the *prehistory* of X in ξ to be the set of all contingent durations that finish *before* X in ξ. However, defining prehistories in this way turns out to allow execution strategies to bypass the above-mentioned requirement. (See Hunsberger [10] for an in-depth discussion of this important point.) Fortunately, the problem is easily fixed by defining the prehistories of *numbers*, instead of the prehistories of time-points (i.e., variables), as follows.

[10] Morris et al. defined execution strategies as mappings from *projections* to schedules, which is equivalent.

Definition 10 (*Prehistory* [10]) Given a schedule ξ for an STNU $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, and any real number k, the *prehistory* of k in ξ, denoted by $\xi^{<k}$, is given by:

$$\xi^{<k} = \{\langle (A, x, y, C),\ \xi(C) - \xi(A)\rangle \mid (A, x, y, C) \in \mathcal{L} \text{ and } \xi(C) < k\}.$$

For example, given the sample STNU $\mathcal{S}^\dagger$ and the schedule $\xi^\dagger$ discussed above, the following are some of the prehistories associated with $\xi^\dagger$:

$$\begin{aligned}
{\xi^\dagger}^{<1} &= \emptyset \\
{\xi^\dagger}^{<3.5} &= \{\langle (A_1, 2, 9, C_1), 3\rangle\} && \text{since } \xi^\dagger(C_1) = 3 < 3.5 \\
{\xi^\dagger}^{<10} &= \{\langle (A_1, 2, 9, C_1), 3\rangle,\ \langle (A_2, 3, 7, C_2), 6\rangle\} && \text{since } \xi^\dagger(C_2) = 9.5 < 10
\end{aligned}$$

Definition 11 (*Dynamic execution strategy* [10,18]) An execution strategy S for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is a *dynamic execution strategy* (DES) if for any situations, ω' and ω'', and any *executable* time-point $X \in \mathcal{T}_{ex}$, the following condition holds:

$$\text{if } [S(\omega')]_X = k \ \text{ and } S(\omega')^{<k} = S(\omega'')^{<k}, \ \text{ then } [S(\omega'')]_X = k.$$

In other words, if the strategy S in the situation ω' assigns the value k to the executable time-point X, and the prehistories of k in the situations ω' and ω'' are the same, then S must assign the same value k to the time-point X in the situation ω''.

For example, let S be the informally expressed execution strategy for $\mathcal{S}^\dagger$ discussed above. Note that, when following the strategy S, A_2 is the only executable time-point in $\mathcal{S}^\dagger$ whose execution time varies across different situations. In addition, the execution time of A_2 only depends on ω_1 (i.e., the duration of the first contingent link, $(A_1, 2, 9, C_1)$). Consider the situation $\omega' = (3, 6)$. In that situation, A_2 is executed at time $k = 3.5$, and the relevant prehistory is $S(\omega')^{<3.5} = \{\langle (A_1, 2, 9, C_1), 3\rangle\}$ since, in this situation, A_1 was executed at time 0 and C_1 has already been observed to have executed at time $3 < 3.5$. Now, if ω'' is any other situation such that $S(\omega'')^{<3.5} = \{\langle (A_1, 2, 9, C_1), 3\rangle\} = S(\omega')^{<3.5}$, then in the situation ω'' the duration of the first contingent link is 3, but the duration of the second contingent link may be any value in $[3, 7]$. In any case, the schedule $S(\omega'')$ also executes A_2 at time $k = 3.5$ (i.e., $[S(\omega'')]_{A_2} = 3.5$), since the execution time of A_2 only depends on the duration of the first contingent link. Generalizing this argument leads to the conclusion that S is a dynamic execution strategy for $\mathcal{S}^\dagger$.

With the above machinery, the definition of dynamic controllability is now straightforward.

Definition 12 (*Dynamic controllability* [18]) An STNU $\mathcal{S}$ is *dynamically controllable* (DC) if there exists a viable dynamic execution strategy for it.

Since the sample strategy S discussed above is a dynamic execution strategy for $\mathcal{S}^\dagger$, $\mathcal{S}^\dagger$ is dynamically controllable. However, it is quite tedious to apply the semantics directly to confirm this. As will be seen, DC-checking algorithms that manipulate STNU graphs make the problem of checking dynamic controllability much easier.

2.3.2 The RTED-based semantics for dynamic controllability

As discussed in the previous section, the MMV-01 semantics does not clearly articulate such important concepts as execution decisions and the evolution of execution events over time. In contrast, the RTED-based semantics presented in this section: (1) uses *partial schedules* to define the limited information upon which execution decisions may depend; (2) explicitly represents the *real-time execution decisions* (RTEDs) that an execution strategy dynamically generates throughout the execution of an STNU; (3) explicitly models the uncertainty facing an agent in real-time by specifiying the set of situations that are compatible with (or *respected* by) each partial schedule and using them to define the possible *outcomes* for each execution decision; and (4) defines an execution strategy not as a mapping from (complete) situations to (complete) schedules, but as a mapping from partial schedules to real-time execution decisions. In this way, the RTED-based semantics directly models the practical circumstances faced by an agent managing the execution of an STNU in real time.

Definition 13 (*Partial schedule* [10]) Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU. A *partial schedule* for $\mathcal{S}$ is any mapping, $\xi : \mathcal{T}' \longrightarrow \mathbb{R}$, where $\mathcal{T}'$ is a *proper* subset of $\mathcal{T}$.

Each partial schedule specifies the execution times for those time-points that have already executed. As with (complete) schedules (cf. Definition 7), $\xi(X)$ denotes the execution time of X according to ξ. For convenience, a (partial or complete) schedule ξ may be represented by a set of pairs of the form, $(X, \xi(X))$, and the following notation may be used:

$$\begin{aligned}\text{Vars}_\xi &= \mathcal{T}' = \text{ the set of time-points that have already executed according to } \xi.\\ \texttt{now}_\xi &= \max\{\xi(X) \mid X \in \mathcal{T}'\} = \text{ the time of the latest execution event in } \xi.\end{aligned}$$

A partial schedule contains the information upon which the next execution decision may depend. All past execution events occurred at or before $\texttt{now}_\xi$; all subsequent executions will occur strictly after $\texttt{now}_\xi$. For example, the *initial partial schedule* for any STNU is $\xi_0 = \emptyset$, indicating that no time-points have yet been executed. In this case, where $\text{Vars}_{\xi_0} = \emptyset$, the current time is taken to be $\texttt{now}_{\xi_0} = -\infty$. For another example, $\xi = \{(A_1, 0), (X, 1)\}$ is a partial schedule for the sample STNU $\mathcal{S}^\dagger$, representing that the time-points A_1 and X have already executed at times 0 and 1, respectively. For this partial schedule, $\texttt{now}_\xi = 1$. Finally, note that, for this partial schedule, the activation time-point A_1 has already been executed, but the corresponding contingent time-point C_1 has not. In such cases, the contingent time-point C_1 is said to be *currently active* in ξ.

Suppose that ξ is a partial schedule according to which at least one contingent time-point either has not yet been activated or is currently active. Then the information in ξ is not sufficient to determine a unique situation. Instead, the set of situations that are consistent with (or *respected* by) the information in ξ represents the uncertainty facing an agent managing the execution of the STNU in real time.

Definition 14 (*Respect* [10]) A (partial or complete) schedule ξ is said to *respect* a situation $\omega = (\omega_1, \ldots, \omega_K)$ if for each contingent link, (A_i, x_i, y_i, C_i), one of the following holds:

(1) $A_i \notin \text{Vars}_\xi$ and $C_i \notin \text{Vars}_\xi$ (i.e., neither A_i nor C_i have yet been executed);
(2) $A_i \in \text{Vars}_\xi$, $C_i \notin \text{Vars}_\xi$, and $\xi(A_i) + \omega_i > \texttt{now}_\xi$ (i.e., the contingent link has been activated and will complete at some later time); or
(3) $A_i \in \text{Vars}_\xi$, $C_i \in \text{Vars}_\xi$, and $\xi(C_i) - \xi(A_i) = \omega_i$ (i.e., the contingent link has completed, and its duration equals ω_i).

The set of situations that are respected by ξ (i.e., that are consistent with ξ) is notated $\Omega_\xi \subseteq \Omega$. A schedule ξ is called *respectful* if Ω_ξ is non-empty (i.e., if ξ respects at least one situation).

For example, the initial partial schedule, $\xi_0 = \emptyset$, necessarily respects every situation—since condition (1) above holds for every contingent link. Thus, $\Omega_{\xi_0} = \Omega$, and ξ_0 is a respectful partial schedule.

As will be seen, the outcomes of *allowable* execution decisions, defined below, necessarily preserve the property of respect. Thus, the execution semantics ensures that the decisions of an agent cannot affect the environment's choice of duration for any contingent link.

Definition 15 (*Real-time execution decision (RTED)* [10]) Let ξ be a partial schedule for an STNU $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, where $\mathcal{T}_{ex} \subseteq \mathcal{T}$ is the set of executable time-points. A *real-time execution decision* (RTED) for ξ has one of two forms: `wait` or (t, χ). In the latter form, $t \in \mathbb{R}$ and $\chi \subseteq \mathcal{T}_{ex}$.

Definition 16 (*Allowable RTED* [10]) Let ξ be a partial schedule for an STNU $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, where $\mathcal{T}_{ex} \subseteq \mathcal{T}$ is the set of executable time-points. A `wait` RTED is *allowable* for ξ only if *at least one* contingent time-point is currently active in ξ. A (t, χ) RTED is *allowable* for ξ only if $\texttt{now}_\xi < t < \infty$, and χ is a non-empty set of executable time-points that have not yet been executed (i.e., $\emptyset \neq \chi \subseteq \mathcal{T}_{ex}$ and $\chi \cap \text{Vars}_\xi = \emptyset$).

The `wait` RTED represents a decision to simply *wait* for whatever contingent time-point happens to execute next. To avoid waiting forever, the `wait` decision is only allowed for partial schedules where at least one contingent time-point is currently active. For example, `wait` is allowable for the partial schedule, $\xi = \{(A_1, 0), (X, 1)\}$, discussed earlier, since the contingent time-point C_1 is currently active in ξ. However, `wait` is not allowable for the initial partial schedule $\xi_0 = \emptyset$, or the partial schedule $\{(A_1, 0), (X, 1), (C_1, 3)\}$, since neither of these partial schedules has any currently active contingent time-points.

A (t, χ) RTED represents a decision to execute the time-points in χ at some *later* time t. Note that because contingent time-points are controlled by the environment, χ must be a subset of executable time-points. The allowability conditions for a (t, χ) decision can be understood as follows. First, since each time-point can only be executed once, all of the time-points in χ must be *unexecuted* according to ξ. Second, to ensure that something always happens as the result of an execution decision (cf. *outcomes*, discussed below), the set χ must be non-empty. Third, to ensure that execution decisions can depend only on *past* observations, t must be strictly greater than $\texttt{now}_\xi$.[11]

It is useful to interpret a (t, χ) decision as a conditional commitment that can be glossed as: "If nothing happens before time t (i.e., if no contingent time-points happen to execute between `now` and time t), then execute the time-points in χ at time t." For example, given the partial schedule, $\{(A_1, 0), (X, 1)\}$, for which $\texttt{now} = 1$, the decision $(4, \{A_2\})$ is allowable since $\texttt{now} = 1 < 4 = t$, and A_2 is a currently unexecuted executable time-point. The decision $(4, \{A_2\})$ can be interpreted as: "If nothing happens before time 4 (i.e., if the currently active contingent time-point C_1 does not happen to execute before time 4), then execute the executable time-point A_2 at time 4." As will be seen, if C_1 *does* happen to execute before time 4—say, at time 3—then the agent may choose to react—but not instantaneously—say, by deciding to execute A_2 at 3.5.

Outcomes for real-time execution decisions Suppose that ξ is a respectful partial schedule representing the limited information available at a certain point during the execution of an STNU; and that δ is an allowable RTED for ξ. Recall that Ω_ξ (i.e., the set of situations respected by ξ) represents the uncertainty faced by the agent managing the execution of the STNU. As defined below, each situation $\omega \in \Omega_\xi$ determines a potential outcome for the

[11] Allowing t to be equal to $\texttt{now}_\xi$ would enable a form of instantaneous reactivity. For example, an agent might observe the execution of a contingent time-point at time 3, and then instantaneously react by deciding to execute an executable time-point at that same time 3.

decision δ. For example, in one situation the outcome of a $(10, \{X\})$ decision might be that X is executed at time 10, while in another situation the outcome of that same decision might be that a contingent time-point C happens to execute before time 10.

Outcomes for `wait` *decisions* Suppose that `wait` is an allowable RTED for a respectful partial schedule ξ. Since `wait` is allowable, there must be at least one contingent time-point C_i that is currently active in ξ. Equivalently, there must be at least one contingent link (A_i, x_i, y_i, C_i) that, according to ξ, is activated, but not yet completed. Thus, for any situation ω that is respected by ξ, condition (2) from Definition 14 must hold: $\xi(A_i) + \omega_i > \texttt{now}_\xi$. Hence, in the situation ω, this link will complete some time after $\texttt{now}_\xi$. If more than one contingent link is currently active in ξ, then the outcome of the `wait` decision is determined by the earliest time, $\mathrm{ET}(\xi, \omega)$, at which one of the activated contingent links will finish, according to ξ and ω:

$$\mathrm{ET}(\xi, \omega) = \min\{\xi(A_i) + \omega_i : A_i \in \mathrm{Vars}_\xi, C_i \notin \mathrm{Vars}_\xi\}.$$

Since the inequality, $\xi(A_i) + \omega_i > \texttt{now}_\xi$, holds for each currently active contingent link (A_i, x_i, y_i, C_i), it follows that $\mathrm{ET}(\xi, \omega) > \texttt{now}_\xi$.

For example, consider the sample STNU $\mathcal{S}^\dagger$, the partial schedule $\xi_1 = \{(A_1, 0)\}$, and the situation $\omega = (3, 6)$. Note that the contingent link $(A_1, 2, 9, C_1)$ is currently active in ξ_1, and that ξ_1 respects ω. In this case, $\mathrm{ET}(\xi_1, (3, 6)) = 3$, since $A_1 = 0$ in ξ_1, and $C_1 - A_1 = 3$ in ω. Therefore, the outcome of the `wait` decision in this situation would involve the execution of C_1 at time 3. Note that $\texttt{now}_{\xi_1} = 1 < 3 = \mathrm{ET}(\xi_1, (3, 6))$.

In general, it is possible—although rare in practice—for multiple contingent links to complete at the same time. For this reason, Definition 17, below, specifies the *set* $\chi_c(\xi, \omega)$ of contingent time-points that will execute at the time $\mathrm{ET}(\xi, \omega)$, according to ξ and ω.

Definition 17 ($\mathcal{O}(\xi, \omega, \texttt{wait})$ [10]) Let ω be any situation, and ξ any partial schedule that respects ω. If `wait` is an allowable RTED for ξ, then $\mathcal{O}(\xi, \omega, \texttt{wait})$ denotes the *unique* outcome of the `wait` decision, determined by ξ and ω, as follows:

$$\begin{aligned}
&\mathcal{O}(\xi, \omega, \texttt{wait}) = \xi \cup \{(C_i, \mathrm{ET}(\xi, \omega)) \mid C_i \in \chi_c(\xi, \omega)\}, \text{ where:}\\
&\quad \mathrm{ET}(\xi, \omega) = \min\{\xi(A_i) + \omega_i : A_i \in \mathrm{Vars}_\xi, C_i \notin \mathrm{Vars}_\xi\}; \text{ and}\\
&\quad \chi_c(\xi, \omega) = \{C_i \mid A_i \in \mathrm{Vars}_\xi, C_i \notin \mathrm{Vars}_\xi, \text{ and } \xi(A_i) + \omega_i = \mathrm{ET}(\xi, \omega)\}.
\end{aligned}$$

Note that $\mathcal{O}(\xi, \omega, \texttt{wait})$ is a (typically partial) schedule that augments ξ to include the *next* execution event—namely, the simultaneous execution of the contingent time-points in $\chi_c(\xi, \omega)$ at the time $\mathrm{ET}(\xi, \omega)$. As a result, the `now` time for that new schedule will necessarily be equal to $\mathrm{ET}(\xi, \omega)$. In other words, $\texttt{now}_{\mathcal{O}(\xi,\omega,\texttt{wait})} = \mathrm{ET}(\xi, \omega)$. For example, for the sample STNU $\mathcal{S}^\dagger$ and the partial schedule ξ_1 discussed above, the outcome $\mathcal{O}(\xi_1, (3, 6), \texttt{wait})$ is: $\xi_1 \cup \{(C_1, 3)\} = \{(A_1, 0), (C_1, 3)\}$; and $\texttt{now}_{\mathcal{O}(\xi_1,(3,6),\texttt{wait})} = \mathrm{ET}(\xi_1, \texttt{wait}) = 3$.

Outcomes of (t, χ) *decisions* Suppose that (t, χ) is an allowable RTED for a respectful partial schedule ξ, and that ω is a situation respected by ξ (i.e., $\omega \in \Omega_\xi$). Since (t, χ) is allowable, $t > \texttt{now}_\xi$ and χ is a non-empty set of unexecuted executable time-points. The unique outcome of the (t, χ) decision in the situation ω is denoted by $\mathcal{O}(\xi, \omega, (t, \chi))$. That outcome depends on the relationship between the times, t and $\mathrm{ET}(\xi, \omega)$, where t is the time at which the agent has decided to execute the time-points in χ, and $\mathrm{ET}(\xi, \omega)$ is the earliest time at which one or more contingent time-points will execute according to the situation ω, as described above. (Keep in mind that, in general, the agent does not know the situation ω

and, thus, does not know the value of $\mathrm{ET}(\xi, \omega)$.) If $t < \mathrm{ET}(\xi, \omega)$, then the time t arrives before any contingent time-point has a chance to execute; thus, the outcome involves only the execution of the time-points in χ at time t. On the other hand, if $\mathrm{ET}(\xi, \omega) < t$, then one or more contingent time-points will execute before time t arrives; thus, the outcome involves only the execution of the contingent time-points in $\chi_c(\xi, \omega)$ at time $\mathrm{ET}(\xi, \omega)$, as in the case of a `wait` decision. In this case, as will be seen, the agent need not remain committed to executing the time-points in χ at time t, but may choose to react to the newly observed contingent execution(s) by following some alternative execution decision. Finally, if $t = \mathrm{ET}(\xi, \omega)$, which is rarely expected in practice, then the outcome involves the coincidental execution of the time-points in *both* χ and $\chi_c(\xi, \omega)$ at time t.

Definition 18 ($\mathcal{O}(\xi, \omega, (t, \chi))$ [10]) Let ω be any situation, and ξ any partial schedule that respects ω. If (t, χ) is an allowable RTED for ξ, then $\mathcal{O}(\xi, \omega, (t, \chi))$ denotes the unique outcome of the (t, χ) decision, determined by ξ and ω, as follows:

$$\mathcal{O}(\xi, \omega, (t, \chi)) = \begin{cases} \xi \cup \{(X, t) \mid X \in \chi\}, & \text{if } t < \mathrm{ET}(\xi, \omega) \\ \xi \cup \{(C, \mathrm{ET}(\xi, \omega)) \mid C \in \chi_c(\xi, \omega)\}, & \text{if } t > \mathrm{ET}(\xi, \omega) \\ \xi \cup \{(X, t) \mid X \in \chi\} \cup \{(C, t) \mid C \in \chi_c(\xi, \omega)\}, & \text{if } t = \mathrm{ET}(\xi, \omega) \end{cases}$$

For example, recall the sample STNU $\mathcal{S}^\dagger$, the partial schedule $\xi_1 = \{(A_1, 0)\}$, and the situation $(3, 6)$. As discussed above, $\mathrm{ET}(\xi_1, (3, 6)) = 3$. Therefore, in this situation, any decision, (t, χ), for which $t \geq 3$ would result in C_1 executing at time 3. However, a decision such as $(2.5, \{X\})$ would, in that same situation, instead result in X being executed at time 2.5. Finally, in a different situation, such as $(2, 6)$, the decision $(2.5, \{X\})$ would result in an outcome where C_1 executed at time 2.

Definition 19 (*RTED-based execution strategy* [10]) An RTED-based execution strategy for an STNU is a mapping, R, from respectful partial schedules to *allowable* real-time execution decisions. Thus, if ξ is a respectful partial schedule, then $R(\xi)$ is an allowable RTED.

For example, recall the informally expressed execution strategy for $\mathcal{S}^\dagger$ seen earlier:

- Execute A_1 at time 0.
- Execute X at time 1.
- If C_1 is observed to execute *before* time 4, then execute A_2 at time $(C_1 + 4)/2$; otherwise, execute A_2 at time 4.

This strategy can be translated into an RTED-based strategy, R, as follows:

- If $\mathtt{now}_\xi < 0$, then $R(\xi) = (0, \{A_1\})$.
- If $\mathtt{now}_\xi \in [0, 1)$, then $R(\xi) = (1, \{X\})$.
- If $\mathtt{now}_\xi \in [1, 4)$ and $C_1 \notin \mathrm{Vars}_\xi$, then $R(\xi) = (4, \{A_2\})$.
- If $\mathtt{now}_\xi \in [1, 4)$ and $C_1 \in \mathrm{Vars}_\xi$, then $R(\xi) = ((\xi(C_1) + 4)/2, \{A_2\})$.
- Else $R(\xi) = \mathtt{wait}$.

For any situation ω, an RTED-based strategy R determines a unique sequence of (mostly partial) schedules, starting with the initial partial schedule, $\xi_0 = \emptyset$, as follows. $R(\xi_0)$ provides the first execution decision. The outcome of that decision is $\mathcal{O}(\xi_0, \omega, R(\xi_0))$ which, of course, depends on the situation. That outcome becomes the next partial schedule, ξ_1, in the sequence. The execution cycle then continues with $R(\xi_1)$ providing the next execution decision, and so on. Since each outcome involves the execution of at least one time-point, the process eventually terminates in a complete schedule, ξ_p.

A straightforward proof by induction [10] confirms the following lemma.

Lemma 2 *Let R be an RTED-based execution strategy for some STNU. Then each situation ω determines a unique sequence of (mostly partial) schedules, $\emptyset = \xi_0 \subset \xi_1 \subset \cdots \subset \xi_p$, such that:*

- *For each $i < p$, ξ_i respects ω, $\xi_{i+1} = \mathcal{O}(\xi_i, \omega, R(\xi_i))$, and $\texttt{now}_{\xi_i} < \texttt{now}_{\xi_{i+1}}$; and*
- *ξ_p is a complete schedule that respects ω.*

Definition 20 ($\mathcal{O}^*(R, \omega)$) Let R be an RTED-based execution strategy for some STNU; and let $\omega \in \Omega$ be any situation. Then $\mathcal{O}^*(R, \omega)$ denotes the complete schedule, ξ_p, that is uniquely determined by R and ω, as described in Lemma 2. The schedule $\mathcal{O}^*(R, \omega)$ may be called the result of following the strategy R in the situation ω.

In this way, each RTED-based execution strategy R, like the dynamic execution strategies from MMV-01, effectively specifies a mapping from situations to complete schedules. However, the RTED-based execution semantics ensures that the complete schedules resulting from RTED-based strategies necessarily respect each situation.

For example, recall the execution strategy R for the sample STNU $\mathcal{S}^\dagger$, given above. In the situation, $\omega = (3, 6)$, it yields the following sequence of (mostly partial) schedules. Note that A_1 plays the role of the zero time-point.

$$\xi_0 = \emptyset \text{ and } R(\xi_0) = (0, \{A_1\})$$
$$\xi_1 = \mathcal{O}(\xi_0, \omega, (0, \{A_1\})) = \{(A_1, 0)\} \text{ and } R(\xi_1) = (1, \{X\})$$
$$\xi_2 = \mathcal{O}(\xi_1, \omega, (1, \{X\})) = \{(A_1, 0), (X, 1)\} \text{ and } R(\xi_2) = (4, \{A_2\})$$
$$\xi_3 = \mathcal{O}(\xi_2, \omega, (4, \{A_2\})) = \{(A_1, 0), (X, 1), (C_1, 3)\} \text{ and } R(\xi_3) = (3.5, \{A_2\})$$
$$\xi_4 = \mathcal{O}(\xi_3, \omega, (3.5, \{A_2\})) = \{(A_1, 0), (X, 1), (C_1, 3), (A_2, 3.5)\} \text{ and } R(\xi_4) = \texttt{wait}$$
$$\xi_5 = \mathcal{O}(\xi_4, \omega, \texttt{wait}) = \{(A_1, 0), (X, 1), (C_1, 3), (A_2, 3.5), (C_2, 9.5)\}$$

Notice that the outcome, ξ_3, of the decision, $R(\xi_2) = (4, \{A_2\})$, does not involve the execution of A_2 at time 4 because, in the given situation, C_1 happens to execute first, at time 3. The strategy is then able to react—although not instantaneously—by deciding to execute A_2 at time 3.5 instead. In the given situation, that leads to A_2 actually being executed at time 3.5. Finally, consider the complete schedule, $\xi_5 = \mathcal{O}^*(R, (3, 6))$, that results from following this strategy in the situation (3, 6). As guaranteed by Lemma 2, ξ_5 respects the durations in that situation. It also happens to satisfy all of the constraints in the STNU.

More generally, suppose that R is an RTED-based strategy for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, and that ω is any situation. Although Lemma 2 guarantees that the result $\mathcal{O}^*(R, \omega)$ of following the strategy R in the situation ω necessarily respects the durations in ω, it does not guarantee that the schedule $\mathcal{O}^*(R, \omega)$ will necessarily satisfy the constraints in $\mathcal{C}$. Only *reliable* RTED-based strategies, defined below, have that property.

Definition 21 (*Reliable*) An RTED-based strategy R for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is called *reliable* if for each situation $\omega \in \Omega$, the complete schedule $O^*(R, w)$ satisfies all of the constraints in $\mathcal{C}$.

To clarify the relationship between RTED-based strategies and dynamic execution strategies (cf. Definition 11), it is useful to extend the notion of respect (cf. Definition 14) to dynamic execution strategies and to formally define a notion of equivalence between RTED-based strategies and dynamic execution strategies.

Definition 22 (*Respectful DES* [10]) A dynamic execution strategy S is called *respectful* if for each situation $\omega \in \Omega$, the complete schedule $S(\omega)$ respects the durations in ω.

Definition 23 (*Equivalence of execution strategies* [10]) A dynamic execution strategy S and an RTED-based strategy R are called *equivalent* if for each $\omega \in \Omega$, the complete schedules, $S(\omega)$ and $\mathcal{O}^*(R, \omega)$, are identical.

Fact 2 *Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be any STNU; let R be any RTED-based strategy for $\mathcal{S}$; and let S be an* equivalent *dynamic execution strategy for $\mathcal{S}$, as described in Definition 23. Then S is necessarily respectful; and R is reliable if and only if S is viable.*

Proof Let ω be any situation. Since R and S are equivalent, the complete schedules $\mathcal{O}^*(R, \omega)$ and $S(\omega)$ are identical. By Lemma 2, $\mathcal{O}^*(R, \omega)$—and hence $S(\omega)$—respects the durations in ω. Thus, S is respectful. If, in addition, R is reliable, then $\mathcal{O}^*(R, \omega)$—and hence $S(\omega)$—satisfies all the constraints in $\mathcal{C}$, whence S is viable. The converse is handled similarly. □

The following theorem explicates the correspondence between RTED-based strategies and respectful dynamic execution strategies. Its in-depth proof is available elsewhere [10].

Theorem 2 (Correspondence of execution strategies [10]) *For any STNU $\mathcal{S}$, there is a one-to-one correspondence between RTED-based execution strategies and respectful dynamic execution strategies. In particular, for any RTED-based strategy R, there is an equivalent respectful dynamic execution strategy S; and for any respectful dynamic execution strategy S, there is an equivalent RTED-based strategy R.*

Corollary 2 *An STNU is dynamically controllable if and only if there exists a reliable RTED-based execution strategy for it.*

Proof First, suppose that $\mathcal{S}$ is dynamically controllable. Then, by definition, there exists a viable dynamic execution strategy S for $\mathcal{S}$. Since S being viable implies that S is respectful, there must be an equivalent RTED-based strategy, by Theorem 2. But then R must be reliable, by Fact 2. The converse is handled similarly. □

As summarized above, the MMV-01 and RTED-based semantics for STNUs and dynamic controllability are equivalent. The rest of the paper uses the RTED-based semantics exclusively, primarily because its explicit representation of the incremental nature of the execution of an STNU over time facilitates the theoretical analysis.

2.4 STNU graphs

The most efficient algorithms for determining whether STNUs are dynamically controllable are based on the generation of edges in STNU graphs. There are many helpful parallels between STN graphs and STNU graphs; however, there are also important differences. First, not every path in an STNU graph corresponds to a constraint that must be satisfied; instead, only a subset of paths—the so-called *semi-reducible* paths—have that property. As will be seen, an STNU is dynamically controllable if and only if its graph has no *semi-reducible* negative loops. This section reviews STNU graphs, rules for generating new edges, and the key notion of *semi-reducible* paths.

Definition 24 (*STNU graphs* [19]) The graph for an STNU, $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, is a pair, $\mathcal{G} = \langle \mathcal{T}, \mathcal{E}^+ \rangle$, where the time-points in $\mathcal{T}$ serve as the nodes in the graph, and the constraints in $\mathcal{C}$ and the contingent links in $\mathcal{L}$ together correspond to the edges in $\mathcal{E}^+$, as follows. First, for each ordinary constraint, $Y - X \leq \delta$ in $\mathcal{C}$, there is an ordinary edge, $X \xrightarrow{\delta} Y$ in $\mathcal{E}^+$, exactly the same as for STNs. Next, for each contingent link, (A, x, y, C), there are two *labeled* edges in $\mathcal{E}^+$: a *lower-case* edge, $A \xrightarrow{c:x} C$, and an *upper-case* edge, $A \xleftarrow{C:-y} C$.

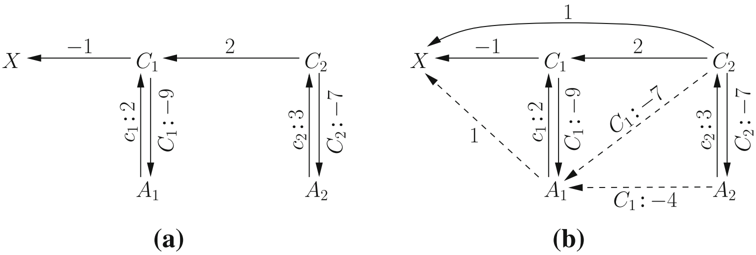


Fig. 2 **a** The graph $\mathcal{G}^\dagger$ for the sample STNU $\mathcal{S}^\dagger$. **b** The same graph after adding new edges

Table 1 The edge-generation rules for STNUs from Morris and Muscettola [19]

Rule	If these edges exist …	And if …	Then add this edge:
(No case)	$D \xrightarrow{v} E \xrightarrow{w} F$		$D \xrightarrow{v+w} F$
(Upper case)	$D \xrightarrow{v} E \xrightarrow{B:w} F$	$D \not\equiv B$	$D \xrightarrow{B:v+w} F$
(Lower case)	$A \xrightarrow{c:v} C \xrightarrow{w} F$	$w \leq 0, C \not\equiv F$	$A \xrightarrow{v+w} F$
(Cross case)	$A \xrightarrow{c:v} C \xrightarrow{B:w} F$	$w \leq 0, B \not\equiv C \not\equiv F$	$A \xrightarrow{B:v+w} F$
(Label removal)	$D \xrightarrow{C:z} A \xrightarrow{c:x} C$	$z \geq -x$	$D \xrightarrow{z} A$

Each lower-case edge in an STNU graph represents the *uncontrollable possibility* that the duration of the corresponding contingent link might be its *minimum* value x; and each upper-case edge represents the *uncontrollable possibility* that the duration of the corresponding contingent link might be its *maximum* value y.[12]

The graph for the sample STNU $\mathcal{S}^\dagger$ is shown in Fig. 2a. Note that the labeled edges joining A_1 and C_1 in the STNU graph are quite different from the ordinary edges joining A and C in the only-superficially-similar STN graph in Fig. 1a. With the STN, the agent controls the execution of both A and C, and is free to choose any value in [2, 9] for the duration, $C - A$. With the STNU, the agent controls only A_1; the environment chooses the duration, $C_1 - A_1$.

Edge generation and path transformation in STNU graphs

Given that the labeled edges in an STNU graph represent uncontrollable possibilities, unlike the ordinary edges which represent constraints to be satisfied, constraint propagation (equivalently, edge generation) for STNUs is necessarily more complicated than for STNs. For example, without their alphabetic labels, the two labeled edges from the same contingent link would be mutually inconsistent. As this example suggests, the alphabetic labels on edges in an STNU graph must be carefully managed during constraint propagation/edge generation. Toward that end, Morris and Muscettola [19] presented the edge-generation rules listed in Table 1.[13]

[12] In the original definition of STNU graphs [19], each contingent link also gave rise to two *ordinary* edges, $A \xrightarrow{y} C$ and $A \xleftarrow{-x} C$, representing the *known fact* that the duration of the contingent link must fall within the interval $[x, y]$. However, it has been shown that including these extra edges is not necessary—because the execution semantics for STNUs ensures that they will be satisfied [13].

[13] In Table 1, applicability conditions of the form, $X \not\equiv Y$, should be construed as requiring that X and Y be distinct time-points, not as a constraint on their values.

These rules are equivalent to, but more uniformly and concisely expressed than the constraint-propagation rules for STNUs presented in earlier work [18].

For convenience, the pre-existing edges in the edge-generation rules from Table 1 are called *parent edges,* and the generated edges are called *child edges.* The only exception is the pre-existing lower-case edge from A to C in the *Label Removal* rule. That edge is not considered to be a parent edge; instead, it is viewed as providing the context within which the upper-case label can be removed from the pre-existing edge from D to A. Since the first four rules involve pairs of parent edges, they are called *binary* edge-generation rules.

The No Case *rule* The *No Case* rule encodes the generation of ordinary edges as happens in STN graphs or any shortest-paths computation. In the context of determining dynamic controllability, this rule is *sound* in the sense that any RTED-based strategy that satisfies the parent constraints, $F - E \leq w$ and $E - D \leq v$, necessarily satisfies the generated child constraint, $F - D \leq v + w$, since $F - D = (F - E) + (E - D) \leq w + v = v + w$.

The next three rules in Table 1 generate edges that effectively guard against the uncontrollable possibilities represented by involved lower-case and upper-case edges. These rules are illustrated in Fig. 2b, starting from the sample STNU graph seen earlier in Fig. 2a. For comparison purposes, the edges generated by the rules are shown as dashed arrows in Fig. 2b.

The Lower Case *rule* The *Lower Case* rule generates edges that effectively guard against the uncontrollable possibility that a contingent link might take on its minimum duration. Applying this rule to the edges from A_1 to C_1 to X in Fig. 2b generates the ordinary edge, $A_1 \xrightarrow{1} X$, shown as dashed in the figure. (Note that the edge, $A_1 \xrightarrow{1} X$, generated by the *Lower Case* rule represents a much stronger constraint than the corresponding edge, $A \xrightarrow{8} X$, generated earlier for the sample STN in Fig. 1.) Intuitively, this application of the rule expresses the following: given the uncontrollable possibility that the contingent duration, $C_1 - A_1$, might equal its minimum value 2, any RTED-based strategy that satisfies the pre-existing constraint, $X - C_1 \leq -1$, must also satisfy the generated constraint, $X - A_1 \leq 1$. To informally verify that this is the case, first, note that to satisfy the pre-existing constraint, $X - C_1 \leq -1$ (i.e., $X \leq C_1 - 1$), the agent must execute X *before* C_1 (i.e., before knowing the value of the contingent duration, $C_1 - A_1$). Since C_1 might happen to execute as early as 2 units after A_1, the agent can only ensure that $X \leq C_1 - 1$ holds by requiring X to occur no more than 1 unit after A_1.

The Upper Case *rule* The *Upper Case* rule generates edges that effectively guard against the uncontrollable possibility that a contingent link might take on its maximum duration. For example, applying this rule to the edges from C_2 to C_1 to A_1 in Fig. 2 generates the upper-case edge, $C_2 \xrightarrow{C_1:-7} A_1$, shown as dashed in the figure. This new edge represents a conditional *wait* constraint that can be glossed as: "If C_1 is not yet executed, then C_2 must *wait* at least 7 units after A_1" [18]. Thus, this application of the *Upper Case* rule effectively makes the following assertion: given the uncontrollable possibility that the contingent duration, $C_1 - A_1$, might equal its maximum value 9, any RTED-based strategy that satisfies the pre-existing constraint, $C_1 - C_2 \leq 2$, must also ensure that while C_1 remains unexecuted, C_2 waits at least 7 units after A_1. Given that the agent cannot directly control the execution of the contingent time-point C_2, the only way it can ensure that this conditional wait constraint is satisfied by C_2 is through its direct control over the corresponding activation time-point A_2.

The Cross Case *rule* The *Cross Case* rule from Table 1 covers the interaction of two different contingent links. Thus, it generates new edges that effectively guard against the uncontrollable possibility of one contingent link taking on its minimum duration, while another takes on its maximum duration. Applying the *Cross Case* rule to the edges from A_2 to C_2 to A_1 in Fig. 2b, where the upper-case edge from C_2 to A_1 was generated previously, yields the upper-case edge, $A_2 \xrightarrow{C_1:-4} A_1$. This upper-case edge can be glossed as: "As long as C_1 remains unexecuted, A_2 must wait until at least 4 units after the execution of A_1." Since A_2 is an executable time-point that the agent directly controls, the agent can enforce this constraint. Doing so will ensure that the wait constraint on the contingent time-point C_2, discussed above, will be satisfied. Thus, by controlling the execution of the activation time-point, A_2, the agent indirectly controls the timing of the contingent time-point, C_2.

The Label Removal *rule* The *Label Removal* rule specifies conditions under which the upper-case label may be removed from an upper-case edge. The *Label Removal* rule is not considered a binary rule because it has only one parent edge: the upper-case edge from D to A in Table 1. The lower-case edge from A to C is not considered to be a parent edge; instead, its presence is part of the applicability conditions for this rule. Those conditions stipulate that if the wait time for the upper-case edge is less than or equal to the minimum duration for the corresponding contingent link, then the upper-case label may be removed from the upper-case edge. In other words, because the contingent time-point C cannot execute before the wait-time has expired, the conditional wait constraint has the force of an unconditional constraint. For example, suppose that, in Fig. 2b, the contingent link from A_1 to C_1 had a lower bound of 5 instead of 2. In that case, the conditional constraint, "As long as C_1 remains unexecuted, A_2 must wait until at least 4 units after A_1", could be replaced by the unconditional constraint, "A_2 must wait until at least 4 units after A_1". Thus, in such a case, the upper-case edge, $A_2 \xrightarrow{C_1:-4} A_1$, could be replaced by the ordinary edge, $A_2 \xrightarrow{-4} A_1$.

Each edge-generation rule $\mathcal{R}$ from Table 1 will be shown to be *sound* in the following sense: any RTED-based strategy that satisfies the parent constraints in any instance of the rule $\mathcal{R}$ must also satisfy the child constraint generated by that rule instance. However, some techniques introduced by Morris [16] for analyzing the structure of STNU graphs will simplify the task of proving the soundness of the rules.

Path transformation Each of the edge-generation rules from Table 1 can be viewed as a path-transformation (or path-reduction) rule, as follows.

Definition 25 (*Path transformation/reduction* [16]) Suppose a path $\mathcal{P}$ contains consecutive edges, e_1 and e_2, to which one of the binary edge-generation rules applies, yielding a new edge, e', as illustrated on the lefthand side of Fig. 3. The path $\mathcal{P}'$ obtained from $\mathcal{P}$ by replacing e_1 and e_2 by e' is called a *transformation* of $\mathcal{P}$. Equivalently, $\mathcal{P}$ is said to *reduce* to $\mathcal{P}'$. The *Label Removal* rule can similarly be used to transform (or *reduce*) a path, as illustrated on the righthand side of Fig. 3. Furthermore, any sequence of such transformations is also considered a transformation.

For example, the path from C_2 to C_1 to A_1 in Fig. 2b reduces to the edge from C_2 to A_1 in one step, while the path from A_2 to C_2 to C_1 to A_1 reduces to the edge from A_2 to A_1 in two steps. Of course, path transformations do not always result in a single edge (cf. Fig. 3).

Definition 26 (*Reduced distance* [16]) For any path $\mathcal{P}$ in an STNU graph, the *reduced distance* of $\mathcal{P}$, denoted by $|\mathcal{P}|$, is the length of $\mathcal{P}$, ignoring any alphabetic labels.

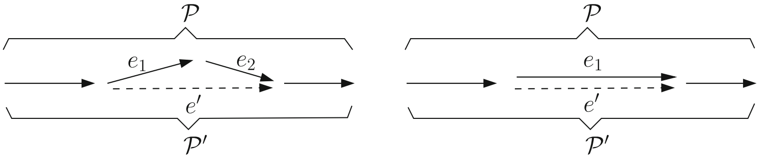


Fig. 3 Using edge-generation rules to transform a path in an STNU graph

For example, the reduced distance (or length) of the path from A_2 to C_2 to C_1 to A_1 in Fig. 2b is -4. Note that all of the edge-generation rules from Table 1 preserve reduced distance. As a result, the generated edge from A_2 to A_1 in Fig. 2b also has length -4.

2.4.1 Semi-reducible paths

Recall that any path in an STN graph represents a constraint that must be satisfied by any solution for that STN. However, it is *not* in general the case that any path in an STNU graph represents a constraint that must be satisfied by any reliable RTED-based execution strategy for that STNU. As will be seen, for STNUs, it is only the *semi-reducible* paths, defined below, that represent constraints that must be satisfied by any reliable RTED-based strategy. Thus, in a dynamically controllable network, the shortest semi-reducible paths represent the strongest constraints that any reliable execution strategy must satisfy.

Definition 27 (*Semi-reducible path* [16]) A path in an STNU graph is called *semi-reducible* if it can be transformed into a path without any *lower-case* edges.

For a negative example, consider the loop in Fig. 2b from A_1 to C_1 to A_1. Since the constituent edges are the lower-case and upper-case edges from the same contingent link, $(A_1, 2, 9, C_1)$, the *Cross Case* rule does not apply to this loop; thus, it is not semi-reducible. Neither does it represent any kind of constraint that must be satisfied by a reliable execution strategy because its two labeled edges represent alternative possibilities for a single contingent duration. Nonetheless, each of these edges can be used separately by the edge-generation rules from Table 1 to generate other edges that *do* represent constraints that must be satisfied by any reliable strategy, as discussed earlier.

For a positive example of a semi-reducible path, consider the path, $\mathcal{P}$, shown on the lefthand side of Fig. 4. The edges in this path belong to the graph for the sample STNU $\mathcal{S}^\dagger$, seen previously in Fig. 2b. To clarify the order of the edges in $\mathcal{P}$, the time-point C_1 has been drawn twice in Fig. 4. As shown on the righthand side of Fig. 4, $\mathcal{P}$ is semi-reducible because it can be transformed into a path, $\mathcal{P}'$, containing only ordinary or upper-case edges. In this

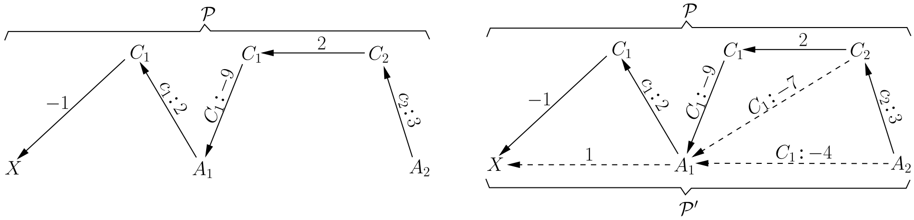


Fig. 4 Transforming a semi-reducible path, $\mathcal{P}$, into a path, $\mathcal{P}'$, having only ordinary or upper-case edges

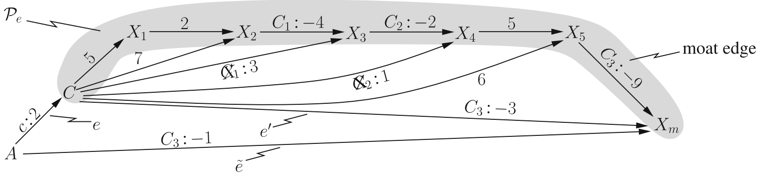


Fig. 5 The *canonical elimination* of a lower-case edge, e, by its extension sub-path, $\mathcal{P}_e$

case, $\mathcal{P}'$ has two edges. Note that although $\mathcal{P}$ is semi-reducible, it contains a sub-loop, from C_1 to A_1 to C_1, that is not semi-reducible.

Analogously to the negative loops in an STN graph whose presence characterizes inconsistent STNs, the presence of *semi-reducible negative loops* (SRN loops) in an STNU graph will be shown to characterize STNUs that are *not* dynamically controllable.

Definition 28 (*SRN loop* [13]) A *semi-reducible negative loop* (SRN loop) is any semi-reducible path $\mathcal{P}$ that is a loop with negative length.

The following notation helps to simplify the subsequent presentation.

Definition 29 (*LO-edge/path, OU-edge/path*, OU_g*-edge/path*)

LO-edge	Any edge that is either a lower-case edge or an ordinary edge
OU-edge	Any edge that is either an ordinary edge or an upper-case edge
OU_g*-edge*	Any edge that is either an ordinary edge or a *generated* upper-case edge
LO-path	Any path consisting solely of LO-edges
OU-path	Any path consisting solely of OU-edges
OU_g*-path*	Any path consisting solely of OU_g-edges

For example, a semi-reducible path is any path that can be transformed into an OU-path using the edge-generation rules from Table 1. Below, concepts and techniques introduced by Morris [16] for analyzing the structure of semi-reducible paths are summarized.

Extension sub-paths, moat edges and breaches If a path $\mathcal{P}$ is semi-reducible, then in the transformation of $\mathcal{P}$ into an OU-path, each occurrence, e, of a lower-case edge in $\mathcal{P}$ must eventually be "eliminated" by an application of either the *Lower Case* or *Cross Case* rule.[14] Thus, for each such e, there must be some sub-path of $\mathcal{P}$—call it $\mathcal{P}_e$—that reduces to a single edge, e', such that e and e' can then be transformed into a single edge, $\tilde{e}$, as illustrated in Fig. 5, where e' is the edge, $C \overset{C_3:-3}{\longrightarrow} X_m$, and $\tilde{e}$ is the edge, $A \overset{C_3:-1}{\longrightarrow} X_m$.[15]

Definition 30 (*Extension sub-path and moat edge* [13,16]) Let e be an occurrence of a lower-case edge in a path $\mathcal{P}$ in an STNU graph. The *extension sub-path* and *moat edge* for e in $\mathcal{P}$ are defined as follows. First, let $e_1, e_2, \ldots$ be the sequence of edges immediately *following* e in $\mathcal{P}$. (Thus, $e \neq e_1$.) Then, for each i, let $\mathcal{P}_e^i$ be the path consisting of the

[14] Note that a semi-reducible path might contain multiple occurrences of the same lower-case edge [13].

[15] Definitions 30 and 31, and the proof technique for Theorem 5, below, were originally presented by Morris [16], but only in the context of an execution semantics that allowed a form of instantaneous reactivity. This author subsequently modified them to conform to the standard STNU execution semantics [11,13].

edges, $e_1, \ldots, e_i$ and, if it exists, let m be the smallest integer such that either: $|\mathcal{P}_e^m| < 0$; or $|\mathcal{P}_e^m| = 0$ and $\mathcal{P}_e^m$ is *not* a loop. The *extension sub-path* (ESP) for e in $\mathcal{P}$, notated $\mathcal{P}_e$, is the sub-path $\mathcal{P}_e^m$; and its last edge, e_m, is the *moat edge* for e in $\mathcal{P}$. If no such m exists, then e has no ESP or moat edge in $\mathcal{P}$.

Within any *semi-reducible* path, each occurrence, e, of a lower-case edge can be reduced away by its extension sub-path $\mathcal{P}_e$, as illustrated in Fig. 5 (cf. Corollary 2 in Hunsberger [13]). Furthermore, the properties of extension sub-paths (e.g., every proper prefix has non-negative length) are such that the transformation of $\mathcal{P}_e$ into a single edge, e', can always be done in the left-to-right manner shown in the figure, where any upper-case edges that are encountered along the way can have their labels removed, indicated by their being crossed out in the figure. Morris calls this the *canonical elimination* of a lower-case edge [16].

If e is the lower-case edge, $A \xrightarrow{c:x} C$, and some upper-case edge, $V \xrightarrow{C:-w} A$, whose upper-case label is C, occurs in an extension sub-path for e, then that upper-case edge has the potential of blocking the elimination of e—and thereby threatening the semi-reducibility of the path $\mathcal{P}$. (Recall that the *Cross Case* rule does not apply if the lower-case and upper-case edges have corresponding alphabetic labels.) Such edges are called *breaches.*

Definition 31 (*Breach* [16]) Let e be a lower-case edge, $A \xrightarrow{c:x} C$; and let $\mathcal{P}_e$ be the extension sub-path for e in some path $\mathcal{P}$. Any occurrence in $\mathcal{P}_e$ of an *upper-case* edge labeled by C is called a *breach.*

3 Fundamental theorem for STNUs, part 1

The main thrust of the fundamental theorem for STNUs is that an STNU is dynamically controllable if and only if its graph contains no semi-reducible negative loops. This result provides the basis for all DC-checking algorithms that have been presented so far in the literature. However, this crucial result has, to date, not been given a rigorous and complete proof. As a result, important details have been glossed over or missed altogether. One of the main goals of this paper is to provide such a proof. This is done in two steps: (1) proving that an STNU that has an SRN loop must *not* be dynamically controllable; and (2) proving that an STNU *without* any SRN loops must be dynamically controllable. The first step is addressed in this section, the second step in the next section.

Proving that an STNU that has an SRN loop must not be dynamically controllable is done by proving that the edge-generation rules from Table 1 are sound—that is, whenever an RTED-based strategy "satisfies" the relevant parent edge(s) in an edge-generation rule, then it necessarily satisfies the corresponding child edge generated by that rule. Therefore, this section begins by specifying what it means for an RTED-based strategy to satisfy an edge in an STNU, effectively specifying how each kind of STNU edge should be interpreted.

A subtle point arises with regard to the upper-case edges that are generated by the *Upper Case* and *Cross Case* rules, in contrast to the upper-case edges in the original STNU graph. As has already been suggested, the original upper-case edges represent uncontrollable possibilities, while the generated upper-case edges, such as $V \xrightarrow{C:-r} A$, represent conditional wait constraints that can be glossed as: "While C remains unexecuted, V must wait at least r units after A." Now, suppose that the generated upper-case edge, $V \xrightarrow{C:-r} A$, corresponds to the contingent link is (A, x, y, C), and the wait amount r is greater than y. (Such wait times will be called *improper.*) Since C cannot remain unexecuted after $A + y$, this upper-case edge could be replaced by $V \overset{C:-y}{\longrightarrow} A$, where the wait time has been decreased to y. Although such a rule would be sound, it would not be length preserving; thus, its use would violate one of the most important preconditions of Morris' techniques for analyzing the structure of semi-reducible paths, which play important roles in the rest of the paper. Similar remarks apply to the *General Unordered Reduction* rule from the MMV-01 paper [18], which is also not length preserving. This issue becomes particularly important here because some of the proofs in Sect. 4 depend on propagating *all* wait constraints, even those whose wait times are greater than the upper bound on the corresponding contingent link.

Recall that the execution semantics for STNUs ensures that the complete schedule that results from following an RTED-based strategy in any given situation necessarily satisfies the bounds on all contingent links. This models the property that a decision by an agent cannot cause the environment to violate the bounds on any contingent link. However, not all RTED-based strategies will necessarily generate complete schedules that satisfy the ordinary constraints associated with ordinary edges in an STNU graph or the conditional wait constraints associated with *generated* upper-case edges. To determine whether the outcomes of an RTED-based strategy will satisfy the constraints associated with ordinary or *generated* upper-case edges (i.e., with OU$_g$-edges) requires specifying what it means for an RTED-based strategy to *satisfy* an OU$_g$-edge. Doing so provides a formal interpretation of OU$_g$-edges in terms of ordinary constraints or conditional wait constraints.

Definition 32 (*Satisfaction of* OU$_g$ *edges/constraints*) Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU; let R be an RTED-based strategy; let e be an ordinary edge, $X \overset{\delta}{\longrightarrow} Y$; and let E be a *generated* upper-case edge, $V \overset{C:u}{\longrightarrow} A$, where the relevant contingent link is (A, x, y, C).

- R is said to *satisfy* the ordinary edge e if for every situation ω, the corresponding complete schedule, $\xi_\omega = \mathcal{O}^*(R, \omega)$, satisfies $\xi_\omega(Y) - \xi_\omega(X) \leq \delta$. In such cases, R may also be said to satisfy the corresponding constraint, $Y - X \leq \delta$.
- R is said to *satisfy* the *generated* upper-case edge E if for each situation ω, the corresponding complete schedule, $\xi_\omega = \mathcal{O}^*(R, \omega)$, satisfies either $\xi_\omega(V) \geq \xi_\omega(A) - u$ (i.e., V waited $-u$ units after A) or $\xi_\omega(V) > \xi_\omega(C)$ (i.e., C executed before V). In such cases, R may also be said to satisfy the corresponding conditional constraint, "while C is unexecuted, V must wait at least $-u$ units after A".

Definition 33 (*Soundness for edge-generation rules*) An edge-generation rule (cf. Table 1) is *sound* if for all RTED-based strategies R the following holds: whenever R satisfies all of the rule's OU$_g$ parent edges under the relevant applicability conditions, R also satisfies the corresponding child edge.

The soundness of the edge-generation rules is given by Lemmas 3–10, below. Aside from the heavy use of Lemma 3, which draws from prior work [10], and the Proof of Lemma 9, which is novel, many of the techniques used in the proofs of Lemmas 5–8 and 10 are similar to those introduced by Morris et al. in the soundness proofs of the MMV-01 edge-generation rules [18]. To simplify the presentation of the proofs, the following notation is used: given an RTED-based strategy R, a situation ω, and a time-point X, the value assigned to X by the complete schedule $\mathcal{O}^*(R, \omega)$ that results from following the strategy R in the situation ω is denoted by X_ω. Notation such as X_ω is only used when the strategy R is understood from the context.

Lemma 3, below, encapsulates a technique used repeatedly in the subsequent lemmas.

Lemma 3 *Let R be an RTED-based strategy for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$; let (A, x, y, C) be a contingent link in $\mathcal{S}$; let $\tilde{\omega}$ and $\hat{\omega}$ be two situations that are identical except that $C_{\tilde{\omega}} - A_{\tilde{\omega}} < C_{\hat{\omega}} - A_{\hat{\omega}}$; and let $d \in \mathbb{R}$ be the first point at which the sequences of (mostly partial) schedules generated by following the strategy R in the situations, $\tilde{\omega}$ and $\hat{\omega}$, differ. Then all of the following hold:*

- *$A_{\hat{\omega}} = A_{\tilde{\omega}} < C_{\tilde{\omega}} = d < C_{\hat{\omega}}$;*
- *if X is an executable time-point such that $X_{\tilde{\omega}} \leq d$ or $X_{\hat{\omega}} \leq d$, then $X_{\tilde{\omega}} = X_{\hat{\omega}}$; and*
- *if B is a contingent time-point other than C such that $B_{\tilde{\omega}} \leq d$ or $B_{\hat{\omega}} \leq d$, then $B_{\tilde{\omega}} = B_{\hat{\omega}}$.*

Proof By Lemma 2, the situation $\tilde{\omega}$ determines a unique sequence of (mostly partial) schedules that result from following the strategy R in $\tilde{\omega}$, notated as follows:

$$\emptyset = \xi_0^{\tilde{\omega}} \subset \xi_1^{\tilde{\omega}} \subset \cdots \subset \xi_{\tilde{p}}^{\tilde{\omega}}, \text{ where for each } 0 \leq i < \tilde{p},\ \xi_{i+1}^{\tilde{\omega}} = \mathcal{O}(\xi_i^{\tilde{\omega}}, \tilde{\omega}, R(\xi_i^{\tilde{\omega}})).$$

Similarly, the situation $\hat{\omega}$ determines a unique sequence notated as follows:

$$\emptyset = \xi_0^{\hat{\omega}} \subset \xi_1^{\hat{\omega}} \subset \cdots \subset \xi_{\hat{p}}^{\hat{\omega}}, \text{ where for each } 0 \leq i < \hat{p},\ \xi_{i+1}^{\hat{\omega}} = \mathcal{O}(\xi_i^{\hat{\omega}}, \hat{\omega}, R(\xi_i^{\hat{\omega}})).$$

Let j be the greatest index for which $\xi_j^{\tilde{\omega}} = \xi_j^{\hat{\omega}}$. Then $j + 1$ is the first index for which $\xi_{j+1}^{\tilde{\omega}} \neq \xi_{j+1}^{\hat{\omega}}$. As a result, d, the time of the first difference between the two sequences, is given by: $d = \min\{\mathtt{now}_{\xi_{j+1}^{\tilde{\omega}}}, \mathtt{now}_{\xi_{j+1}^{\hat{\omega}}}\}$. Furthermore, by construction, no execution events occur in either sequence strictly between the time of the jth partial schedules and d. Finally, because the jth partial schedules are identical, the decisions generated by R based on them must be identical: $R(\xi_j^{\tilde{\omega}}) = R(\xi_j^{\hat{\omega}})$.

Now, suppose A were not yet executed in the jth partial schedules. In that case, the contingent time-point C would not be currently active and, because all contingent durations other than $C - A$ are identical in $\tilde{\omega}$ and $\hat{\omega}$, it follows that $\mathrm{ET}(\xi_j^{\tilde{\omega}}, \tilde{\omega}) = \mathrm{ET}(\xi_j^{\hat{\omega}}, \hat{\omega})$ and $\chi_c(\xi_j^{\tilde{\omega}}, \tilde{\omega}) = \chi_c(\xi_j^{\hat{\omega}}, \hat{\omega})$. But then the outcomes, $\xi_{j+1}^{\tilde{\omega}}$ and $\xi_{j+1}^{\hat{\omega}}$ would necessarily be the same, contradicting the choice of j. Thus, A must already be executed in the jth partial schedules (i.e., $A_{\tilde{\omega}} = A_{\hat{\omega}}$) and C must be currently active. Furthermore, the difference in the $(j + 1)^{\text{st}}$ outcomes can only be due to differences in the corresponding ET values and χ_c sets, which are due to the fact that $C_{\tilde{\omega}} - A_{\tilde{\omega}} < C_{\hat{\omega}} - A_{\hat{\omega}}$. Thus, $\mathrm{ET}(\xi_j^{\tilde{\omega}}, \tilde{\omega}) < \mathrm{ET}(\xi_j^{\hat{\omega}}, \hat{\omega})$, and, in the case of a (t, χ) decision, $\mathrm{ET}(\xi_j^{\tilde{\omega}}, \tilde{\omega}) \leq t$. Therefore, in the situation $\tilde{\omega}$, C executes at time d, while in the situation $\hat{\omega}$, C executes at some later time (i.e., $C_{\tilde{\omega}} = d < C_{\hat{\omega}}$).

Now, suppose that X is an executable time-point that executes at or before time d in $\tilde{\omega}$ or $\hat{\omega}$. If X is already executed in the jth partial schedules, then $X_{\tilde{\omega}} = X_{\hat{\omega}}$. Otherwise, X must execute at time d, as part of the outcome of a (t, χ) decision; whence $X \in \chi$ and $d = t$, implying that X executes at time d in both situations (i.e., $X_{\tilde{\omega}} = X_{\hat{\omega}}$).

Similarly, suppose that B is a contingent time-point other than C that executes at or before time d in $\tilde{\omega}$ or $\hat{\omega}$. If B is already executed in the jth partial schedules, then $B_{\tilde{\omega}} = B_{\hat{\omega}}$. Otherwise, B must execute at time d in one of the situations. But that implies that B was currently active in the jth partial schedules, in which case its activation time-point was executed at the same time in both situations. Since the contingent duration for the relevant contingent link is the same in both situations, B must execute at the same time in both situations (i.e., $B_{\tilde{\omega}} = B_{\hat{\omega}}$). □

Lemma 4 *The* No Case *rule from Table 1 is sound.*

Proof Let R be an RTED-based strategy that satisfies the ordinary edges, $D \xrightarrow{v} E$ and $E \xrightarrow{w} F$. Then, for every situation ω, $E_\omega - D_\omega \leq v$ and $F_\omega - E_\omega \leq w$, whence $F_\omega - D_\omega \leq v + w$. Thus, R satisfies the generated edge, $D \xrightarrow{v+w} F$. □

Lemma 5 *The* Lower Case *rule from Table* 1 *is sound.*

Proof Suppose that $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is an STNU that contains a lower-case edge, $A \xrightarrow{c:v} C$, and that R is an RTED-based strategy for $\mathcal{S}$ that satisfies an edge, $C \xrightarrow{u} F$, for some $F \in \mathcal{T}$ and some $u \leq 0$.[16] Contrary to the statement of the lemma, suppose that R does *not* satisfy the child edge, $A \xrightarrow{v+u} F$.

Now, since R satisfies the edge, $C \xrightarrow{u} F$, then for every situation ω, $F_\omega - C_\omega \leq u$ and, hence, $F_\omega \leq C_\omega + u \leq C_\omega$, since $u \leq 0$. However, since R does *not* satisfy the edge, $A \xrightarrow{v+u} F$, there must be some situation $\hat{\omega}$ for which $F_{\hat{\omega}} - A_{\hat{\omega}} > v + u$ and, hence, $A_{\hat{\omega}} + v + u < F_{\hat{\omega}}$.

Next, note that if the duration, $C - A$, equals its lower bound v in the situation $\hat{\omega}$ (i.e., if $C_{\hat{\omega}} = A_{\hat{\omega}} + v$), then $F_{\hat{\omega}} \leq C_{\hat{\omega}} + u = A_{\hat{\omega}} + v + u < F_{\hat{\omega}}$, which yields the contradiction, $F_{\hat{\omega}} < F_{\hat{\omega}}$. Therefore, $C - A$ must not equal v in $\hat{\omega}$. But in that case, let ω be the situation that is the same as $\hat{\omega}$ except that $C - A = v$ in ω. Then Lemma 3 gives that $A_{\text{ω}} = A_{\hat{\omega}}$ and $C_{\text{ω}} = d < C_{\hat{\omega}}$, where d is the time of the first difference between the schedules resulting from following R in the situations ω and $\hat{\omega}$. Furthermore, since $F_\omega \leq C_\omega$ in every situation ω, it follows that $F_{\text{ω}} \leq C_{\text{ω}} = d$, in which case, since $F \not\equiv C$, Lemma 3 gives that $F_{\text{ω}} = F_{\hat{\omega}}$. Thus: $F_{\hat{\omega}} = F_{\text{ω}} \leq C_{\text{ω}} + u = A_{\text{ω}} + v + u = A_{\hat{\omega}} + v + u < F_{\hat{\omega}}$, again yielding the contradiction, $F_{\hat{\omega}} < F_{\hat{\omega}}$. □

That the *Upper Case* and *Cross Case* rules are sound is given by Lemmas 6–9, below. They address different cases based on whether the parent upper-case edge is *original* or *generated*, and whether or not the associated wait time is *proper* (i.e., less than or equal to the maximum duration of the corresponding contingent link), as follows:

Lemma	Rule(s)	Upper-case parent edge
Lemma 6	*Upper case* rule	*Original*
Lemma 7	*Upper case* rule	*Generated* with *proper* wait time
Lemma 8	*Cross case* rule	*Generated* with *proper* wait time
Lemma 9	*Upper case* and *cross case* rules	*Generated* with *improper* wait time

Note that the applicability conditions for the *Cross Case* rule prevent it from being applied in the case of a parent edge that is an *original* upper-case edge.

Lemma 6 *The* Upper Case *rule, when restricted to* orginal *upper-case edges, is sound.*

Proof Suppose that $C \xrightarrow{C:-y} A$ is an original upper-case edge in an STNU graph corresponding to a contingent link, (A, x, y, C); and that R is an RTED-based strategy that satisfies the ordinary edge, $D \xrightarrow{v} C$ (i.e., for every situation ω, $C_\omega - D_\omega \leq v$).[17]

[16] To avoid confusion between the letters w and ω, the proof uses the letter u instead of w.

[17] This is an application of the *Upper Case* rule from Table 1 where $E \mapsto C$, $B \mapsto C$, and $F \mapsto A$.

Contrary to the lemma, suppose R does *not* satisfy the generated edge, $D \xrightarrow{C:v-y} A$. Then there must be some situation $\tilde{\omega}$ for which $D_{\tilde{\omega}} < A_{\tilde{\omega}} + y - v$ and $D_{\tilde{\omega}} \leq C_{\tilde{\omega}}$.

Case 1: $v < 0$. In this case, $C_{\tilde{\omega}} \leq D_{\tilde{\omega}} + v < D_{\tilde{\omega}} \leq C_{\tilde{\omega}}$, which is a contradiction.

Case 2: $v \geq 0$. First, note that if the contingent duration, $C - A$, happens to take on its maximum value in the situation $\tilde{\omega}$ (i.e., if $C_{\tilde{\omega}} = A_{\tilde{\omega}} + y$), then $D_{\tilde{\omega}} \geq C_{\tilde{\omega}} - v = A_{\tilde{\omega}} + y - v$, thereby satisfying the generated edge. On the other hand, if $C - A$ does not take on its maximum value in the situation $\tilde{\omega}$, let $\hat{\omega}$ be the same situation as $\tilde{\omega}$, except that $C - A$ *does* take on its maximum value in $\hat{\omega}$. Therefore, by Lemma 3, $A_{\tilde{\omega}} = A_{\hat{\omega}}$, and $C_{\tilde{\omega}} = d < C_{\hat{\omega}}$, where d is the time of the first difference between the schedules resulting from following R in the situations $\tilde{\omega}$ and $\hat{\omega}$. In addition, $D_{\tilde{\omega}} \leq C_{\tilde{\omega}} = d$ implies that $D_{\tilde{\omega}} = D_{\hat{\omega}}$. Therefore, $D_{\hat{\omega}} = D_{\tilde{\omega}} < A_{\tilde{\omega}} + y - v = A_{\hat{\omega}} + y - v = C_{\hat{\omega}} - v$, and hence $C_{\hat{\omega}} > D_{\hat{\omega}} + v$, contradicting that R satisfies the ordinary edge, $D \xrightarrow{v} C$. □

Lemma 7 *The* Upper Case *rule from Table 1, in the case where the parent upper-case edge is a* generated *upper-case edge whose wait time is proper, is sound.*

Proof Suppose that R is an RTED-based strategy that satisfies the ordinary edge, $D \xrightarrow{v} E$. Thus, in every situation ω, $E_\omega \leq D_\omega + v$. Suppose further that R satisfies the generated upper-case edge, $E \xrightarrow{B:u} F$, where (F, x, y, B) is the corresponding contingent link and the wait time, $-u$, satisfies $-u \leq y$.[18] Thus, in every situation ω, $E_\omega \geq F_\omega - u$ or $E_\omega > B_\omega$. Finally, contrary to the lemma, suppose that R does *not* satisfy the corresponding child edge, $D \xrightarrow{B:v+u} F$. Thus, for some situation $\tilde{\omega}$, $D_{\tilde{\omega}} < F_{\tilde{\omega}} - (v + u)$ and $D_{\tilde{\omega}} \leq B_{\tilde{\omega}}$.

Case 1: $v < 0$. In this case, $E_{\tilde{\omega}} \leq D_{\tilde{\omega}} + v < D_{\tilde{\omega}} \leq B_{\tilde{\omega}}$; hence, $E_{\tilde{\omega}} < B_{\tilde{\omega}}$. But then $E_{\tilde{\omega}} \leq D_{\tilde{\omega}} + v < (F_{\tilde{\omega}} - (v + u)) + v = F_{\tilde{\omega}} - u$ and, thus, $E_{\tilde{\omega}} < F_{\tilde{\omega}} - u$. But $E_{\tilde{\omega}} < B_{\tilde{\omega}}$ and $E_{\tilde{\omega}} < F_{\tilde{\omega}} - u$ together contradict that R satisfies the upper-case edge from E to F.

Case 2: $v \geq 0$. First, note that $E_{\tilde{\omega}} \leq D_{\tilde{\omega}} + v < F_{\tilde{\omega}} - u \leq F_{\tilde{\omega}} + y$. Thus, if $B_{\tilde{\omega}} = F_{\tilde{\omega}} + y$ (i.e., if the contingent duration, $B - F$, takes on its maximum value y), then $E_{\tilde{\omega}} < B_{\tilde{\omega}}$ which, together with $E_{\tilde{\omega}} < F_{\tilde{\omega}} - u$, contradicts that R satisfies the upper-case edge from E to F. Therefore, it must be that $B_{\tilde{\omega}} < F_{\tilde{\omega}} + y$ (i.e., $B - F$ does not take on its maximum value). In that case, let $\hat{\omega}$ be the same situation as $\tilde{\omega}$ except that $B_{\hat{\omega}} = F_{\hat{\omega}} + y > B_{\tilde{\omega}}$. Then, by Lemma 3 (with F and B playing the roles of A and C, respectively), $F_{\hat{\omega}} = F_{\tilde{\omega}}$, and $B_{\tilde{\omega}} = d < B_{\hat{\omega}}$, where d is the time of the first difference between the schedules obtained in $\tilde{\omega}$ and $\hat{\omega}$. In addition, $D_{\tilde{\omega}} \leq B_{\tilde{\omega}} = d$ implies that $D_{\tilde{\omega}} = D_{\hat{\omega}}$. As a result, $E_{\hat{\omega}} \leq D_{\hat{\omega}} + v = D_{\tilde{\omega}} + v < F_{\tilde{\omega}} - u = F_{\hat{\omega}} - u$ and, hence, $E_{\hat{\omega}} < F_{\hat{\omega}} - u$; and, since $F_{\tilde{\omega}} - u \leq F_{\tilde{\omega}} + y = F_{\hat{\omega}} + y = B_{\hat{\omega}}$, it follows that $E_{\hat{\omega}} < B_{\hat{\omega}}$. But $E_{\hat{\omega}} < F_{\hat{\omega}} - u$ and $E_{\hat{\omega}} < B_{\hat{\omega}}$ together contradict that R satisfies the upper-case edge from E to F. □

Lemma 8 *The* Cross Case *rule from Table 1, in the case where the parent upper-case edge is a* generated *upper-case edge whose wait time is proper, is sound.*

Proof Let R be an RTED-based strategy for an STNU $\mathcal{S}$; let $A \xrightarrow{c:v} C$ be a lower-case edge in the graph for $\mathcal{S}$; and let $C \xrightarrow{B:u} F$ be a *generated* upper-case edge, where the corresponding contingent link is (F, x, y, B), $B \not\equiv C$, $u \leq 0$, and $-u \leq y$. Suppose that R satisfies that upper-case edge. Thus, for each situation ω, $C_\omega \geq F_\omega - u$ or $C_\omega > B_\omega$. Finally, contrary to the statement of the lemma, suppose that R does not satisfy the corresponding child edge, $A \xrightarrow{B:v+u} F$. In that case, there must be some situation $\hat{\omega}$ for which $A_{\hat{\omega}} < F_{\hat{\omega}} - (v + u)$ and $A_{\hat{\omega}} \leq B_{\hat{\omega}}$.

[18] The letter u is used instead of the w from the *Upper Case* rule in Table 1 to avoid confusion with ω.

Case 0: $B_{\hat{\omega}} = F_{\hat{\omega}} + y$ (i.e., the duration $B - F$ takes on its maximum value in $\hat{\omega}$). Now, since R satisfies the upper-case edge from C to F, either $C_{\hat{\omega}} \geq F_{\hat{\omega}} - u$ or $C_{\hat{\omega}} > B_{\hat{\omega}}$. But $C_{\hat{\omega}} > B_{\hat{\omega}}$ leads to the following: $C_{\hat{\omega}} > B_{\hat{\omega}} = F_{\hat{\omega}} + y \geq F_{\hat{\omega}} - u$. Thus, $C_{\hat{\omega}} \geq F_{\hat{\omega}} - u$ necessarily holds. But then $A_{\hat{\omega}} - F_{\hat{\omega}} = (A_{\hat{\omega}} - C_{\hat{\omega}}) + (C_{\hat{\omega}} - F_{\hat{\omega}}) \geq (A_{\hat{\omega}} - C_{\hat{\omega}}) - u$.

Now suppose that $C_{\hat{\omega}} = A_{\hat{\omega}} + v$ (i.e., the duration $C - A$ takes on its minimum value in $\hat{\omega}$). But then $A_{\hat{\omega}} - F_{\hat{\omega}} \geq -v - u$, contradicting the choice of $\hat{\omega}$. Therefore, the duration $C - A$ must not take on its minimum value in $\hat{\omega}$. But then let $\tilde{\omega}$ be the same situation as $\hat{\omega}$, except that $C - A = v$ in $\tilde{\omega}$. Then, by Lemma 3, $A_{\tilde{\omega}} = A_{\hat{\omega}}$, and $C_{\tilde{\omega}} = d < C_{\hat{\omega}}$, where d is the time of the first difference between the schedules obtained by following R in $\tilde{\omega}$ and $\hat{\omega}$.

Sub-Case 0.A: $C_{\tilde{\omega}} \geq F_{\tilde{\omega}} - u$. But then $d = C_{\tilde{\omega}} \geq F_{\tilde{\omega}} - u \geq F_{\tilde{\omega}}$ which, by Lemma 3, implies that $F_{\tilde{\omega}} = F_{\hat{\omega}}$. But then: $A_{\hat{\omega}} - F_{\hat{\omega}} = A_{\tilde{\omega}} - F_{\tilde{\omega}} = (A_{\tilde{\omega}} - C_{\tilde{\omega}}) + (C_{\tilde{\omega}} - F_{\tilde{\omega}}) = -v + (C_{\tilde{\omega}} - F_{\tilde{\omega}}) \geq -v - u$, contradicting the choice of $\hat{\omega}$.
Sub-Case 0.B: $C_{\tilde{\omega}} > B_{\tilde{\omega}}$. Here, $d = C_{\tilde{\omega}} > B_{\tilde{\omega}}$ implies that $B_{\tilde{\omega}} = B_{\hat{\omega}}$, by Lemma 3. Similarly, $d = C_{\tilde{\omega}} > B_{\tilde{\omega}} = B_{\hat{\omega}} > F_{\hat{\omega}}$ implies that $F_{\tilde{\omega}} = F_{\hat{\omega}}$. But then $A_{\hat{\omega}} - F_{\hat{\omega}} = A_{\tilde{\omega}} - F_{\tilde{\omega}} = (A_{\tilde{\omega}} - C_{\tilde{\omega}}) + (C_{\tilde{\omega}} - F_{\tilde{\omega}}) = -v + (C_{\tilde{\omega}} - F_{\tilde{\omega}}) > -v + (B_{\tilde{\omega}} - F_{\tilde{\omega}}) = -v + y \geq -v - u$, contradicting the choice of $\hat{\omega}$.

Case 1: $B_{\hat{\omega}} < F_{\hat{\omega}} + y$ (i.e., the duration $B - F$ does not *take on its maximum value in $\hat{\omega}$).* Let ω' be the same situation as $\hat{\omega}$, except that $B - F$ takes on its maximum value in ω'. That is: $B_{\omega'} = F_{\omega'} + y$. Then, by Lemma 3, $F_{\hat{\omega}} = F_{\omega'}$, and $B_{\hat{\omega}} = d < B_{\omega'}$, where d is the time of the first difference between the corresponding schedules. In addition, $A_{\hat{\omega}} \leq B_{\hat{\omega}} = d$ implies that $A_{\omega'} = A_{\hat{\omega}}$ and, since the duration of $C - A$ is the same in $\hat{\omega}$ and ω', $C_{\omega'} = C_{\hat{\omega}}$. Thus, F, A and C all have the same values in $\hat{\omega}$ and ω'.

Now, exactly as in the first part of Case 0, $C - A = v$ in ω' leads to a contradiction. Therefore, $C_{\omega'} - A_{\omega'} > v$. So, let $\tilde{\omega}$ be the same situation as ω' except that $C_{\tilde{\omega}} - A_{\tilde{\omega}} = v$. Then, by Lemma 3, $A_{\tilde{\omega}} = A_{\omega'}$, and $C_{\tilde{\omega}} = e < C_{\omega'}$, where e is the time of the first difference between the corresponding schedules. Thus, $A_{\tilde{\omega}} = A_{\omega'} = A_{\hat{\omega}}$.

Case 1.A: $C_{\tilde{\omega}} \geq F_{\tilde{\omega}} - u$. But then, $e = C_{\tilde{\omega}} \geq F_{\tilde{\omega}} - u \geq F_{\tilde{\omega}}$, which by Lemma 3 implies that $F_{\tilde{\omega}} = F_{\omega'} = F_{\hat{\omega}}$. Thus, $A_{\hat{\omega}} - F_{\hat{\omega}} = A_{\tilde{\omega}} - F_{\tilde{\omega}} = (A_{\tilde{\omega}} - C_{\tilde{\omega}}) + (C_{\tilde{\omega}} - F_{\tilde{\omega}}) = -v + (C_{\tilde{\omega}} - F_{\tilde{\omega}}) \geq -v - u$, contradicting the choice of $\hat{\omega}$.
Case 1.B: $B_{\tilde{\omega}} < C_{\tilde{\omega}}$. Then, $B_{\tilde{\omega}} < C_{\tilde{\omega}} = e$ implies, by Lemma 3, that $B_{\tilde{\omega}} = B_{\omega'}$. Similarly, $F_{\tilde{\omega}} < B_{\tilde{\omega}} < e$ implies that $F_{\tilde{\omega}} = F_{\omega'}$. Thus, $F_{\tilde{\omega}} = F_{\omega'} = F_{\hat{\omega}}$. But then: $A_{\hat{\omega}} - F_{\hat{\omega}} = A_{\tilde{\omega}} - F_{\tilde{\omega}} = (A_{\tilde{\omega}} - C_{\tilde{\omega}}) + (C_{\tilde{\omega}} - F_{\tilde{\omega}}) = -v + (C_{\tilde{\omega}} - F_{\tilde{\omega}}) > -v + (B_{\tilde{\omega}} - F_{\tilde{\omega}}) = -v + (B_{\omega'} - F_{\omega'}) = -v + y \geq -v - u$, contradicting the choice of $\hat{\omega}$. □

Lemma 9 *The* Upper Case *and* Cross Case *rules from Table 1 are sound in the case where the parent upper-case edge is a generated edge whose wait time is* not *proper.*

Proof Let e be any *generated* upper-case edge, $E \xrightarrow{B:u} F$, for some contingent link, (F, x, y, B), where the wait time is improper (i.e., $-u > y$). Since e is a generated upper-case edge, it must have been generated from some prior application of the *Upper Case* or *Cross Case* rule, using some parent upper-case edge e'. And if e' happens to be a generated edge, it must have been generated from another prior application of the *Upper Case* or *Cross Case* rule using some parent upper-case edge e''. This process continues until the *original* upper-case edge, $B \xrightarrow{B:-y} F$, is eventually reached, as illustrated in Fig. 6, where the original upper-case edge lies vertically on the righthand side; each generated upper-case edge in the sequence is dashed; the *lengths* of the lefthand parent edges (i.e., the lower-case or ordinary parent edges) are notated as $v_1, \ldots, v_r$; and the upper-case edge

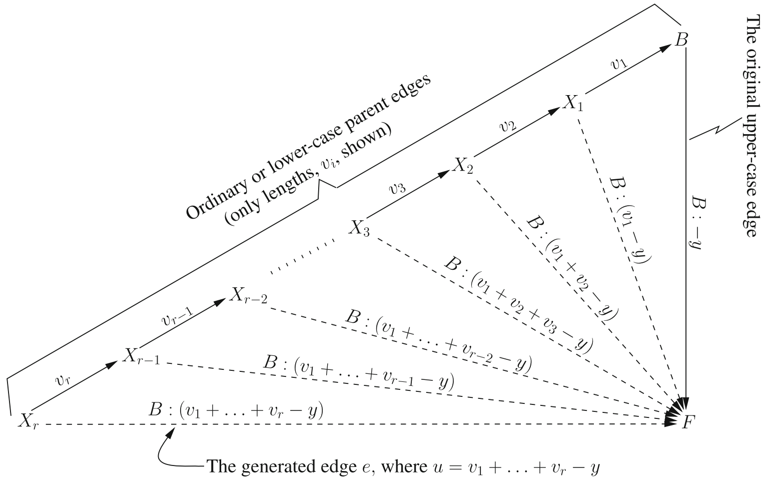

Ordinary or lower-case parent edges
(only lengths, v_i, shown)
The original upper-case edge
B
v_1
X_1
v_2
X_2
v_3
X_3
X_{r-2}
v_{r-1}
X_{r-1}
v_r
X_r
F
$B : -y$
$B : (v_1 - y)$
$B : (v_1 + v_2 - y)$
$B : (v_1 + v_2 + v_3 - y)$
$B : (v_1 + \ldots + v_{r-2} - y)$
$B : (v_1 + \ldots + v_{r-1} - y)$
$B : (v_1 + \ldots + v_r - y)$
The generated edge e, where $u = v_1 + \ldots + v_r - y$

Fig. 6 A sequence of applications of the *Upper Case* or *Cross Case* rules discussed in the Proof of Lemma 9

e, which could be any one of the dashed edges, is shown as the last generated edge in the sequence.

The rest of this proof addresses sequences of the kind shown in Fig. 6, proving that for any RTED-based strategy R, if R satisfies all of the *ordinary* parent edges shown in the figure, then R necessarily satisfies all of the generated upper-case edges. For convenience, let $X_0 = B$. Then for each $i \in \{0, 1, 2, \ldots, r-1\}$, the edge from X_{i+1} to X_i is either a lower-case or ordinary edge that is the *lefthand* parent edge for the corresponding application of the *Upper Case* or *Cross Case* rule. The proof analyzes the chain of LO-edges starting at $X_0 = B$ and working backward toward X_r. All lower-case edges considered below have contingent time-points that are distinct from B, as required by the *Cross Case* rule.

Case 1: A chain of ordinary edges from X_j to $X_0 = B$. The *No Case* rule can be used to collapse the path from X_j to X_0 into a single ordinary edge, $X_j \xrightarrow{v} B$, whose length is $v = v_j + v_{j-1} + \cdots + v_2 + v_1$. Since the *No Case* rule is sound (cf. Lemma 4), R must satisfy this edge. Next, applying the *Upper Case* rule to the edges, $X_j \xrightarrow{v} B$ and $B \xrightarrow{B:-y} F$, generates the upper-case edge, $X_j \xrightarrow{B:v-y} F$, which R must satisfy by Lemma 6. Note that the wait time, $y - v$, for this generated upper-case edge may not be proper.

Case 2: A chain consisting of a single lower-case edge followed by one or more ordinary edges, terminating at $B = X_0$. Let $A \xrightarrow{c:s} C$ be the lower-case edge that starts the chain, and let the rest of the chain be as described in Case 1, using $C = X_j$. As in Case 1, the ordinary edges from X_j to $X_0 = B$ can be collapsed into a single ordinary edge, $X_j \xrightarrow{v} B$, that must be satisfied by R.

Case 2a: $v > 0$. As in Case 1, the *Upper Case* rule can be used to generate the edge, $X_j \xrightarrow{B:v-y} F$, which must be satisfied by R. But here the wait time, $-v + y$, must be proper since $v > 0$. Therefore, the *Cross Case* rule can be applied to the edges, $A \xrightarrow{c:s} X_j$ and $X_j \xrightarrow{B:v-y} F$, to generate the upper-case edge, $A \xrightarrow{B:s+v-y} F$, whose wait time, $-s-v+y$, must also be proper since s and v are both positive.

Case 2b: $v \leq 0$. In this case, the ordinary path from X_j to X_0 has negative length; thus, some prefix of that path—say the path from X_j to X_h, where $j > h \geq 0$—constitutes an extension sub-path for the lower-case edge and, thus, can be used to reduce it away, resulting in the ordinary edge, $A \xrightarrow{s+v_j+\cdots+v_{h+1}} X_h$. This edge must be satisfied by R since the *No Case* and *Lower Case* rules are sound (cf. Lemmas 4 and 5). Then, the ordinary path consisting of this edge followed by all of the edges from X_h to $X_0 = B$ can be collapsed into a single ordinary edge by the *No Case* rule, as in Case 1. Then, an application of the *Upper Case* rule generates an upper-case edge from A to F that must be satisfied by R (cf. Lemma 6), although its wait time may be improper.

Cases 3 and 4, below, are similar to Cases 1 and 2, respectively, except that, instead of the original upper-case edge, $B \xrightarrow{B:-y} F$, they deal with any generated upper-case edge, $X_0 \xrightarrow{B:w} F$, that is satisfied by the strategy R, and whose wait time, $-w$, is proper (i.e., $-w \leq y$). In these cases, X_0 is some time-point other than B.

Case 3: A sequence of ordinary edges terminating at X_0. This case is identical to Case 1, except that it is Lemma 7 instead of Lemma 6 that ensures that the resulting upper-case edge must be satisfied by the strategy R.

Case 4: A lower-case edge followed by zero or more ordinary edges terminating at X_0. Let $A \xrightarrow{c:s} C$ be the lower-case edge, whose length s is necessarily positive.

Case 4a: $C = X_0$ *(i.e., zero ordinary edges).* Here, one application of the *Cross Case* rule generates the upper-case edge, $A \xrightarrow{B:s+w} F$, whose wait time, $-s - w$, is proper since $-s - w < -w \leq y$, since $s > 0$. R must satisfy this child edge by Lemma 8.

Case 4b: The lower-case edge followed by a non-negative-length path consisting solely of ordinary edges. In this case, let $C = X_j$, and let $\mathcal{P}$ be the non-negative length path from $C = X_j$ to X_0, all of whose edges are ordinary. Now, the *No Case* rule can be used to transform $\mathcal{P}$ into a single ordinary edge, $C \xrightarrow{v} X_0$, where $v = |\mathcal{P}| \geq 0$. By Lemma 4, R must satisfy this edge.

Next, an application of the *Upper Case* rule to the edges, $C \xrightarrow{v} X_0$ and $X_0 \xrightarrow{B:w} F$, generates the upper-case edge, $C \xrightarrow{B:v+w} F$, whose wait time, $-v - w \leq -w \leq y$, is proper, since $v \geq 0$. Lemma 7 ensures that R must satisfy this edge. Finally, the *Cross Case* rule can be used to combine the edges, $A \xrightarrow{c:s} C$ and $C \xrightarrow{B:v+w} F$, to generate the upper-case edge, $A \xrightarrow{B:s+v+w} F$, which, by Lemma 8, must be satisfied by R. Furthermore, its wait time, $-s - v - w$, must be proper since $-s < 0$, $-v \leq 0$ and $-w \leq y$.

Case 4c: The lower-case edge followed by a negative-length path, $\mathcal{P}$, *consisting solely of ordinary edges.* Since the ordinary path $\mathcal{P}$ has negative length, some prefix of $\mathcal{P}$—say the path from $C = X_j$ to X_h—constitutes an extension sub-path for the lower-case edge. The *No Case* rule can be used to reduce this extension sub-path to a single ordinary edge from C to X_h, which by Lemma 4 must be satisfied by R. A single application of the *Lower Case* rule, which is also sound (cf. Lemma 5), then yields an ordinary edge from A to X_h, that R must also satisfy. The *No Case* rule can then be used to generate a single ordinary edge, $A \xrightarrow{\delta} X_0$, that must be satisfied by R, and whose length is: $\delta = s + v_j + \cdots + v_1$. Finally, the *Upper Case* rule can be applied to $A \xrightarrow{\delta} X_0$ and $X_0 \xrightarrow{B:w} F$, to generate the upper-case edge, $A \xrightarrow{B:\delta+w} F$, that must be satisfied by R (cf. Lemma 8). However, the wait time, $-\delta - w$, may not be proper.

Finally, recall the arbitrary sequence of applications of the *Upper Case* and *Cross Case* rules shown in Fig. 6, under the assumption that the strategy R satisfies all of the ordinary edges that appear in the path from X_r to $X_0 = B$. First, if *all* of the edges from X_r to $X_0 = B$ are ordinary, then Case 1 ensures that all of the generated upper-case edges are satisfied by R. Second, working backward from X_0 toward X_r, let e be the first lower-case edge encountered—say that e is a lower-case edge from X_{j+1} to X_j. By Case 2, the generated upper-case edge is satisfied by R, but its wait time need not be proper. However, if it is not proper, recall from the Proof of Case 2 that the lower-case edge, e, can be reduced away to an ordinary edge, resulting in only ordinary edges from X_{j+1} to X_0. Now, if the result of the above is a path of ordinary edges from X_j to X_0, then the same line of reasoning can be used to handle the next lower-case edge encountered on the way to X_r. But if the result of the above is a generated upper-case edge from X_j to F whose wait time is proper, then Cases 3 and 4 can instead be used to process the next lower-case edge encountered. Since this processing again yields the same kinds of results—either a sequence of ordinary edges or a generated child edge with a proper wait time—it follows that all of the upper-case edges generated by this process are satisfied by R. □

Lemma 10 *The* Label Removal *rule from Table 1 is sound.*

Proof Let $D \xrightarrow{C:z} A$ be any upper-case edge, and (A, x, y, C) the corresponding contingent link, where $z \geq -x$. Let R be any RTED-based strategy that satisfies that UC edge (i.e., for any situation ω, $D_\omega \geq A_\omega - z$ or $D_\omega > C_\omega$). Now, if $D_\omega > C_\omega$, then $D_\omega > C_\omega \geq A_\omega + x \geq A_\omega - z$. Thus, $D_\omega \geq A_\omega - z$ invariably holds. Since ω was arbitrary, it follows that R satisfies the unconditional constraint, $A - D \leq z$, that corresponds to the ordinary edge, $D \xrightarrow{z} A$, generated by the *Label Removal* rule. □

Theorem 3 *Each edge obtained from any* sequence *of applications of the edge-generation rules in Table 1 must be satisfied by any reliable RTED-based strategy.*

Proof Let R be a reliable RTED-based strategy for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$. By definition, R satisfies all of the constraints in $\mathcal{C}$. Thus, R satisfies all OU$_g$ edges in the original graph $\mathcal{G}$ for $\mathcal{S}$. Furthermore, this property is recursively maintained by any application of the edge-generation rules, all of which are sound (cf. Lemmas 4–10). □

For example, the upper-case edge, $A_2 \xrightarrow{C_1:-4} A_1$ in Fig. 2b, must be satisfied by any reliable RTED-based strategy for the given STNU. It is not hard to verify that the sample execution strategy discussed earlier satisfies the corresponding conditional wait constraint (i.e., "While C_1 remains unexecuted, A_2 must wait at least 4 units after A_1").

Finally, Theorem 4, below, states that an STNU that has an SRN loop must *not* be dynamically controllable, which is the first half of the fundamental theorem for STNUs.

Theorem 4 (SRN Loop ⇒ non-DC [16]) *Suppose that $\mathcal{S}$ is an STNU whose graph $\mathcal{G}$ contains an SRN loop. Then $\mathcal{S}$ is not dynamically controllable.*

Proof Let $\mathcal{S}$ be an STNU whose graph, $\mathcal{G}$, contains an SRN loop, $\mathcal{P}$. Since $\mathcal{P}$ is semi-reducible, the rules from Table 1 can be used to transform $\mathcal{P}$ into a loop, $\mathcal{P}'$, that contains only ordinary or upper-case edges. Now, any ordinary edge that is followed by an upper-case edge can effectively be "reduced away" by the *Upper Case* rule, resulting in a single upper-case edge. And the *Label Removal* rule can remove the upper-case label from any upper-case edge whose wait time is smaller than the lower bound on the corresponding contingent link.

And any sequence of ordinary edges can be transformed into a single ordinary edge by the *No Case* rule. Applying these rules in this way, in any combination, must eventually terminate since each application of the *No Case* or *Upper Case* rule decreases the number of edges in the loop; and each application of the *Label Removal* rule eliminates one upper-case label. The end result will be a loop $\mathcal{P}''$ that (1) consists of a single ordinary edge of the form, $X \xrightarrow{\delta} X$, where $\delta < 0$; or (2) consists solely of upper-case edges each of whose wait time is greater than the lower bound on the corresponding contingent link. By Theorem 3, any reliable RTED-based strategy for $\mathcal{S}$ must satisfy the edges in $\mathcal{P}''$. However, in the first case, the edge represents the constraint, $X - X \leq \delta < 0$, which cannot be satisfied. For the second case, let R be any reliable strategy for $\mathcal{S}$; let ω be any situation; and let $X \xrightarrow{C:-u} A$ be an upper-case edge for some contingent link (A, x, y, C), where $u > x$. If the edge is an original upper-case edge, then the RTED-based semantics ensures that $X_\omega - A_\omega \geq x > 0$. Otherwise, R satisfies the generated edge by either $X_\omega \geq A_\omega + u > A_\omega + x$ or $X_\omega > C_\omega \geq A_\omega + x$. Thus, in either case, $X_\omega - A_\omega > 0$. Therefore, a cycle of upper-case edges implies a cycle of constraints of the form, $X_1 - X_2 > 0$, $X_2 - X_3 > 0$, …, $X_n - X_1 > 0$. Summing these constraints yields $0 > 0$, which is impossible to satisfy. Thus, in both cases, no reliable RTED-based strategy exists for $\mathcal{S}$. □

4 Fundamental theorem for STNUs, part 2

This section addresses the second half of the fundamental theorem for STNUs: if $\mathcal{S}$ is any STNU whose graph $\mathcal{G}$ does not contain any SRN loops, then $\mathcal{S}$ must be dynamically controllable. This result is proven in steps. First, $\mathcal{G}$ having no SRN loops is shown to imply that the notion of *shortest* semi-reducible path is well defined for $\mathcal{G}$ and, in addition, that the *all-pairs, shortest-semi-reducible-paths* (APSSRP) matrix, $\mathcal{D}^*$, can be computed in polynomial time. Next, $\mathcal{D}^*$ is used to generate RTEDs on the fly, in any incrementally revealed situation, effectively defining an RTED-based execution strategy for $\mathcal{S}$. Finally, that strategy is proved to be a reliable strategy for $\mathcal{S}$, thereby confirming that $\mathcal{S}$ is dynamically controllable. As will be seen, proving these results requires addressing important details that have been glossed over in the literature.

4.1 Preliminary definitions and results

Lemma 11 *If $\mathcal{P}$ is an LO-path in an STNU graph that is a* loop *whose length is non-positive, then $\mathcal{P}$ must be semi-reducible.*

Proof Suppose that $\mathcal{P}$ contains at least one lower-case edge. Since $\mathcal{P}$ is a loop containing only lower-case and ordinary edges, the *No Case* rule can be used to transform $\mathcal{P}$ into a loop $\mathcal{P}'$ such that at most one ordinary edge separates any pair of consecutive lower-case edges; and $|\mathcal{P}'| = |\mathcal{P}| \leq 0$. (If $\mathcal{P}$ has only one lower-case edge, then $\mathcal{P}'$ will have only two edges: one lower-case and one ordinary.) Since each lower-case edge has positive length, $\mathcal{P}'$ must contain at least one ordinary edge, E, that has negative length. Let E_1 be the lower-case edge immediately preceding E in $\mathcal{P}'$. Then the *Lower Case* rule can be used to replace E_1 and E by some ordinary edge, resulting in a loop $\mathcal{P}''$ that has one fewer lower-case edges than $\mathcal{P}$. Since such transformations preserve lengths, $|\mathcal{P}''| = |\mathcal{P}'| = |\mathcal{P}| \leq 0$. Hence, continuing in this way, $\mathcal{P}$ can be transformed into a loop containing only ordinary edges; thus, $\mathcal{P}$ is semi-reducible. □

Definition 34 (*Signature of an edge*) The *signature* of an edge in an STNU graph is a term of the form, `type(args)`, where `type` is one of `Ord`, `LC`, or `UC`; and `args` identifies certain time-points that are relevant to the edge, as follows:

Edge type	Sample edge	Signature
Ordinary	$X \xrightarrow{\delta} Y$	$\texttt{Ord}(X, Y)$
Lower-Case	$A \xrightarrow{c:x} C$	$\texttt{LC}(C)$
Upper-Case	$V \xrightarrow{C:w} A$	$\texttt{UC}(V, C)$

Note that for any kind of edge, its length is not part of its signature. In addition, since the edge-generation rules generate only OU-edges, an STNU having K contingent links will always have exactly K lower-case edges, one for each contingent link; thus, the signature of a lower-case edge need only specify the corresponding contingent time-point. Finally, since the target of any upper-case edge, whether generated or original, must be the activation time-point for the corresponding contingent link, the signature for an upper-case edge need only specify the source time-point V, and the upper-case label C. Given these observations, it follows that in a network with N time-points and K contingent links, there are at most N^2 distinct signatures for ordinary edges, at most NK distinct signatures for upper-case edges, and exactly K distinct signatures for lower-case edges. Thus, there are at most $N^2 + NK + K$ distinct signatures overall.

The following theorem enables subsequent proofs to effectively side-step the problems introduced by *breaches* in semi-reducible paths (cf. Definition 31).

Theorem 5 *Let $\mathcal{G}$ be an STNU graph; and let $\mathcal{P}$ be any semi-reducible path from X to Y in $\mathcal{G}$. Then there exists a* breach-free *semi-reducible path $\mathcal{P}'$, also from X to Y, that is obtained by extracting zero or more non-negative sub-loops from $\mathcal{P}$. Hence, $|\mathcal{P}'| \leq |\mathcal{P}|$.*

Proof Let e be an occurrence of the lower-case edge, $A \xrightarrow{c:x} C$, in a semi-reducible path $\mathcal{P}$ whose extension sub-path, $\mathcal{P}_e$, contains a breach edge: $V \xrightarrow{C:-w} A$. If there multiple instances of breaches in $\mathcal{P}$, choose the one that is *outermost* with that property.[19] Thus, any extension sub-paths in $\mathcal{P}$ that contain $\mathcal{P}_e$ are breach-free. Let B denote the breach edge; and let $\mathcal{P}_B$ denote the prefix of $\mathcal{P}_e$ that contains all of the edges in $\mathcal{P}_e$ up to and including B. Now, since $\mathcal{P}$ is semi-reducible, it follows that $|\mathcal{P}_B| \geq -x$. (Otherwise, the upper-case label from the breach could not be removed, which would block the use of the *Cross Case* rule in the canonical elimination of e.) Let S be the path that consists of e followed by $\mathcal{P}_B$. Note that S is a loop from A (the source of the lower-case edge e) to A (the destination of the breach edge B). Furthermore, $|S| = |e| + |\mathcal{P}_B| \geq x + (-x) = 0$. Therefore, since any extension sub-paths that contain $\mathcal{P}_e$ are breach-free, extracting S from $\mathcal{P}$ cannot introduce any breaches into such extension sub-paths; it can only potentially shrink the number of edges in them. Thus, extracting S from $\mathcal{P}$ cannot affect its semi-reducibility. Since this process results in a path having fewer edges, repeating it to remove any other breaches must eventually terminate in the desired path $\mathcal{P}'$. □

[19] Morris showed that extension sub-paths within the same path $\mathcal{P}$ must either be disjoint or nested, one inside the other [13,16]. An *outermost* extension sub-path is one that is not nested inside any other.

4.2 Ensuring that the notion of *shortest* semi-reducible path is well defined

In the case of an STN, if its graph has no negative loops, then it is easy to show that the notion of shortest path is well defined. However, the analogous result for STNUs—if its graph has no semi-reducible negative loops, then the notion of shortest semi-reducible path is well defined—is much more challenging to prove. In particular, the intuitively appealing claim that one can discard any non-shortest edges during the process of edge generation is true, but requires great care to prove. The intuitive appeal of this claim is undoubtedly why its proof has not before appeared in the literature.

Definition 35 (*Shortest edges*) For any set $\mathcal{E}$ of edges, let $\texttt{shortest}(\mathcal{E})$ be the set of edges obtained by keeping, for each signature, only the shortest edge from $\mathcal{E}$ with that signature.

Below, rounds of edge generation are defined. Note that lower-case edges are kept separate because the edge-generation rules can only generate OU-edges.

Definition 36 (*Rounds of edge generation*) Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU having N time-points and K contingent links; let $\mathcal{G}$ be the corresponding graph; let $\mathcal{E}_L$ be the set of K lower-case edges in $\mathcal{G}$; and let $\mathcal{E}_{ou}$ be the *original* OU-edges in $\mathcal{G}$ (i.e., the edges in $\mathcal{C}$ together with the orginal upper-case edges in $\mathcal{G}$). The rounds of edge generation for $\mathcal{G}$ are defined as follows.

$\texttt{children}(\mathcal{E}_L, \mathcal{E})$: For *any* set $\mathcal{E}$ of OU-edges, $\texttt{children}(\mathcal{E}_L, \mathcal{E})$ is the set of edges obtained by *non-recursively* applying the edge-generation rules from Table 1 to all possible combinations of parent edges drawn from $\mathcal{E}_L \cup \mathcal{E}$. Note that, like $\mathcal{E}$, $\texttt{children}(\mathcal{E}_L, \mathcal{E})$ contains only OU-edges.

Edge-Generation Rounds, $(\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_2, \ldots)$: First, $\mathcal{E}_0 = \mathcal{E}_{\text{ou}}$ is the set of all *original* OU-edges in $\mathcal{G}$. Then, for each $i \geq 0$, $\mathcal{E}_{i+1} = \mathcal{E}_i \cup \texttt{children}(\mathcal{E}_L, \mathcal{E}_i)$ contains the original OU-edges, plus all OU-edges obtained from $i + 1$ rounds of edge generation.

Shortest-Only Edge-Generation Rounds, $(\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \ldots)$: First, $\mathcal{F}_0 = \mathcal{E}_{\text{ou}}$. Then, for each $i \geq 0$, $\mathcal{F}_{i+1} = \texttt{shortest}(\mathcal{F}_i \cup \texttt{children}(\mathcal{E}_L, \mathcal{F}_i))$ contains all of the OU-edges obtained by $i + 1$ rounds of edge generation, where any *non-shortest* edges are discarded *at the end of each round.*

For example, if $\mathcal{E}_0$ is the set of four OU-edges represented by the solid arrows in Fig. 2b (not including the two lower-case edges), then $\mathcal{E}_1$ contains the edges in $\mathcal{E}_0$ together with the (dashed) edges from A_1 to X, and from C_2 to A_1; and $\mathcal{E}_2$ contains all of those edges together with the (dashed) edge from A_2 to A_1.

Note that the sets, $\mathcal{E}_i$, may contain an inordinate number of redundant edges (i.e., edges that are weaker than other edges in $\mathcal{E}_i$ having the same signature). As a result, the number of edges in those sets can grow quite quickly—in fact, exponentially. In contrast, since each $\mathcal{F}_i$ contains at most one edge per signature, each $|\mathcal{F}_i|$ is bounded above by $N^2 + NK + K$, which is constant for a given network. Thus, the process of computing shortest semi-reducible paths can be made vastly more efficient if all *non-shortest* generated edges can be discarded. Theorem 6 ensures that this is the case.

Theorem 6 (Non-shortest edges can be discarded during edge generation) *Let $\mathcal{G}$ be an STNU graph; let $\mathcal{E}_L$ be the set of lower-case edges in $\mathcal{G}$; let $\mathcal{F}_0, \mathcal{F}_1, \ldots$ be the sequence of* shortest-only *edge-generation rounds defined above, where non-shortest edges are discarded after each round; and let $\mathcal{P}$ be any semi-reducible path in $\mathcal{G}$ from X to Y. Then there exists a semi-reducible path $\mathcal{P}^\dagger$ in $\mathcal{G}$ from X to Y such that $|\mathcal{P}^\dagger| \leq |\mathcal{P}|$, and $\mathcal{P}^\dagger$ can be transformed* *into an OU-path $\mathcal{P}_{\text{ou}}$ such that (1) for some k, the set $\mathcal{F}_k$ contains all of the edges in $\mathcal{P}_{\text{ou}}$; and (2) each OU-edge encountered during the transformation of $\mathcal{P}^\dagger$ into $\mathcal{P}_{\text{ou}}$ belongs to some $\mathcal{F}_i$ where $i \leq k$.*

Proof Let $\mathcal{P}$ be any semi-reducible path in $\mathcal{G}$ from some X to some Y. By Theorem 5, there is a *breach-free* semi-reducible path, $\mathcal{P}'$, from X to Y that can be obtained by extracting zero or more non-negative loops from $\mathcal{P}$. Thus, $|\mathcal{P}'| \leq |\mathcal{P}|$. The rest of the proof constructs a sequence of paths, $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$, where:

(1) for each i, $\mathcal{P}_i$ is a breach-free, semi-reducible path from X to Y;
(2) for each i, all of the OU-edges in $\mathcal{P}_i$ belong to $\mathcal{F}_i$;
(3) $|\mathcal{P}_k| \leq |\mathcal{P}_{k-1}| \leq \ldots \leq |\mathcal{P}_1| \leq |\mathcal{P}_0| \leq |\mathcal{P}|$; and
(4) $\mathcal{P}_k$ is an OU-path.

The construction relies on the following fact.

> *Fact.* Suppose that $\mathcal{Q}$ is a breach-free, semi-reducible path from X to Y, and E is an OU-edge in $\mathcal{Q}$. Let $\mathcal{Q}_s$ be the same as $\mathcal{Q}$ except that E is replaced by a shorter edge E_s having the same signature as E. Then $\mathcal{Q}_s$ is breach-free and semi-reducible.
>
> *Proof of Fact.* Suppose that the insertion of the edge E_s introduced a breach into an extension sub-path, $\mathcal{P}_e$, for some lower-case edge e in $\mathcal{Q}$. Since $|E_s| < |E|$, the moat edge for e in $\mathcal{Q}_s$ is either E_s or some other edge from $\mathcal{P}_e$. But, since $\mathcal{Q}$ is breach-free, the only way a breach could be introduced into $\mathcal{Q}_s$ is by E_s being the offending breach edge. But E and E_s have the same signature, implying that E would have been a breach in $\mathcal{Q}$, which is a contradiction. Thus, $\mathcal{Q}_s$ must be breach-free. Furthermore, $\mathcal{Q}_s$ must also be semi-reducible since replacing E by E_s can, at worst, only decrease the number of edges in any breach-free extension sub-path containing E, which cannot prevent it from being able to reduce away the corresponding lower-case edge. Thus, $\mathcal{Q}_s$ is both breach-free and semi-reducible.

Base Case. Let $\mathcal{P}_0 = \mathcal{P}'$, which is a breach-free, semi-reducible path from X to Y. In addition, $|\mathcal{P}_0| = |\mathcal{P}'| \leq |\mathcal{P}|$. Finally, since all of the edges in $\mathcal{P}'$ belong to $\mathcal{G}$, it follows that all of the OU-edges in $\mathcal{P}_0$ belong to $\mathcal{F}_0$.

Recursive Case. Suppose that $i \geq 0$ and $\mathcal{P}_i$ is a breach-free, semi-reducible path from X to Y, all of whose OU-edges belong to $\mathcal{F}_i$. If $\mathcal{P}_i$ is an OU-path, then let $k = i$ and the construction is completed. Therefore, consider the case where $\mathcal{P}_i$ has at least one lower-case edge. Below, the path $\mathcal{P}_{i+1}$ is constructed in stages: from $\mathcal{P}_i$ to $\mathcal{P}_i'$ to $\mathcal{P}_i''$ to $\mathcal{P}_{i+1}$. It is shown that $\mathcal{P}_{i+1}$ is a breach-free, semi-reducible path from X to Y, all of whose edges belong to $\mathcal{F}_{i+1}$, and such that $|\mathcal{P}_{i+1}| \leq |\mathcal{P}_i|$.

To begin, since $\mathcal{P}_i$ is semi-reducible, there is a (not necessarily unique) sequence of transformations in which each lower-case edge in $\mathcal{P}_i$ is canonically eliminated (i.e., reduced away) by its corresponding extension sub-path. Restrict attention to the *first* step in this sequence, whereby a child OU-edge E replaces its (one or two) parent edges. Let $\mathcal{P}_i'$ be the path that is the same as $\mathcal{P}_i$ except that the parent edge(s) have been replaced by E, as illustrated previously in Fig. 3. Since the edge-generation rules preserve length, $|\mathcal{P}_i'| = |\mathcal{P}_i|$. In addition, $\mathcal{P}_i'$ must be semi-reducible since it represents the first step in the transformation of $\mathcal{P}_i$ into an OU-path. Furthermore, the canonical elimination of lower-case edges never introduces breaches; thus, $\mathcal{P}_i'$ is breach-free.

Next, since every OU-edge in $\mathcal{P}_i$ belongs to $\mathcal{F}_i$, it follows that $E \in \texttt{children}(\mathcal{E}_L, \mathcal{F}_i)$. However, it may happen that $E \notin \texttt{shortest}(\mathcal{F}_i \cup \texttt{children}(\mathcal{E}_L, \mathcal{F}_i))$ (i.e., $E \notin \mathcal{F}_{i+1}$).

In such a case, there must be an edge $E_s \in \mathcal{F}_{i+1}$ that has the same signature as E but such that $|E_s| < |E|$. Let $\mathcal{P}''_i$ be the same as $\mathcal{P}'_i$ except that E is replaced by E_s. By the fact proven earlier, $\mathcal{P}''_i$ must be breach-free and semi-reducible. In addition, $|\mathcal{P}''_i| \leq |\mathcal{P}'_i| = |\mathcal{P}_i|$. On the other hand, if $E \in \mathcal{F}_{i+1}$, then let $\mathcal{P}''_i = \mathcal{P}'_i$, and note that the same properties hold.

At this point, $\mathcal{P}''_i$ has all of the desired properties, except that its remaining OU-edges are only guaranteed to belong to $\mathcal{F}_i$, not necessarily to $\mathcal{F}_{i+1}$. However, repeated use of the fact proven earlier ensures that replacing each such occurrence of a non-shortest OU-edge by its shorter counterpart from $\mathcal{F}_{i+1}$ will preserve the breach-free and semi-reducible properties of the path. Therefore, let $\mathcal{P}_{i+1}$ be the path obtained from $\mathcal{P}''_i$ by replacing all non-shortest OU-edges, if any, by their shorter counterparts from $\mathcal{F}_{i+1}$. Then $\mathcal{P}_{i+1}$ is a breach-free, semi-reducible path from X to Y such that $|\mathcal{P}_{i+1}| \leq |\mathcal{P}''_i| \leq |\mathcal{P}'_i| = |\mathcal{P}_i|$.

Regarding termination, note that the original path $\mathcal{P}_0 = \mathcal{P}'$ has only finitely many edges and upper-case labels, and that each transformation from $\mathcal{P}_i$ to $\mathcal{P}'_i$ either decreases the number of edges in the path by one or removes an upper-case label from an edge. Therefore, the iterative process of creating the paths in the sequence, $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$, must eventually terminate. Thus, there must be a smallest integer k such that $\mathcal{P}_k$ is an OU-path from X to Y that is breach-free, semi-reducible, all of whose edges belong to $\mathcal{F}_k$, and such that $|\mathcal{P}_k| \leq |\mathcal{P}_{k-1}| \leq \ldots \leq |\mathcal{P}_1| \leq |\mathcal{P}_0| = |\mathcal{P}'| \leq |\mathcal{P}|$. Therefore, $\mathcal{P}_{\text{ou}} = \mathcal{P}_k$ has the desired properties.

Finally, let $\mathcal{P}^\dagger$ be the path obtained from $\mathcal{P}_{\text{ou}}$ by unwinding the transformations that generated each of its edges. Now, by its construction, $\mathcal{P}^\dagger$ is a semi-reducible path from X to Y in $\mathcal{G}$ such that $|\mathcal{P}^\dagger| = |\mathcal{P}_{\text{ou}}| \leq |\mathcal{P}'| \leq |\mathcal{P}|$. Furthermore, every edge in $\mathcal{P}_{\text{ou}}$ belongs to $\mathcal{F}_k$. It only remains to show that every OU-edge appearing anywhere during the transformation of $\mathcal{P}^\dagger$ into $\mathcal{P}_{\text{ou}}$ must belong to some $\mathcal{F}_i$. That follows immediately from how the $\mathcal{F}_i$ sets are defined (cf. Defn. 36): for each $i \geq 0$, $\mathcal{F}_{i+1} = \texttt{shortest}(\mathcal{F}_i \cup \texttt{children}(\mathcal{E}_L, \mathcal{F}_i))$. Thus, the only way that an edge E can belong to $\mathcal{F}_{i+1}$ is if it already belongs to $\mathcal{F}_i$ or it was generated by parent edges that were either lower-case edges or OU-edges in $\mathcal{F}_i$. □

The next theorem ensures that if an STNU has no SRN loops, then the process of generating strictly shorter edges cannot continue indefinitely. In particular, the generation of strictly shorter edges converges to shortest edges after a finite number of rounds, thereby ensuring that the notion of shortest semi-reducible paths is well defined. The proof technique draws from Morris and Muscettola [19].

Theorem 7 (No SRN loop ⇒ edge generation bounded [19]) *Let $\mathcal{S}$ be an STNU with N time-points and K contingent links; and let $\mathcal{G}$ be the graph for $\mathcal{S}$. If $\mathcal{G}$ has no SRN loops, then after at most $N^2 + NK + K$ rounds, the application of the edge-generation rules from Table 1 to edges in $\mathcal{G}$ cannot generate any* strictly shorter *edges. In other words, $\mathcal{F}_{i+1} = \mathcal{F}_i$, for each $i \geq N^2 + NK + K$.*

Proof Suppose, contrary to the statement of the theorem, that for some $r \geq N^2 + NK + K$, $\mathcal{F}_{r+1}$ contains an edge, E_{r+1}, that does not belong to $\mathcal{F}_r$. Since $E_{r+1} \notin \mathcal{F}_r$, it follows that (at least) one of the edges used to generate E_{r+1} must belong to $\mathcal{F}_r$, but not $\mathcal{F}_{r-1}$. Call that edge E_r. Similarly, (at least) one of the edges used to generate E_r must belong to $\mathcal{F}_{r-1}$, but not $\mathcal{F}_{r-2}$. Call that edge E_{r-1}. Continue in this way until, after r steps, an edge E_1 is found that belongs to $\mathcal{F}_1$, but not $\mathcal{F}_0$. Now, consider the sequence of $r+1$ edges, $E_{r+1}, E_r, E_{r-1}, \ldots, E_1$. Since $r + 1 > N^2 + NK + K$, it follows that for some $i > j$, the edges E_i and E_j have the same signature. Furthermore, since E_i was generated after E_j, and only shortest edges are kept in the sets $\mathcal{F}_i$, it follows that $|E_i| < |E_j|$. Since E_i and E_j have the same signature, they must have the same end-points—say, X and Y. As a

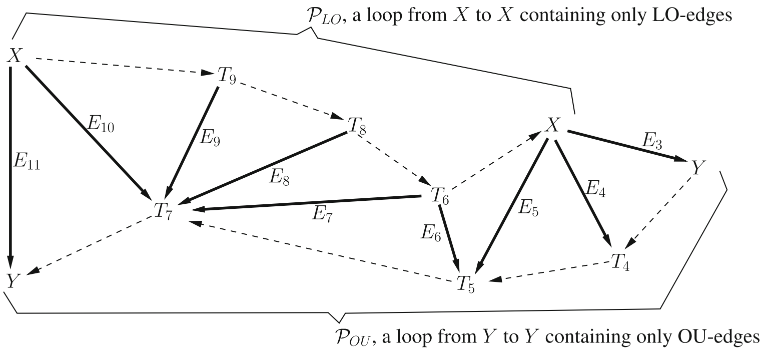

$\mathcal{P}_{LO}$, a loop from X to X containing only LO-edges
X
T_9
E_{10}
E_9
T_8
X
E_3
Y
E_{11}
E_8
T_6
E_4
E_5
T_7
E_7
E_6
T_4
Y
T_5
$\mathcal{P}_{OU}$, a loop from Y to Y containing only OU-edges

Fig. 7 A chain of reductions resulting in an LO-loop from X to X, and an OU-loop from Y to Y

result, the chain of transformations used to generate E_i from E_j must include two loops: $\mathcal{P}_{LO}$, a loop from X to X consisting of LO-edges; and $\mathcal{P}_{OU}$, a loop from Y to Y consisting of OU-edges, as illustrated in Fig. 7, where $E_i = E_{11}$ and $E_j = E_3$. The reason is that in each of the binary edge-generation rules, the lefthand edge must be an LO-edge and the righthand edge must be an OU-edge. (The *Label Removal* rule converts upper-case edges into ordinary edges, which does not disturb this conclusion.) Since $|E_i| < |E_j|$ and the rules preserve path-length, it follows that one of these loops must have negative length. Now, if $\mathcal{P}_{OU}$ has negative length, then unwinding the transformations that generated its edges results in a path $\mathcal{P}_1$ in $\mathcal{G}$ that, because it reduces to $\mathcal{P}_{OU}$ is, by definition, semi-reducible. On the other hand, if $\mathcal{P}_{LO}$ has negative length, then, by Lemma 11, it must be semi-reducible. Unwinding the transformations that generated the edges in $\mathcal{P}_{LO}$ therefore results in a path $\mathcal{P}_2$ in $\mathcal{G}$ that also must be semi-reducible. Thus, either case results in an SRN loop in $\mathcal{G}$, contradicting the premise of the theorem. □

Definition 37 ($\mathcal{F}^*$ *and* $\texttt{strip}(\mathcal{F}^*)$) Let $\mathcal{G}$ be the graph for an STNU with N time-points and K contingent links. If $\mathcal{G}$ does not have any SRN loops, then $\mathcal{F}^* = \mathcal{F}_{N^2+NK+K}$ denotes the set of OU-edges obtained from at most $N^2 + NK + K$ rounds of edge generation in which, for each signature, non-shortest edges are discarded at the end of each round; and $\texttt{strip}(\mathcal{F}^*)$ denotes the set of *ordinary* edges obtained by stripping the alphabetic labels from any upper-case edges in $\mathcal{F}^*$.

Lemma 12 *Let $\mathcal{G}$ be the graph for an STNU. If $\mathcal{G}$ has no SRN loops, then the notion of shortest semi-reducible paths in $\mathcal{G}$ is well defined.*

Proof Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU whose corresponding graph, $\mathcal{G} = \langle \mathcal{T}, \mathcal{E}^+ \rangle$, has no SRN loops; and let $\mathcal{F}^*$ be the set of all OU-edges obtained by applying the edge-generation rules from Table 1 to edges in $\mathcal{G}$. By Theorem 7, $\mathcal{F}^*$ is obtained in at most $N^2 + NK + K$ rounds; and since $\mathcal{F}^*$ contains only the shortest OU-edge for each signature, $|\mathcal{F}^*| \leq N^2 + NK$. Furthermore, note that each path $\mathcal{P}^*$ consisting of edges in $\mathcal{F}^*$ has a corresponding *semi-reducible* path $\mathcal{P}_{\text{sr}}$ in $\mathcal{G}$ that is obtained by unwinding the transformations used to generate the edges in $\mathcal{P}^*$ and, hence, $|\mathcal{P}_{\text{sr}}| = |\mathcal{P}^*|$.

Next, let X and Y be *distinct* time-points in $\mathcal{G}$. Since $|\mathcal{F}^*|$ is finite, there are only finitely many *loopless* paths from X to Y whose edges belong to $\mathcal{F}^*$. Therefore, the value

$$\delta^*(X, Y) = \min\{|\mathcal{P}^*_{xy}| : \mathcal{P}^*_{xy} \text{ is a loopless path from } X \text{ to } Y \text{whose edges are in } \mathcal{F}^*\}$$

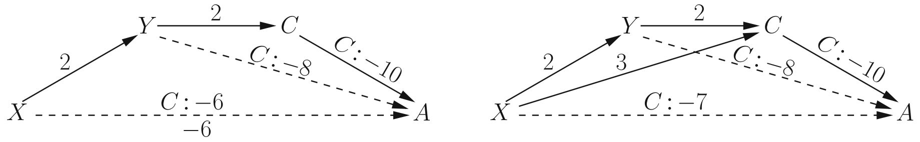


Fig. 8 Examples illustrating some subtleties of discarding non-shortest edges

is guaranteed to satisfy $\delta^*(X, Y) > -\infty$. (If there are no paths from X to Y whose edges are in $\mathcal{F}^*$, then $\delta^*(X, Y) = \infty$.)

Next, let $\mathcal{P}$ be any semi-reducible path in $\mathcal{G}$ from X to Y. By Theorem 6, there is a semi-reducible path $\mathcal{P}^\dagger$ from X to Y in $\mathcal{G}$ such that $|\mathcal{P}^\dagger| \leq |\mathcal{P}|$ and $\mathcal{P}^\dagger$ can be transformed into an OU-path, $\mathcal{P}_{\text{ou}}$, such that (1) the edges in $\mathcal{P}_{\text{ou}}$ all belong to $\mathcal{F}^*$; and (2) each OU-edge that appears during the transformation of $\mathcal{P}^\dagger$ into $\mathcal{P}_{\text{ou}}$ belongs to some $\mathcal{F}_i$.[20] Now, if $\mathcal{P}_{\text{ou}}$ happens to contain any subsidiary loops, let $\mathcal{P}^\star_{\text{ou}}$ be the path obtained by extracting all such loops from $\mathcal{P}_{\text{ou}}$. Now, any such loop, $\mathcal{P}_\circ$, because its edges are all in $\mathcal{F}^*$, has a corresponding semi-reducible loop in $\mathcal{G}$ with the same length; and since $\mathcal{G}$ has no SRN loops, it follows that $|\mathcal{P}_\circ| \geq 0$. Therefore, extracting all such loops from $\mathcal{P}_{\text{ou}}$ cannot increase its length; hence, $|\mathcal{P}^\star_{\text{ou}}| \leq |\mathcal{P}_{\text{ou}}|$. But then the *loopless* path $\mathcal{P}^\star_{\text{ou}}$, with edges in $\mathcal{F}^*$, satisfies:

$$-\infty < \delta^*(X, Y) \leq |\mathcal{P}^\star_{\text{ou}}| \leq |\mathcal{P}_{\text{ou}}| = |\mathcal{P}^\dagger| \leq |\mathcal{P}|.$$

Therefore, $\delta^*(X, Y)$ is a lower bound for the lengths of semi-reducible paths from X to Y in $\mathcal{G}$. Furthermore, since there are only finitely many loopless paths from X to Y having edges in $\mathcal{F}^*$, it follows that one of them—call it $\mathcal{P}^{\text{xy}}_{\text{ou}}$—has a path length equal to $\delta^*(X, Y)$. Since its edges are all in $\mathcal{F}^*$, $\mathcal{P}^{\text{xy}}_{\text{ou}}$ must have a corresponding semi-reducible path $\mathcal{P}^{\text{xy}}_{\text{sr}}$ from X to Y in $\mathcal{G}$ such that $|\mathcal{P}^{\text{xy}}_{\text{sr}}| = |\mathcal{P}^{\text{xy}}_{\text{ou}}|$, it follows that $\mathcal{P}^{\text{xy}}_{\text{sr}}$ is a shortest semi-reducible path from X to Y in $\mathcal{G}$, and that $|\mathcal{P}^{\text{xy}}_{\text{sr}}| = \delta^*(X, Y)$.

Finally, consider the case where $X \equiv Y$. If there are no semi-reducible loops containing X, then $\delta^*(X, X) = \infty$. Otherwise, the same kind of argument as that used above can be used for any semi-reducible loop $\mathcal{P}$ containing X, except that only *proper* sub-loops should be extracted from the corresponding loop $\mathcal{P}_{\text{ou}}$ to generate a loop $\mathcal{P}^\star_{\text{ou}}$ that does not contain any proper sub-loops. Similarly, the definition of $\delta^*(X, X)$ should be based on loops with edges in $\mathcal{F}^*$ that contain X, but do not have any proper sub-loops. Finally, since $\mathcal{G}$ has no SRN loops, the length of any semi-reducible loop containing X is bounded below by 0; hence, $0 \leq \delta^*(X, X) \leq |\mathcal{P}^\star_{\text{ou}}| \leq |\mathcal{P}_{\text{ou}}| \leq |\mathcal{P}|$. Thus, whether $X \equiv Y$ or not, $\delta^*(X, Y)$ equals the length of the shortest semi-reducible path from X to Y in $\mathcal{G}$. □

Figure 8 illustrates some of the subtleties associated with discarding non-shortest edges. The left side of the figure shows an STNU graph that leads to the generation of the upper-case edge, $X \xrightarrow{C:-6} A$. Assuming that the relevant contingent link is $(A, 6, 10, C)$, the *Label Removal* rule then yields the ordinary edge, $X \xrightarrow{-6} A$. The right side of the figure shows the same graph, except that it now includes an edge of length 3 from X to C. That leads to the generation of the upper-case edge, $X \xrightarrow{C:-7} A$. However, the *Label Removal* rule does not apply to this edge, since the wait time 7 is greater than the lower bound on the contingent link. If the non-shortest upper-case edge, $X \xrightarrow{C:-6} A$, is discarded, then the ordinary edge, $X \xrightarrow{-6} A$, will never be generated. Therefore, the shortest ordinary edge from X to A is

[20] In view of Theorem 7, the set $\mathcal{F}_k$ from Theorem 6 can be taken to be $\mathcal{F}^*$ with no loss of generality.

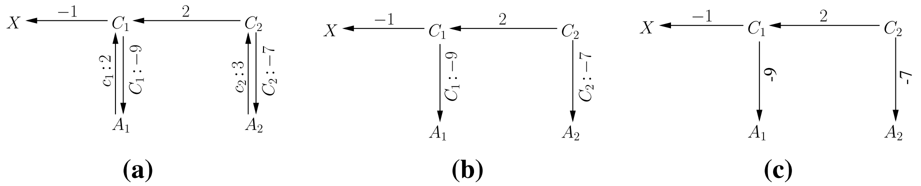


Fig. 9 Graphs associated with the sample STNU $\mathcal{S}^\dagger$ (cf. Fig. 2). **a** The graph $\mathcal{G}$ for the STNU $\mathcal{S}^\dagger$. **b** The corresponding OU-graph $\mathcal{G}_{\text{ou}}$. **c** The corresponding *AllMax* graph $\mathcal{G}_x$

missed. However, the shortest *semi-reducible* path from X to A, as represented by the upper-case edge of length -7, is not missed. And that is what counts with regard to dynamic controllability.

Given that the notion of shortest paths is well defined for STNUs having no SRN loops, the following definitions culminate in a definition of the *All-Pairs-Shortest-Semi-Reducible-Paths* (APSSRP) matrix, $\mathcal{D}^*$, which plays a central role in all that follows.

Definition 38 (*OU-graph*, $\mathcal{G}_{\text{ou}}$ [13]; *AllMax graph*, $\mathcal{G}_x$ [19]) Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU; $\mathcal{G}$ its graph; and $\mathcal{E}_{\text{ou}}$ the set of all OU-edges from $\mathcal{G}$. Then:

$$\begin{aligned}\mathcal{G}_{\text{ou}} &= \text{the graph } \langle \mathcal{T}, \mathcal{E}_{\text{ou}}\rangle\text{—called the } \textit{OU-graph} \text{ for } \mathcal{S}.\\ \mathcal{G}_x &= \text{the } \textit{STN} \text{ graph, } \langle \mathcal{T}, \mathtt{strip}(\mathcal{E}_{\text{ou}})\rangle\text{, obtained by stripping the alphabetic labels}\\ &\quad\ \text{from any upper-case edges in } \mathcal{E}_{\text{ou}}\text{—called the } \textit{AllMax} \text{ graph for } \mathcal{S}.\end{aligned}$$

Figure 9 shows a sample STNU graph $\mathcal{G}$ with the corresponding OU-graph and *AllMax* graph. Note that the paths in $\mathcal{G}_{\text{ou}}$ are precisely the OU-paths from $\mathcal{G}$. Note further that, unlike the OU-graph, the *AllMax* graph is an STN graph: it contains only ordinary edges. The *AllMax* graph acquired its name from the fact that removing the labels from upper-case edges effectively forces the corresponding contingent links to take on their maximum durations.[21]

The graphs $\mathcal{G}^*$, $\mathcal{G}^*_{\text{ou}}$ and $\mathcal{G}^*_x$, defined below, are obtained by inserting the edges from $\mathcal{F}^*$ into the graphs $\mathcal{G}$, $\mathcal{G}_{\text{ou}}$ and $\mathcal{G}_x$, respectively, except that in the case of $\mathcal{G}^*_x$, any upper-case labels are first stripped from the edges in $\mathcal{F}^*$. The graphs, $\mathcal{G}^*_{\text{ou}}$ and $\mathcal{G}^*_x$ play particularly important roles during the execution of an STNU.

Definition 39 ($\mathcal{G}^*, \mathcal{G}^*_{\text{ou}}, \mathcal{G}^*_x, \mathcal{D}^*_x$) Let $\mathcal{G} = \langle \mathcal{T}, \mathcal{E}^+\rangle$ be an STNU graph that has no SRN loops. Then:

$$\begin{aligned}\mathcal{G}^* &= \langle \mathcal{T}, \mathcal{E}^+ \cup \mathcal{F}^*\rangle;\\ \mathcal{G}^*_{\text{ou}} &= \langle \mathcal{T}, \mathcal{F}^*\rangle \text{ is the OU-graph for } \mathcal{G}^*;\\ \mathcal{G}^*_x &= \langle \mathcal{T}, \mathtt{strip}(\mathcal{F}^*)\rangle \text{ is the } \textit{AllMax} \text{ graph for } \mathcal{G}^*;\ \text{and}\\ \mathcal{D}^*_x &= \text{the distance matrix for } \mathcal{G}^*_x.\end{aligned}$$

Figure 10 shows the graphs, $\mathcal{G}^*$, $\mathcal{G}^*_{\text{ou}}$ and $\mathcal{G}^*_x$, that correspond to the graphs in Fig. 9 after inserting the edges from the set $\mathcal{F}^*$. Note that $\mathcal{G}^*_x$ is an STN graph.

[21] Recall that, for any contingent link, (A, x, y, C), the execution semantics for STNUs ensures that the duration of each contingent link must stay within its upper bound, y. This, together with the ordinary constraint, $C - A \geq y$, represented by the edge, $C \xrightarrow{-y} A$ in the *AllMax* graph, effectively forces $C - A = y$. For an alternative view, note that if the additional ordinary edges discussed in Footnote 12 were included in the STNU graph, then for each contingent link, (A, x, y, C), the *AllMax* graph would have *two* ordinary edges together representing the constraint, $C - A = y$.

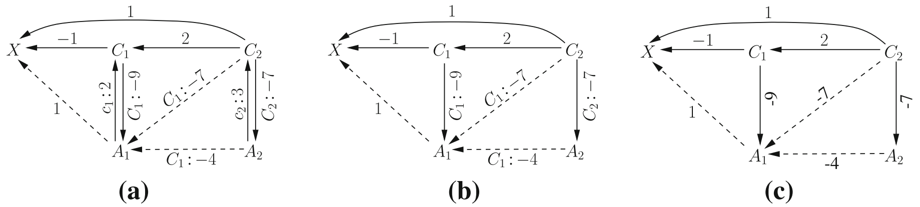


Fig. 10 The graphs, $\mathcal{G}^*, \mathcal{G}^*_{\text{ou}}$ and $\mathcal{G}^*_x$, derived from the corresponding graphs from Fig. 9. **a** $\mathcal{G}^* = \langle \mathcal{T}, \mathcal{E}^+ \cup \mathcal{F}^* \rangle$. **b** $\mathcal{G}^*_{\text{ou}} = \langle \mathcal{T}, \mathcal{F}^* \rangle$. **c** $\mathcal{G}^*_x = \langle \mathcal{T}, \texttt{strip}(\mathcal{F}^*) \rangle$

$\mathcal{D}^*$	A_1	C_1	A_2	C_2	X
A_1	0				1
C_1	−9	0			−8
A_2	−4		0		−3
C_2	−11	2	−7	0	−10
X					0

Fig. 11 The APSSRP matrix for the sample STNU from Figs. 9 and 10 (blank entries are ∞)

Corollary 3 *If $\mathcal{G}$ is an STNU graph that has no SRN loops, then shortest paths in the graph, $\mathcal{G}^*_x = \langle \mathcal{T}, \texttt{strip}(\mathcal{F}^*) \rangle$, correspond to shortest semi-reducible paths in $\mathcal{G}$.*

Proof Let X and Y be any time-points in $\mathcal{G}$. In the Proof of Lemma 12, the value $\delta^*(X, Y)$ is shown to equal the length of the shortest path from X to Y whose edges are in $\mathcal{F}^*$, as well as the length of the shortest semi-reducible path from X to Y in $\mathcal{G}$. Since stripping the alphabetic labels from upper-case edges does not affect path lengths, $\delta^*(X, Y)$ also equals the lengths of the shortest path from X to Y in the STN graph, $\mathcal{G}^*_x = \langle \mathcal{T}, \texttt{strip}(\mathcal{F}^*) \rangle$. □

In view of Corollary 3, if $\mathcal{G}$ is an STNU graph that has no SRN loops, then for any time-points X and Y, the value $\mathcal{D}^*_x(X, Y)$ equals the length of the shortest semi-reducible path from X to Y in $\mathcal{G}$, thus motivating the following definition.

Definition 40 (*APSSRP matrix*) Let $\mathcal{S}$ be an STNU whose graph $\mathcal{G} = \langle \mathcal{T}, \mathcal{E}^+ \rangle$ has no SRN loops. Then the *AllMax* matrix $\mathcal{D}^*_x$, defined above, is also called the *all-pairs shortest-semi-reducible-paths* (APSSRP) matrix for $\mathcal{S}$. It is frequently notated as $\mathcal{D}^*$.

The APSSRP matrix for the sample STNU from Figs. 9 and 10 is shown in Fig. 11, where entries that are infinite are left blank to improve readability.[22] For example, there are no edges emanating from X, hence all off-diagonal entries, $\mathcal{D}^*(X, Y)$, are infinite. For another example, note that $\mathcal{D}^*(C_2, X) = -10$, which is the length of the path in $\mathcal{G}^*_x$ that goes from C_2 to A_2 to A_1 to X. Although this path is shorter than the ordinary edge in $\mathcal{G}^*_{\text{ou}}$ from C_2 to X, that ordinary edge must be retained for use by the execution algorithm, as will be seen.

Computing the APSSRP matrix Morris and Muscettola's $O(N^5)$-time DC-checking algorithm computes the APSSRP matrix $\mathcal{D}^*$ using at most $N^2 + NK + K$ rounds of edge generation, at a cost of $O(N^3)$ time per round, to incrementally generate the sets, $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}^*$, followed by an $O(N^3)$-time APSP computation to generate the distance matrix for $\mathcal{G}^*_x$ [19]. Since the lower-case edges in any semi-reducible path must eventually

[22] The distance matrix shown in Fig. 11 presumes that there are self-loops of length 0 at each time-point, which is common practice in the literature. For dynamically controllable networks, this leads to the distance matrix having zeroes down its main diagonal.

get “reduced away”, Morris’ faster, $O(N^4)$-time DC-checking algorithm searches for extension sub-paths that can be used to reduce away lower-case edges [16]. Although based on a much different approach, Morris’ algorithm has been shown to compute the same matrix, $\mathcal{D}^*$ [11]. Finally, Shah et al. [27] and Nilsson et al. [23] have argued, in effect, that it suffices to restrict the *No Case* and *Upper Case* to cases where $w \leq 0 \leq v$. The reader can easily verify that restricting the rules in this way does not hinder the canonical elimination of lower-case edges first described by Morris [16]. Adapting the proofs of Theorems 6 and 7 to accommodate this (slightly) more restricted form of edge generation is left as an exercise.

4.3 An STNU with no SRN loops must be dynamically controllable

This section uses the matrix, $\mathcal{D}_x^*$, to define an RTED-based execution strategy that is then shown to be reliable (i.e., it satisfies all constraints in the network no matter how the contingent durations turn out). The RTED-based strategy presented in this section draws from prior work [11,12], but is streamlined to facilitate the proof of its reliability.

The main insight behind the execution strategy is that the *AllMax* graph $\mathcal{G}_x^*$, which does not distinguish ordinary and upper-case edges, contains all of the information needed to generate the next execution decision [11]. The reason for this is that, prior to the execution of a contingent time-point C, any upper-case edge, $Y \xrightarrow{C:-w} A$, labeled by C, is indistinguishable from the corresponding ordinary edge, $Y \xrightarrow{-w} A$, in terms of its effect on as-yet-unexecuted time-points. The catch, however, is that after C executes, the conditional “wait” constraints represented by upper-case edges labeled by C are no longer applicable and, thus, must be *removed* from the corresponding OU-graph $\mathcal{G}_{\mathrm{ou}}^*$, thereby leading to changes in the *AllMax* graph $\mathcal{G}_x^*$ and its corresponding distance matrix, $\mathcal{D}_x^*$.[23]

The execution algorithm is iterative. Each iteration involves the following steps. First, the matrix, $\mathcal{D}_x^*$, is used to generate an execution decision. Second, the execution outcome is observed: one or more time-points executing at some time t. Third, the OU-graph and the *AllMax* graph are updated. Finally, the matrix, $\mathcal{D}_x^*$, is updated in preparation for the next iteration.

Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU, and $\mathcal{G} = \langle \mathcal{T}, \mathcal{E}^+ \rangle$ its graph. This section presumes that there are no SRN loops in $\mathcal{G}$, and hence no negative loops in the corresponding *AllMax* graph, $\mathcal{G}_x^* = \langle \mathcal{T}, \texttt{strip}(\mathcal{F}^*) \rangle$. Now, initially, $\mathcal{D}_x^*$ is both the APSSRP matrix for $\mathcal{G}$, and the distance matrix for the *AllMax* graph $\mathcal{G}_x^*$. The main goal during the execution of the network is to ensure that no negative loops are ever introduced into the *AllMax* graph. Since that STN graph contains all of the constraints in $\mathcal{C}$, it suffices to preserve its consistency during execution. As a consequence, the edge-generation rules from Table 1 are not used during execution. Instead, edges are merely inserted into or removed from the graphs, $\mathcal{G}_{\mathrm{ou}}^*$ and $\mathcal{G}_x^*$, and the distance matrix $\mathcal{D}_x^*$ is appropriately updated. It is not necessary to address whether the updated matrix $\mathcal{D}_x^*$ remains the APSSRP matrix for the correspondingly updated STNU.

The execution phase By Lemma 1, there is some time-point, $X_0 \in \mathcal{T}$, such that constraining X_0 to equal 0, and constraining every other time-point in the *AllMax* STN to occur at or after X_0, will not affect the consistency of that STN. That time-point shall henceforth be called Z. Distance matrix entries involving Z play an important role during execution.

Let $\mathcal{U}_x$ denote the as-yet-unexecuted executable time-points. Recall that for any $X \in \mathcal{U}_x$, its execution window is: $[-\mathcal{D}_x^*(X, Z), \mathcal{D}_x^*(Z, X)]$. As a result, the earliest time that *any* time-

[23] As a consequence, during execution, the *AllMax* graph only forces all *as-yet-unexecuted* contingent links to take on their maximum durations.

point in $\mathcal{U}_x$ can be executed is: $t_L = \min\{-\mathcal{D}^*_x(X, Z) \mid X \in \mathcal{U}_x\}$. And the latest time before *some* time-point in $\mathcal{U}_x$ *must* be executed is: $t_U = \min\{\mathcal{D}^*_x(Z, X) \mid X \in \mathcal{U}_x\}$.

During execution, the graphs, $\mathcal{G}^*_{\text{ou}}$ and $\mathcal{G}^*_x$, are both updated—by inserting or removing edges—in response to the most recent execution events. For example, if a (contingent or executable) time-point X executes at time t, then the edges, $Z \xrightarrow{t} X$ and $Z \xleftarrow{-t} X$, are inserted into the graphs, representing the execution constraints, $X \leq t$ and $X \geq t$ (i.e., $X = t$). Second, since upper-case edges correspond to conditional wait constraints ("As long as C remains unexecuted, V must wait at least w units after A"), whenever a contingent time-point C is executed, all upper-case edges labeled by C are *removed* from the OU-graph, which typically leads to changes in the *AllMax* graph, $\mathcal{G}^*_x$ [11]. In turn, the distance matrix $\mathcal{D}^*_x$ for $\mathcal{G}^*_x$ must be updated. For example, if all upper-case edges labeled by C_1 were removed from the OU-graph $\mathcal{G}^*_{\text{ou}}$ in Fig. 10b, then the corresponding *ordinary* edges would be removed from the *AllMax* graph $\mathcal{G}^*_x$ in Fig. 10c. Those changes would then make the edge from C_2 to X of length 1 the new (longer) shortest path from C_2 to X, requiring updating $\mathcal{D}^*(C_2, X)$ from -10 to 1.

All edges that are inserted into these graphs during execution invariably involve Z, whether as a source or destination time-point; but edges that are removed may not involve Z.

The execution strategy Definition 41, below, defines an execution strategy, $\hat{R}$, that is used to demonstrate the existence of a reliable strategy for an STNU $\mathcal{S}$ whose graph has no SRN loops. The decisions generated by $\hat{R}$ depend solely on the information in the *updated* matrix $\mathcal{D}^*_x$ which, in turn, is derived solely from the initial STNU graph together with execution constraints of the form, $X = t$, for the already executed time-points. As a result, the decisions generated by $\hat{R}$ depend only on information present in the current partial schedule. In this way, $\hat{R}$ is effectively a mapping from respectful partial schedules to RTEDs and, thus, is an RTED-based execution strategy.

As usual, the initial partial schedule is $\xi_0 = \emptyset$. The first decision, $\hat{R}(\xi_0)$, is defined to be $(0, \chi_0)$, where χ_0 is the set of time-points that are constrained to co-occur with Z in the *AllMax* STN. (Typically, χ_0 will only contain Z.) Since no contingent time-points have been activated yet, the outcome of this initial decision is fully determined: the time-points in χ_0 are executed at 0. For all other partial schedules ξ, $\hat{R}(\xi)$ is defined as follows.

Definition 41 (*RTED-based strategy*, $\hat{R}$) Let ξ be a *non-empty* respectful partial schedule; let $\mathcal{U}_x = \mathcal{T}_x - \text{Vars}(\xi)$ be the set of executable time-points that are not yet executed in ξ; and let $\mathcal{D}^*_x$ be the corresponding *AllMax* matrix. If $\mathcal{U}_x$ is empty, then $\hat{R}(\xi) = \texttt{wait}$. Otherwise, $\hat{R}(\xi) = (t, \chi)$, where t and χ are determined as follows. First, let t_L be the minimum lower bound for as-yet-unexecuted executable time-points; and let χ be the set of time-points whose lower bound is t_L:

$$t_L = \min\{-\mathcal{D}^*_x(X, Z) \mid X \in \mathcal{U}_x\}; \text{ and}$$
$$\chi = \{X \in \mathcal{U}_x \mid -\mathcal{D}^*_x(X, Z) = t_L\}.$$

Next, if $t_L > \texttt{now}_\xi$, then let $t = t_L$. Otherwise, $t_L \leq \texttt{now}_\xi$, which is not a legal time for an execution decision based on ξ; therefore, let $t = \frac{\texttt{now}_\xi + t_U}{2}$, where t_U is the minimum *upper bound* for all as-yet-unexecuted executable time-points:

$$t_U = \min\{\mathcal{D}^*_x(Z, X) \mid X \in \mathcal{U}_x\}.$$

Under the assumption that $t_U > \texttt{now}_\xi$, t is thus guaranteed to be greater than $\texttt{now}_\xi$.[24]

Theorem 8, below, confirms the most important property of STNUs.

[24] The Proof of Theorem 8 will show that $t_U > \texttt{now}_\xi$ invariably holds when following the strategy $\hat{R}$.

Theorem 8 (No SRN loops $\Rightarrow$ DC) *Let $\mathcal{S}$ be any STNU. If the graph for $\mathcal{S}$ has no SRN loops, then $\mathcal{S}$ must be dynamically controllable.*

Proof Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be any STNU whose graph $\mathcal{G} = \langle \mathcal{T}, \mathcal{E}^+ \rangle$ has no SRN loops. It will be shown that the RTED-based execution strategy $\hat{R}$ from Definition 41 is a reliable strategy for $\mathcal{S}$ (i.e., for every situation ω, the complete schedule $\mathcal{O}^*(\hat{R}, \omega)$ that results from following $\hat{R}$ in ω satisfies all of the constraints in $\mathcal{C}$) and, therefore, that $\mathcal{S}$ is dynamically controllable.

Let $\mathcal{G}^*_{\text{ou}} = \langle \mathcal{T}, \mathcal{F}^* \rangle$ be the OU-graph for $\mathcal{G}$ (cf. Definition 39). During execution, $\mathcal{G}^*_{\text{ou}}$ will be updated (i.e., destructively modified) as follows: (1) whenever any (contingent or executable) time-point X executes, edges corresponding to constraints of the form, $X = t$, will be inserted into $\mathcal{G}^*_{\text{ou}}$; and (2) whenever any contingent time-point C executes, the upper-case edges labeled by C will be removed from $\mathcal{G}^*_{\text{ou}}$. At each point during the execution of the network, the *AllMax* graph $\mathcal{G}^*_x$ (cf. Definition 39) is the graph obtained by stripping any alphabetic labels from whatever edges are *currently* in $\mathcal{G}^*_{\text{ou}}$; and the *AllMax* matrix $\mathcal{D}^*_x$ is the corresponding distance matrix.[25]

Let ω be any situation; and let $\xi_0, \xi_1, \xi_2, \ldots, \xi_p = \mathcal{O}^*(\hat{R}, \omega)$ be the unique sequence of schedules that results from following $\hat{R}$ in ω (cf. Lemma 2). For each $i \leq p$:

- let $\mathcal{U}^i_x$ be the set of executable time-points that are not yet executed in ξ_i;
- let $(\mathcal{G}^*_{\text{ou}})_i$ be the OU-graph updated to reflect all of the execution events in ξ_i;
- let $(\mathcal{G}^*_x)_i$ be the *AllMax* graph derived from $(\mathcal{G}^*_{\text{ou}})_i$;
- let $(\mathcal{D}^*_x)_i$ be the *AllMax* matrix derived from $(\mathcal{G}^*_x)_i$;
- let $t^i_L = \min\{-(\mathcal{D}^*_x)_i(X, Z) \mid X \in \mathcal{U}^i_x\}$ be the minimum lower bound among the executable time-points that are not yet executed in ξ_i; and
- let $t^i_U = \min\{(\mathcal{D}^*_x)_i(Z, X) \mid X \in \mathcal{U}^i_x\}$ be the minimum upper bound among the executable time-points that are not yet executed in ξ_i.

Next, for each $i \leq p$, consider the following propositions.

- $P(i)$: the graph $(\mathcal{G}^*_x)_i$ has no negative loops; and
- $Q(i)$: $\texttt{now}_{\xi_i} < t^i_U$.

The main goal of the rest of the proof is to show by induction that $P(i)$ holds for each $i \leq p$ (i.e., that the *AllMax* STN remains consistent throughout the execution of the network). Since the *AllMax* STN contains all of the constraints in $\mathcal{C}$, as well as all of the execution constraints, $P(p)$ holding implies that the unique outcome obtained by following the strategy $\hat{R}$ in the situation ω satisfies all constraints in $\mathcal{C}$, as desired. The proof is facilitated by showing in parallel that $Q(i)$ also holds for each $i \leq p$ (i.e., that the upper bounds for all as-yet-unexecuted time-points are in the future).

Base cases $P(0)$ and $P(1)$. Since $\mathcal{G}$ has no SRN loops, the initial *AllMax* graph, $(\mathcal{G}^*_x)_0$, cannot have any negative loops. Therefore, $P(0)$ holds. In addition, since the set χ_0 in the initial decision, $\hat{R}(\xi_0) = (0, \chi_0)$, contains precisely those executable time-points X that, in the *AllMax* graph, are constrained to co-occur with Z, the corresponding execution constraints, $X = 0$, are redundant in $(\mathcal{G}^*_x)_0$ (cf. Lemma 1). Therefore, $(\mathcal{G}^*_x)_1$ and $(\mathcal{G}^*_x)_0$ are equivalent. Hence, $P(1)$ also holds.

[25] For the purposes of the proof, it is convenient to ignore issues of computational efficiency. Thus, the graph $\mathcal{G}^*_x$, and the distance matrix $\mathcal{D}^*_x$ are thought of as being effectively re-computed from scratch after each update to the OU-graph $\mathcal{G}^*_{\text{ou}}$. The FAST-EX algorithm presented in Sect. 5 uses incremental techniques to efficiently update $\mathcal{G}^*_x$ and $\mathcal{D}^*_x$ after each execution event.

Base cases $Q(0)$ *and* $Q(1)$. Since $(\mathcal{G}_x^*)_0$ is consistent, it follows that for each $X \in \mathcal{U}_x^0$, $(\mathcal{D}_x^*)_0(Z, X) > -\infty = \texttt{now}_{\xi_0}$. Thus, $t_U^0 > \texttt{now}_{\xi_0}$ (i.e., $Q(0)$ holds). In addition, all executable time-points that remain unexecuted at time 0 must have upper bounds that are strictly greater than 0, since all those executable time-points that were constrained, in the *AllMax* graph, to co-occur with Z were defined, by $\hat{R}$, to be in the set χ_0. As a result, $\texttt{now}_{\xi_1} = 0 < t_U^1$ (i.e., $Q(1)$ holds).

Recursive case Suppose the propositions $P(i)$ and $Q(i)$ hold for some i, where $1 \leq i < p$. Let $\Delta_i = \hat{R}(\xi_i)$ be the execution decision determined by $\hat{R}$ for the partial schedule ξ_i, as prescribed by Definition 41. In view of Definitions 17 and 18, there are three possibilities for the outcome ξ_{i+1} that results from the decision $\hat{R}(\xi_i)$:[26]

(1) Some set, χ_c, of contingent time-points execute together at some time ρ;
(2) Some set, χ, of executable time-points execute together at some time t; or
(3) Some set, $\chi_c \cup \chi$, of contingent *and* executable time-points execute together at some time $\rho = t$.

The following analysis shows that the propositions $P(i + 1)$ and $Q(i + 1)$ must hold no matter which combination of time-points happens to execute. The analysis is done incrementally. First, the updates associated with any contingent executions that might have occurred are applied to the graph $(\mathcal{G}_x^*)_i$. Afterward, the updates associated with the execution of any executable time-points are applied.

For clarity, $\mathcal{G}_{\text{ou}}^*$ will notate the OU-graph at each stage of the analysis. Thus, $\mathcal{G}_{\text{ou}}^*$ will start out as $(\mathcal{G}_{\text{ou}}^*)_i$ and, after all the updates are done, will end up as $(\mathcal{G}_{\text{ou}}^*)_{i+1}$. Similar remarks apply to $\mathcal{G}_x^*$ and $\mathcal{D}_x^*$, which will start out as $(\mathcal{G}_x^*)_i$ and $(\mathcal{D}_x^*)_i$, respectively, and end up as $(\mathcal{G}_x^*)_{i+1}$ and $(\mathcal{D}_x^*)_{i+1}$.

Case 1: Contingent executions Let χ_c be the set of contingent time-points that, according to the outcome ξ_{i+1}, execute at the time $\rho = \texttt{now}_{\xi_{i+1}} > \texttt{now}_{\xi_i}$. According to the execution semantics, a contingent time-point C can only execute at time ρ if the corresponding contingent link, (A, x, y, C), is *active* in ξ_i, and $\texttt{now}_{\xi_i} < a + y$, where $a = \xi_i(A)$ is the execution time for A in ξ_i. The following updates will be done to the graph $\mathcal{G}_{\text{ou}}^*$, in the order specified.

(i) for each $C \in \chi_c$, remove all *generated* upper-case edges labeled by C;
(ii) for each $C \in \chi_c$, remove the corresponding *original* upper-case edge, $C \xrightarrow{C:-y} A$, and then insert the ordinary edge, $C \xrightarrow{-\rho} Z$ (i.e., $C \geq \rho$); and
(iii) for each $C \in \chi_c$, insert the ordinary edge, $Z \xrightarrow{\rho} C$ (i.e., $C \leq \rho$).

Since operation (i), above, only removes edges from $\mathcal{G}_{\text{ou}}^*$, its only impact on the *AllMax* graph $\mathcal{G}_x^*$ is the possible weakening of constraints, which cannot introduce an inconsistency. For example, if E is some upper-case edge from X to Y that is removed from $\mathcal{G}_{\text{ou}}^*$, that might cause some other edge (or path) from X to Y in $\mathcal{G}_{\text{ou}}^*$ to become the new (longer) shortest path from X to Y in $\mathcal{G}_{\text{ou}}^*$, and its stripped counterpart in $\mathcal{G}_x^*$ to become the new (longer) shortest path from X to Y in the *AllMax* graph, but it cannot introduce any *shorter* paths into the *AllMax* graph. Thus, after all of the edge removals by operation (i), the graph $\mathcal{G}_x^*$ cannot have any negative loops.

Similarly, the updates in operation (ii), above, correspond to weakening constraints in the *AllMax* graph and, thus, cannot introduce any negative loops into $\mathcal{G}_x^*$. To see this, note that each remove-then-insert update is equivalent to *replacing* the original upper-case edge,

[26] For complete generality, each outcome involves a *set* of time-points that executes. However, in practice, these sets are often singleton sets.

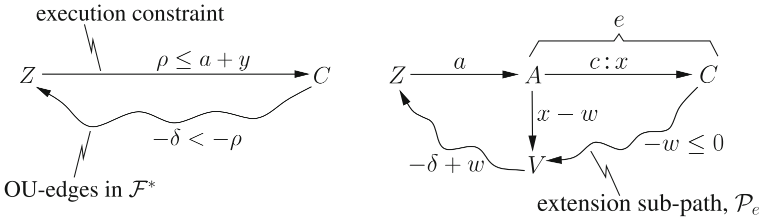


Fig. 12 The scenario described in Case 1a in the Proof of Theorem 8

$C \overset{C:-y}{\longrightarrow} A$, associated with some contingent link (A, x, y, C), by the weaker ordinary edge, $C \overset{-\rho+a}{\longrightarrow} A$, where $a = \xi_i(A)$. The execution semantics ensures that $\rho \leq a + y$ and, hence, that $-\rho + a \geq -y$. Thus, the net effect of this replacement on the path from C to A to Z in the graph $\mathcal{G}^*_{\text{ou}}$ is to *increase* its length from $-y-a$ to $(-\rho+a)-a = -\rho \geq -y-a$. (If $\rho = a+y$, then the path length stays the same.) This weakened path from C to A to Z of length $-\rho$ is equivalent to the execution constraint, $Z - C \leq -\rho$ (i.e., $C \geq \rho$). Increasing path lengths in $\mathcal{G}^*_{\text{ou}}$ cannot introduce negative loops into $\mathcal{G}^*_x$. Therefore, after all of the remove-then-insert updates by operation (ii), the graph $\mathcal{G}^*_x$ cannot have any negative loops.

Unlike the first two operations, operation (iii) typically involves the insertion of tighter constraints and, thus, the possibility of it introducing a negative loop into $\mathcal{G}^*_{\text{ou}}$—and, hence, into $\mathcal{G}^*_x$—must be checked. Toward that end, suppose that an upper-bound edge, $Z \overset{\rho}{\longrightarrow} C$, is the *first* such edge to introduce a negative loop into $\mathcal{G}^*_{\text{ou}}$ (and $\mathcal{G}^*_x$). Now, that can only happen if $\mathcal{G}^*_x$ contains a path from C to Z of some length $-\delta < -\rho$; and, hence, $\mathcal{G}^*_{\text{ou}}$ must contain a path from C to Z of length $-\delta < -\rho$. There are two sub-cases to consider.

Case 1a. The lefthand side of Fig. 12 illustrates the case where all of the edges in the path from C to Z belong to $\mathcal{F}^*$ and, hence, were present during the edge-generation phase, prior to any execution. Although $\mathcal{F}^*$ does not contain any lower-case edges, all lower-case edges are available for use during the edge-generation phase. Thus, for example, let e be the lower-case edge, $A \overset{c:x}{\longrightarrow} C$, associated with the contingent link, (A, x, y, C). Since the path from C to Z has negative length, some prefix of that path must be an extension sub-path $\mathcal{P}_e$ for e, as illustrated on the righthand side of Fig. 12, where $|\mathcal{P}_e| = -w \leq 0$. In addition, since all upper-case edges labeled by C were removed during operations (i) and (ii), the extension sub-path $\mathcal{P}_e$ cannot contain any breach edges. Therefore, $\mathcal{F}^*$ must also contain the (ordinary or upper-case) edge from A to V of length $x - w$, obtained by using $\mathcal{P}_e$ to "reduce away" the lower-case edge e, as shown in the figure.

Now, suppose that the generated edge from A to V was an upper-case edge labeled by some contingent time-point K. That could only happen if the moat edge (i.e., last edge) in $\mathcal{P}_e$ was itself an upper-case edge labeled by K. Since that moat edge is in $\mathcal{G}^*_{\text{ou}}$, then K must still be unexecuted, since otherwise all upper-case edges labeled by K would have been removed. But that implies that the generated edge from A to V, also labeled by K, must still belong to $\mathcal{G}^*_{\text{ou}}$. Thus, whether ordinary or upper-case, the generated edge from A to V must still belong to $\mathcal{G}^*_{\text{ou}}$. (Ordinary edges never get removed during execution.)

Next, since the execution semantics ensures that the activation time-point A executes before C, A must have already been executed in ξ_i. Therefore, the execution edge from Z to A of length a, where a is the execution time for A, must belong to $(\mathcal{G}^*_x)_i$. As a result, all of the edges in the loop from Z to A to V to Z must have been present in $(\mathcal{G}^*_x)_i$. But the length of this loop is $a + (x - w) + (-\delta + w) = a + x - \delta < a + x - \rho \leq \rho - \rho = 0$. This loop, being negative, contradicts the inductive hypothesis, $P(i)$.

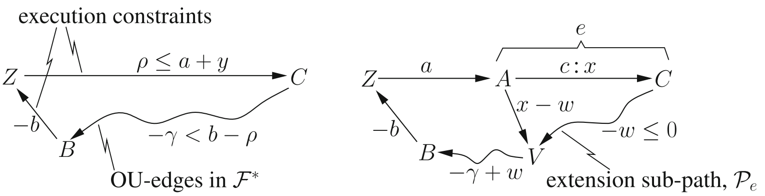


Fig. 13 The scenario described in Case 1b in the Proof of Theorem 8

Case 1b. The lefthand side of Fig. 13 illustrates the only other alternative. In this case, the path from C to Z that completes the negative loop in $\mathcal{G}_x^*$—and hence also in $\mathcal{G}_{\text{ou}}^*$—terminates in an execution edge from B to Z, where B is some time-point that executed at some time $b \le \rho$. Note that in the STN graph $\mathcal{G}_x^*$, the shortest path from C to Z can be assumed to involve only one occurrence of the time-point Z (cf. the Proof of Fact 1); therefore, the shortest path from C to Z in $\mathcal{G}_{\text{ou}}^*$ can similarly be assumed to involve only one occurrence of Z. Thus, without loss of generality, the path from C to B of length $-\gamma$ can be assumed to *not* include the time-point Z. Therefore, all of the edges in the path from C to B must belong to $\mathcal{F}^*$. Now, in this case, the length of the negative loop is $\rho - \gamma - b < 0$, which implies that $-\gamma < b - \rho \le 0$. Therefore, the path from C to B must contain an extension sub-path $\mathcal{P}_e$ for the lower-case edge e, as illustrated on the righthand side of Fig. 13. In addition, because all upper-case edges labeled by C were removed by operations (i) and (ii), $\mathcal{P}_e$ cannot contain any breaches. And, since all of the edges in $\mathcal{P}_e$ belong to $\mathcal{F}^*$, it follows that the (ordinary or upper-case) edge, $A \xrightarrow{x-w} V$, obtained by using $\mathcal{P}_e$ to "reduce away" the lower-case edge e must also belong to $\mathcal{F}^*$. As in Case 1a, if the generated edge from A to V is an upper-case edge, it cannot yet have been removed. But then all of the edges in the path from Z to A to V to B to Z must be present in $\mathcal{G}_{\text{ou}}^*$ prior to any of the updates done by operation (iii). Since the length of this path is $a + (x - w) + (-\gamma + w) - b = a + x - \gamma - b \le \rho - \gamma - b < 0$, which is negative, it contradicts that $\mathcal{G}_{\text{ou}}^*$ had no negative loops prior to the beginning of operation (iii).

From the above analyses, it follows that, after all the updates from operations (i), (ii) and (iii) have been completed, the graph $\mathcal{G}_x^*$ has no negative loops. Thus, the execution of contingent time-points has not caused the proposition $P(i + 1)$ to be violated.

Next, recall that values of the form, $\mathcal{D}_x^*(Z, D)$, where $D \in \mathcal{U}_x^{i+1}$, are relevant to the computation of t_U^{i+1}, whose value is the subject of the proposition $Q(i+1)$. Although updates from operations (i), (ii) and (iii) may change entries in the $\mathcal{D}_x^*$ matrix, it will be shown that each relevant $\mathcal{D}_x^*(Z, D)$ value remains strictly greater than $\texttt{now}_{\xi_{i+1}}$ at every stage of the updating process, in conformance with the proposition $Q(i + 1)$. Since, as already observed, updates from operations (i) and (ii) can only lengthen shortest paths in $\mathcal{G}_{\text{ou}}^*$ (and $\mathcal{G}_x^*$), any violation of $Q(i + 1)$ involving some $\mathcal{D}_x^*(Z, D) \le \texttt{now}_{\xi_{i+1}}$ must occur either before any updates have been made (i.e., when $\mathcal{D}_x^* = (\mathcal{D}_x^*)_i$) or as a result of an update from operation (iii).

First, note that if $\mathcal{U}_x^{i+1} = \emptyset$, then $t_U^{i+1} = \infty$, in which case $Q(i + 1)$ is guaranteed to hold no matter what updates are done. Therefore, suppose that $\mathcal{U}_x^{i+1} \neq \emptyset$, in which case, $\mathcal{U}_x^i \neq \emptyset$, since $\mathcal{U}_x^{i+1} \subseteq \mathcal{U}_x^i$. As a result, the decision $\hat{R}(\xi_i)$ generated by the strategy $\hat{R}$ for the partial schedule ξ_i must have the form (t, χ) for some time $t > \texttt{now}_{\xi_i}$. Furthermore, since the outcome represented by ξ_{i+1} involves the execution of at least one contingent time-point at some time ρ, it follows that $\rho = \texttt{now}_{\xi_{i+1}} \le t$.

Suppose, contrary to the goal, that for some $D \in \mathcal{U}_x^{i+1}$, $\mathcal{D}_x^*(Z, D) \le \rho$ at some point during the updating process (e.g., before any updates are done or as the result of an update from operation (iii)). Without loss of generality, consider the first point during the updating

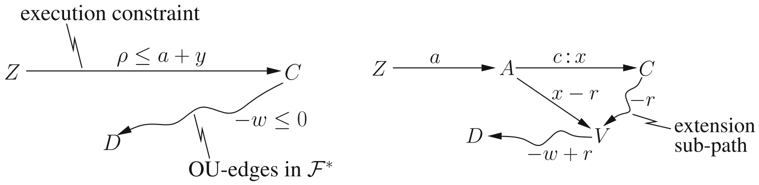


Fig. 14 The scenario described in Case 1d of the Proof of Theorem 8

process at which this occurs. For convenience, let $d = \mathcal{D}_x^*(Z, D)$ be the upper-bound for D in $\mathcal{G}_x^*$. Thus, $d = \mathcal{D}_x^*(Z, D) \leq \rho \leq t$; and there must exist a path from Z to D of length d in $\mathcal{G}_x^*$, and hence also in $\mathcal{G}_{\text{ou}}^*$.

Case 1c. Suppose that all edges in the upper-bound path from Z to D are in $(\mathcal{G}_{\text{ou}}^*)_i$. Now, by $P(i)$, there are no negative loops in $(\mathcal{G}_x^*)_i$ or $(\mathcal{G}_{\text{ou}}^*)_i$; thus, it follows that the lower bound for D in $(\mathcal{G}_x^*)_i$ satisfies: $-(\mathcal{D}_x^*)_i(D, Z) \leq d$. Now, if the value of t for the decision (t, χ) was derived from $t = t_L^i$ (cf. Definition 41), then $t = t_L^i \leq -(\mathcal{D}_x^*)_i(D, Z) \leq d \leq \rho \leq t$, from which it follows that $t = \rho$ and, therefore, that D is executed in ξ_{i+1}, contradicting that $D \in \mathcal{U}_x^{i+1}$. On the other hand, if t was derived from $t = (\texttt{now}_{\xi_i} + t_U^i)/2$, where $t_U^i > \texttt{now}_{\xi_i}$ by $Q(i)$, then $d \leq \rho \leq t = (\texttt{now}_{\xi_i} + t_U^i)/2 < t_U^i$, which contradicts the definition of t_U^i, since $D \in \mathcal{U}_x^{i+1} \subseteq \mathcal{U}_x^i$.

Case 1d. The only other alternative is that the upper-bound path in $\mathcal{G}_x^*$ from Z to D of length $d \leq \rho \leq t$ involves an upper-bound execution edge, $Z \xrightarrow{\rho} C$, introduced by an update from operation (iii) in response to a contingent time-point C executing at time ρ. Since that path is a shortest path in an STN graph, it can be assumed to have only one occurrence of the time-point Z. But then the corresponding path in $\mathcal{G}_{\text{ou}}^*$ from Z to C to D would require a path from C to D in $\mathcal{G}_x^*$ of some length, $-w \leq 0$, as illustrated on the lefthand side of Fig. 14, where the path from C to D must consist solely of OU-edges from $\mathcal{F}^*$. Since that path has non-positive length, and since operations (i) and (ii) removed all upper-case edges labeled by C, the path from C to D must contain a breach-free extension sub-path for the lower-case edge, $A \xrightarrow{c:x} C$, as shown on the righthand side of Fig. 14, where the extension sub-path goes from C to V and has length $-r \leq 0$. But this implies that $\mathcal{F}^*$ must also contain the (ordinary or upper-case edge) from A to V of length $x-r$ obtained by "reducing away" that lower-case edge, as shown in the figure. As in prior cases, if the generated edge from A to V is some upper-case edge labeled by K, then it must be that the moat edge in the extension sub-path is also an upper-case edge labeled by K; and since that moat edge has not yet been removed from $\mathcal{G}_{\text{ou}}^*$, the generated edge from A to V must still belong to $\mathcal{G}_{\text{ou}}^*$. In addition, since the activation time-point A must have already been executed in ξ_i, it follows that *all* of the edges in the OU-path from Z to A to V to D must be in $\mathcal{G}_{\text{ou}}^*$. The length of that path is: $a + (x-r) + (-w+r) = a + x - w \leq \rho - w \leq \rho$. But then the upper-bound for D in $(\mathcal{G}_x^*)_i$ satisfies: $(\mathcal{D}_x^*)_i(Z, D) \leq \rho \leq t$. But then the same arguments used in Case 1c, above, yield the same contradictions.

Since Cases 1c and 1d both lead to contradictions, it follows that after all of the updates from operations (i), (ii) and (iii), $\mathcal{D}_x^*(Z, D) > \texttt{now}_{\xi_{i+1}}$ holds for all $D \in \mathcal{U}_x^{i+1}$ (i.e., there are no violations of $Q(i+1)$ due to these updates).

Case 2: Executable executions, but no contingent executions In this case, the outcome ξ_{i+1} involves the execution of only executable time-points in some set χ at some time $t = \texttt{now}_{\xi_{i+1}}$. Therefore, the decision $\hat{R}(\xi_i)$ generated by the strategy $\hat{R}$ for the partial schedule ξ_i must have been (t, χ). The only updates to $\mathcal{G}_{\text{ou}}^*$ (and $\mathcal{G}_x^*$) in response to these execution events are the insertion of execution constraints of the form, $X = t$. It remains to show that these updates do not cause any violations of $P(i+1)$ or $Q(i+1)$. For convenience, it is assumed that all lower-bound edges, $X \xrightarrow{-t} Z$, are inserted first, followed by all upper-bound edges, $Z \xrightarrow{t} X$. There are two cases to consider, depending on how the value of t was generated for the decision $\hat{R}(\xi_i)$ (cf. Definition 41).

Case 2a: $t = t_L^i = \min\{-(\mathcal{D}_x^*)_i(X, Z) \mid X \in \mathcal{U}_x^i\}$. In this case, the lower-bound constraints for all time-points $X \in \chi$ were already present in $(\mathcal{G}_x^*)_i$ and, thus, inserting lower-bound edges of the form, $X \xrightarrow{-t} Z$, could not introduce any negative loops. In addition, each executable time-point $X \in \mathcal{U}_x^i$ that is *not* in χ must have a lower bound that is *greater than* $t_L^i = t = \texttt{now}_{\xi_{i+1}}$; otherwise, $\hat{R}$ would have put X into χ. Therefore, since the inductive hypothesis $P(i)$ ensures that $(\mathcal{G}_x^*)_i$ has no negative loops, the corresponding upper bound for X must also be greater than $t_L^i = t$. In other words, for each $X \in \mathcal{U}_x^{i+1}$, $(\mathcal{D}_x^*)_i(Z, X) > t = \texttt{now}_{\xi_{i+1}}$.

Next, suppose that some *upper-bound* edge, $Z \xrightarrow{t} X$, introduced a negative loop into $\mathcal{G}_{\text{ou}}^*$ (and $\mathcal{G}_x^*$). That would require a path in $\mathcal{G}_x^*$ (and hence also in $\mathcal{G}_{\text{ou}}^*$) from X to Z of some length $-\delta < -t$. Now, such a path may be presumed to have only one occurrence of Z (cf. the Proof of Fact 1); and since all lower-bound edges for any time-points that executed at time t were redundant, there must have already been such a path in $(\mathcal{G}_x^*)_i$. But then $-(\mathcal{D}_x^*)_i(X, Z) > t = t_L^i$ contradicts that $X \in \chi$. Thus, no upper-bound edge can introduce a negative loop into $\mathcal{G}_{\text{ou}}^*$ or $\mathcal{G}_x^*$.

Finally, suppose that the insertion of the upper-bound edge, $Z \xrightarrow{t} X$, into the graphs $\mathcal{G}_x^*$ and $\mathcal{G}_{\text{ou}}^*$ caused the upper bound of some still-unexecuted executable time-point $Y \in \mathcal{U}_x^{i+1}$ to become less than or equal to $\texttt{now}_{\xi_{i+1}} = t = t_L^i$. The only way this could happen is if there were a path from X to Y in $\mathcal{G}_x^*$ of some length $-w \leq 0$, yielding a path from Z to Y of length $t - w \leq t$. As discussed previously, such a path from Z to Y can be assumed to not involve multiple occurrences of Z, in which case the path from X to Y of length $-w \leq 0$ does not involve Z and, thus, consists of edges in $\mathcal{F}^*$, and hence in $(\mathcal{G}_x^*)_i$. But that would imply that the lower bound for Y in $(\mathcal{G}_x^*)_i$ must have been less than or equal to the lower bound for X (i.e., $-(\mathcal{D}_x^*)_i(Y, Z) \leq -(\mathcal{D}_x^*)_i(X, Z) = t_L^i = t$), contradicting that $X \in \chi$, while $Y \notin \chi$. Therefore, $\mathcal{D}_x^*(Z, Y) > t = \texttt{now}_{\xi_{i+1}}$, and no violation of proposition $Q(i+1)$ has occurred.

Case 2b: $t = (\texttt{now}_{\xi_i} + t_U^i)/2$. This case arises when $t_L^i \leq \texttt{now}_{\xi_i}$. Thus, some of the lower-bound edges being inserted into $\mathcal{G}_{\text{ou}}^*$ (and $\mathcal{G}_x^*$) may represent stronger constraints (i.e., shorter paths). Suppose that for some $X \in \chi$, inserting the lower-bound edge, $X \xrightarrow{-t} Z$, introduced a negative loop into $\mathcal{G}_x^*$. Choose the insertion that caused the first such negative loop. Since all prior insertions of lower-bound edges did not introduce a negative loop, they could not have changed the lengths of any paths emanating from Z (cf. Corollary 1). Therefore, they could not have caused any entries of the form, $\mathcal{D}_x^*(Z, X)$, to change. Therefore, inserting the lower-bound edge from X to Z could only cause a negative loop now if there were already a path from Z to X of length less than t in $\mathcal{G}_x^*$ (and $\mathcal{G}_{\text{ou}}^*$). Without loss of generality, such a path can be assumed to have only one occurrence of Z. Therefore, $(\mathcal{D}_x^*)_i(Z, X) < t$. Now, by $Q(i)$, $\texttt{now}_{\xi_i} < t_U^i$, which implies that $\texttt{now}_{\xi_i} < (\texttt{now}_{\xi_i} + t_U^i)/2 < t_U^i$. Thus, $t = (\texttt{now}_{\xi_i} + t_U^i)/2 < t_U^i$. But, by definition, $t_U^i \leq (\mathcal{D}_x^*)_i(Z, X)$. Thus, $t < t_U^i \leq (\mathcal{D}_x^*)_i(Z, X) < t$, which is a contradiction. Thus, no insertions of lower-bound edges can introduce a negative loop into $\mathcal{G}_x^*$. As a result, none of the insertions can change any upper bounds on any still-unexecuted executable time-points. Therefore, $\texttt{now}_{\xi_{i+1}} = t < t_U^i \leq \mathcal{D}_x^*(Z, Y)$, for all $Y \in \mathcal{U}_x^{i+1} \subseteq \mathcal{U}_x^i$. Therefore, no insertion of lower-bound edges can cause a violation of either $P(i+1)$ or $Q(i+1)$.

Next, consider the insertion of an upper-bound edge, $Z \xrightarrow{t} X$, into the graphs $\mathcal{G}^*_{\mathrm{ou}}$ and $\mathcal{G}^*_x$. Suppose this edge caused the first negative loop. That would require the existence of a path in $\mathcal{G}^*_x$ from X back to Z of length less than $-t$. Without loss of generality, that path can be assumed to have only the terminal occurrence of Z. If all the edges in that path were in $(\mathcal{G}^*_x)_i$, then the lower bound for X in $(\mathcal{G}^*_x)_i$ would have been greater than t, contradicting that the decision, $\hat{R}(\xi_i) = (t, \chi)$, involved executing X at time t. The only other possibility is that all of the edges in the path from X to Z are in $(\mathcal{G}^*_x)_i$ except for the terminal edge, which would have to be a lower-bound execution edge, $B \xrightarrow{-t} Z$, for some time-point $B \in \chi$ that also executed at time t. But then the path from X to B to Z could only have length less than $-t$ if the sub-path from X to B had negative length. But that would contradict that B and X had the same lower bound in $(\mathcal{G}^*_x)_i$, which is required if both were in χ. Thus, inserting the upper-bound edge cannot introduce a negative loop.

Finally, consider whether inserting an upper-bound edge, $Z \xrightarrow{t} X$, into $\mathcal{G}^*_x$ could cause $\mathcal{D}^*_x(Z, Y) \leq \mathtt{now}_{\xi_{i+1}}$ to hold for some $Y \in \mathcal{U}^{i+1}_x$. Since all shortest paths from Z to Y that begin with Z can be assumed to have no repeat occurrences of Z, that would require a path from X to Y of some length $-w \leq 0$, all of whose edges are in $(\mathcal{G}^*_x)_i$. But then $(\mathcal{D}^*_x)_i(X, Z) \leq (\mathcal{D}^*_x)_i(Y, Z)$, which means that the lower bound for X is at least as great as the lower bound for Y in $\mathcal{G}^*_x$, contradicting that $Y \notin \chi$. Thus, the insertion of an upper-bound edge cannot cause a violation of $Q(i+1)$.

As a result of the above analyses, after all of the updates associated with the execution of the executable time-points in χ at time t, there are no negative loops in $\mathcal{G}^*_x$, and for every $Y \in \mathcal{U}^{i+1}_x$, $\mathcal{D}^*_x(Z, Y) > \mathtt{now}_{\xi_{i+1}}$. Since $\mathcal{G}^*_x$ now equals $(\mathcal{G}^*_x)_{i+1}$ and $\mathcal{D}^*_x$ now equals $(\mathcal{D}^*_x)_{i+1}$, it follows that $P(i+1)$ and $Q(i+1)$ both hold.

Case 3: Simultaneous execution of contingent and executable time-points This case involves the simultaneous execution of the contingent time-points in some set χ_c and the executable time-points in some set χ, at some time $\rho = t$. The updates associated with the execution of the contingent time-points are carried out first, exactly as discussed in Case 1. After those updates have been done, the analysis of Case 1 shows that there are no negative loops in the updated $\mathcal{G}^*_x$, and for every $Y \in \mathcal{U}^{i+1}_x$, $\mathcal{D}^*_x(Z, Y) > \mathtt{now}_{\xi_{i+1}}$.

Next, the updates associated with the execution of the executable time-points are carried out. Since some contingent time-points have already executed, and the above conclusions are in hand, there turns out to be less work to do here. The following sub-cases are analogous to Cases 2a and 2b seen earlier.

Case 3a: $t = t^i_L > \mathtt{now}_{\xi_i}$. In this case, the insertion of a lower-bound edge, $X \xrightarrow{-t} Z$, must be redundant, since $t = t^i_L = -(\mathcal{D}^*_x)_i(X, Z)$. Thus, no negative loops can be introduced. In addition, since such an insertion cannot change the lengths of any shortest paths emanating from Z, the inequality, $\mathcal{D}^*_x(Z, Y) > \mathtt{now}_{\xi_{i+1}}$, must still hold for all $Y \in \mathcal{U}^{i+1}_x$.

Next, consider the insertion of an upper-bound edge, $Z \xrightarrow{t} X$. Showing that such an edge cannot introduce a negative loop can be handled as in Case 2a, except that there is one additional possibility to consider: suppose there is a path from X to Z of some length $-\delta < -t$ whose terminal edge is an execution constraint for some contingent time-point $C \in \chi_c$ that happened to execute at time t. Since that terminal edge has length $-t$, it follows that the path from X to C must have negative length. But then, in $(\mathcal{G}^*_{\mathrm{ou}})_i$, where the corresponding contingent time-point A must have already executed at some time a, it follows that there is an OU-path from C to A consisting of the original upper-case edge, $C \xrightarrow{C:-y} A$, followed by the execution edge, $A \xrightarrow{-a} Z$. Now, the length of this path is $-y - a \leq -t$, since the execution

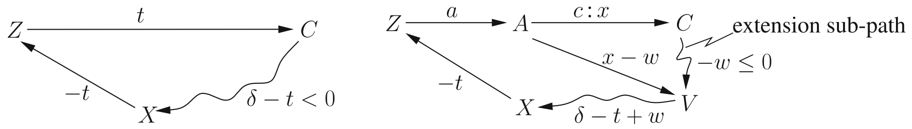


Fig. 15 The scenario described in Case 3b in the Proof of Theorem 8

semantics ensures that the contingent duration, $C - A \leq y$. But that implies that there was a path in $(\mathcal{G}^*_{\rm ou})_i$ from X to Z whose length is less than $-t$ (i.e., that the lower-bound for X in $(\mathcal{G}^*_{\rm ou})_i$ was greater than t), which contradicts that $X \in \chi$. Thus, no upper-bound edge can introduce a negative loop.

Finally, showing that no such upper-bound edge could cause $\mathcal{D}^*_x(Z, Y) \leq t = \texttt{now}_{\xi_{i+1}}$ for some $Y \in \mathcal{U}^{i+1}_x$ can be handled exactly as in Case 2a.

Case 3b: $t = (\texttt{now}_{\xi_i} + t^i_U)/2$. To show that the insertion of a lower-bound execution edge, $X \xrightarrow{-t} Z$, cannot introduce a negative loop into $\mathcal{G}^*_x$ can be handled as in Case 2b, except that there is one additional case to consider: suppose that the negative loop is completed by a path from Z to X of some length $\delta < t$, whose initial edge is an upper-bound execution edge, $Z \xrightarrow{t} C$, for some newly executed contingent time-point, $C \in \chi_c$, as illustrated on the lefthand side of Fig. 15. Since the path from Z to C to X has length $\delta < t$, it follows that the subsidiary path from C to X has length $\delta - t < 0$; and, furthermore, that all of its edges belong to $\mathcal{F}^*$. But then, assuming that the relevant contingent link is (A, x, y, C), it follows that the path from C to X contains a breach-free extension sub-path for the lower-case edge, $A \xrightarrow{c:x} C$, as illustrated on the righthand side of Fig. 15, where the extension sub-path goes from C to V and has length $-w \leq 0$. Reducing away the lower-case edge generates an (ordinary or upper-case) edge from A to V of length $x - w$. Furthermore, since the activation time-point A must have already executed, at some time a, it follows that all of the edges in the path from Z to A to V to X were in the graph $(\mathcal{G}^*_{\rm ou})_i$. But this path has length $a + (x - w) + (\delta - t + w) = a + x + \delta - t \leq t + \delta - t = \delta < t$, since the execution semantics ensures that $a + x \leq t$. But in that case, $t > (\mathcal{D}^*_x)_i(Z, X) \geq t^i_U > t$, which is a contradiction. Thus, no lower-bound execution edge can introduce a negative loop. In addition, as in Case 2b, the insertion of edges terminating at Z cannot change the lengths of shortest paths emanating from Z; thus, such edges cannot cause a violation of proposition $Q(i + 1)$.

Finally, consider the insertion of an upper-bound execution edge, $Z \xrightarrow{t} X$. To show that such an edge cannot introduce a negative loop into $\mathcal{G}^*_x$ can be handled as in Cases 2b and 3a. And to show that such an edge cannot cause $\mathcal{D}^*_x(Z, Y) \leq t = \texttt{now}_{\xi_{i+1}}$ for any $Y \in \mathcal{U}^{i+1}_x$ can be handled as in Case 2b.

In conclusion, the updates associated with the execution of contingent or executable time-points, in any combination, cannot introduce any negative loops into $\mathcal{G}^*_{\rm ou}$ or $\mathcal{G}^*_x$; and they cannot cause $\mathcal{D}^*_x(Z, Y) \leq \texttt{now}_{\xi_{i+1}}$ for any $Y \in \mathcal{U}^{i+1}_x$. Furthermore, by the time all of these updates have been done, $\mathcal{G}^*_{\rm ou}$ and $\mathcal{G}^*_x$ are equal to $(\mathcal{G}^*_{\rm ou})_{i+1}$ and $(\mathcal{G}^*_x)_{i+1}$, respectively; and $\mathcal{D}^*_x$ is equal to $(\mathcal{D}^*_x)_{i+1}$. Therefore, the propositions $P(i+1)$ and $Q(i+1)$ have been shown to hold. □

Theorem 9 (Fundamental theorem of STNUs) *Let $\mathcal{S}$ be any STNU; let $\mathcal{G}$ be its graph; and let $\mathcal{D}^*$ be its APSSRP matrix. Then the following are equivalent: (1) $\mathcal{S}$ is dynamically controllable; (2) $\mathcal{G}$ has no SRN loops; (3) the notion of shortest semi-reducible paths is well defined for $\mathcal{G}$; and (4) $\mathcal{D}^*$ has non-negative entries down its main diagonal.*

Proof (⇒) Suppose that condition (4) does not hold (i.e., that $\mathcal{D}^*$ has a negative entry along its main diagonal). In that case, there must be a negative loop in the *AllMax* graph (i.e., condition (2) does not hold), which corresponds to an SRN loop in $\mathcal{G}$. Furthermore, a negative loop in the *AllMax* graph implies that the notion of shortest paths is ill-defined for that graph which, in turn, implies that the notion of shortest semi-reducible paths is ill-defined for $\mathcal{G}$ (i.e., condition (3) does not hold). (Recall that semi-reducible paths in $\mathcal{G}$ correspond to ordinary paths in $\mathcal{G}_x^*$.) In addition, $\mathcal{G}$ having an SRN loop implies that $\mathcal{S}$ is not dynamically controllable, by Theorem 4 (i.e., condition (1) does not hold).

($\Leftarrow$) Suppose that condition (4) holds. Since $\mathcal{D}^*$ is the distance matrix for the *AllMax* STN, its having only non-negative entries down its main diagonal ensures that the notion of shortest path in $\mathcal{G}_x^*$ is well defined, by the fundamental theorem of STNs, and hence that the notion of shortest semi-reducible path in $\mathcal{G}$ is well defined (i.e., that condition (3) holds). In addition, only non-negative diagonal entries implies that $\mathcal{G}$ has no SRN loops (i.e., that condition (2) holds). And, finally, Theorem 8 ensures that $\mathcal{S}$ is dynamically controllable (i.e., condition (1) holds). □

All of the DC-checking algorithms that have been presented in the literature so far work by determining whether the corresponding graph has any SRN loops. Thus, all such algorithms depend on the fundamental theorem of STNUs. The presentation in this paper is the first comprehensive and rigorous treatment of the theory of STNUs and dynamic controllability aimed squarely at the fundamental theorem. By addressing important details that have been glossed over or missed altogether in prior work, this paper solidifies the theoretical foundations of STNUs and dynamic controllability.

5 Efficiently executing dynamically controllable STNUs

The proof of Theorem 8 provides a constructive algorithm for generating the execution decisions that form a reliable RTED-based execution strategy for any dynamically controllable STNU. The execution decisions depend on properly updating the graphs, $\mathcal{G}_{\text{ou}}^*$ and $\mathcal{G}_x^*$, as well as the *AllMax* matrix, $\mathcal{D}_x^*$. Once those updates have been completed for a given partial schedule, ξ_i, the actual computation of the execution decision, $\hat{R}(\xi_i)$, can be done in linear time. Thus, efficiently managing the updating of the graphs and the *AllMax* matrix is the key to efficiently managing the execution of a dynamically controllable STNU. Furthermore, it is the *removal* of upper-case edges in response to the execution of contingent time-points that dominates the computations.

In prior work, this author presented an execution algorithm, called NEW-EX, that updates the entire *AllMax* matrix after each execution event. Each NEW-EX update of $\mathcal{D}_x^*$ in response to the removal of upper-case edges uses $O(N^3)$ time, where N is the number of time-points in the network [11]. Thus, NEW-EX uses $O(N^4)$ time over an entire execution run: at most N updates at $O(N^3)$ time per update.

More recently, this author presented a faster execution algorithm, called FAST-EX, that requires only $O(N^3)$ time total: at most N updates at $O(N^2)$ time per update [12]. The faster performace is achieved by only updating the entries of $\mathcal{D}_x^*$ that involve the Z time-point, as described below. The FAST-EX algorithm uses $O(N^2)$ space. This section presents a modified version of the FAST-EX algorithm that more effectively (and correctly) organizes its computations by: (1) eschewing the unnecessary *greater-than-now* constraints for unexecuted time-points; and (2) generating RTEDs as described in Definition 41, which properly addresses the possibility that the removal of upper-case edges might allow lower bounds for some unexecuted time-points to slip into the past (cf. the case where $t_L \leq \texttt{now}_\xi$).

Overview of the FAST-EX algorithm A glance at Definition 41 reveals that the execution decisions, $\hat{R}(\xi)$, depend only on entries of $\mathcal{D}^*_x$ that involve Z. However, the removal of upper-case edges may not involve Z. The FAST-EX algorithm takes care of this by using a standard STN technique: reducing a *rigid component* down to a single point. In particular, as each time-point X executes, whether contingent or executable, it becomes rigid with Z (defined carefully below). As a result, any edges involving X can be re-oriented to interact with Z instead. Crucially, when an activation time-point A for a contingent link, (A, x, y, C), executes, all upper-case edges for C, which ordinarily point at A, get re-oriented to point at Z instead. Thus, by the time C executes and its upper-case edges must be removed, all of those edges are pointing at Z. In turn, that enables faster techniques for updating only the entries of $\mathcal{D}^*_x$ that involve Z.

The FAST-EX algorithm uses the following techniques:

- It only updates $\mathcal{D}^*_x$ entries that involve Z, because they are the only entries that are needed to generate execution decisions.
- It collapses down to a single point the *rigid component* consisting of Z and all executed time-points.
- Similarly to Johnson's algorithm [4], it uses a *potential function* to convert all edge-weights in the *AllMax* graph to non-negative values, thereby enabling Dijkstra's *single-source shortest-paths* (SSSP) algorithm [4] to be used to update the needed $\mathcal{D}^*_x$ entries.

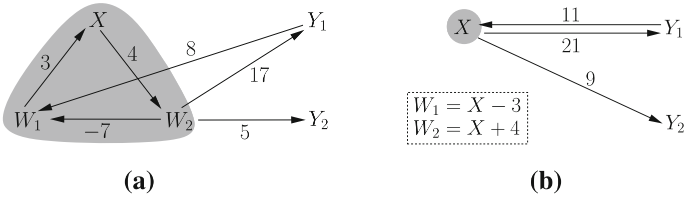
Each of these techniques is discussed in more detail below.

5.1 Managing the rigid component of executed time-points

Any time-points, X and Y, in an STN are *rigidly connected* if $\mathcal{D}(X, Y) + \mathcal{D}(Y, X) = 0$. In such a case, although there may be many consistent choices for the values of X and Y, they are not independent, because the value, $Y - X$, is restricted to a constant. For example, if $\mathcal{D}(X, Y) = 3$ and $\mathcal{D}(Y, X) = -3$, then X and Y must satisfy both $Y - X \leq 3$ and $X - Y \leq -3$ (i.e., $Y - X = 3$). Whatever X's value, Y must equal $X + 3$. More generally, any set, R, of rigidly connected time-points is called a *rigid component.*[27] Many researchers [7,8,33] have noted that since the time-points in a rigid component, R, are fixed relative to one another, the entire component can be effectively represented by a single time-point, as follows. First, let X be any time-point in R; X will serve as the representative. Next, re-orient any edges in the network that interact with time-points in R, as follows. Replace any edge, $Y \xrightarrow{\delta} W$, where $W \in R$, by an edge, $Y \xrightarrow{\delta+w} X$, where $w = \mathcal{D}(W, X)$. And replace any edge, $W \xrightarrow{\gamma} Y$, where $W \in R$, by an edge, $X \xrightarrow{\gamma-w} Y$, again, where $w = \mathcal{D}(W, X) = -\mathcal{D}(X, W)$. Figure 16 illustrates the collapsing of a rigid component containing time-points, X, W_1, and W_2.

At any time during execution, let $\mathcal{R}$ denote the set consisting of Z together with all of the already-executed time-points. Since the execution of each time-point causes it to become rigid with Z, the set $\mathcal{R}$ is a rigid component. Using the techniques described above, the FAST-EX algorithm represents the rigid component, $\mathcal{R}$, by a single point. For convenience, Z is chosen as the representative time-point for $\mathcal{R}$. As each time-point, X, executes, it joins $\mathcal{R}$ and is, thus, effectively removed from the network. In particular, edges that formerly pointed to X are re-directed toward Z; and edges that formerly emanated from X are re-directed to emanate from Z.

[27] Being rigidly connected is an equivalence relation; so, the notion of a rigid component is well defined.



(a) **(b)**

Fig. 16 **a** An STN with a rigid component. **b** The same STN after collapsing its rigid component

```
INPUTS:
    $\mathcal{G}$, a graph for a consistent STN
    $\mathcal{D}$, the distance matrix for $\mathcal{G}$, except that only the entries of the form, $\mathcal{D}(Z, X)$,
        for all $X$, are guaranteed to be correct.
OUTPUT:
    $\mathcal{D}$, updated so that entries, $\mathcal{D}(X, Z)$, for all $X$, are correct.
(1) For each $X$, let $h(X) = \mathcal{D}(Z, X)$.
(2) Create a new graph, $\mathcal{G}^\star$, whose edges correspond to those in $\mathcal{G}$, as follows. For each edge,
    $U \xrightarrow{w} V$, in $\mathcal{G}$, there is an edge, $U \xrightarrow{h(U)+w-h(V)} V$, in $\mathcal{G}^\star$.
(3) Run Dijkstra's single-sink-shortest-paths algorithm on $\mathcal{G}^\star$, using $Z$ as the single sink.
(4) For each $X$, set $\mathcal{D}(X, Z) = \mathcal{D}^\star(X, Z) - h(X)$.
```

Fig. 17 Pseudo-code for the *SinkDijkstra* algorithm

5.2 Using Dijkstra's algorithm to update $\mathcal{D}_x^*$

Unlike the NEW-EX algorithm, every edge that the FAST-EX algorithm adds to or removes from the various graphs during execution necessarily involves Z. This special feature enables a more efficient way of updating the desired distance-matrix entries.

Recall from Fact 1 that if X and Z are distinct time-points in an STN, and $\mathcal{P}$ is a shortest path from Z to X, then there exists a shortest path from Z to X that does not include any edges of the form, $Y \xrightarrow{\delta} Z$. As a result, it follows that whenever an edge of the form, $Y \xrightarrow{\delta} Z$, is added to (or removed from) an STN graph, it cannot affect the lengths of shortest paths whose source time-point is Z. In other words, adding such an edge to an STN (or removing it from the STN) cannot cause changes to distance-matrix entries of the form, $\mathcal{D}(Z, X)$. Thus, as in Johnson's algorithm [4], the values, $h(X) = \mathcal{D}(Z, X)$, can be used as a *potential function* to re-write the edge-weights in the graph so that they will all be non-negative. To see this, note that for any edge, $U \xrightarrow{w} V$, $h(V) \leq h(U) + w$, since the length of the shortest path from Z to V must be less than or equal to the length of the shortest path from Z to V via U. But then $h(U) + w - h(V) \geq 0$. Thus, a new graph, $\mathcal{G}^\star$, is created containing edges derived from those in the original graph, $\mathcal{G}$. In particular, each edge, $U \xrightarrow{w} V$, in the STN graph $\mathcal{G}$ gives rise to an edge, $U \xrightarrow{h(U)+w-h(V)} V$, in $\mathcal{G}^\star$.

Next, since all edge-weights in $\mathcal{G}^\star$ are non-negative, Dijkstra's single-*sink*-shortest-paths algorithm can be run using Z as the single sink/destination. Shortest path information in the new graph is easily translated into shortest path information in the original graph, as follows. For each time-point X, $\mathcal{D}(X, Z) = \mathcal{D}^\star(X, Z) - h(X)$, where $\mathcal{D}^\star$ is the distance matrix for the new graph $\mathcal{G}^\star$. (Of course, Dijkstra's algorithm does not compute *all* of the entries in $\mathcal{D}^\star$; it only computes those terminating in Z.)

For convenience, the technique just described is called *SinkDijkstra*, since it computes the lengths of shortest paths whose sink (or destination) is Z. Pseudo-code for the *SinkDijkstra* procedure is given in Fig. 17.

Similarly, whenever an edge of the form, $Z \xrightarrow{\delta} Y$, is added to (or removed from) an STN, it cannot affect the lengths of the shortest paths whose destination time-point is Z. That is, it cannot affect any entry of the form, $\mathcal{D}(X, Z)$. Thus, $h(X) = \mathcal{D}(X, Z)$ can be used as a potential function to re-write the edge-weights, to make them all non-negative, and Dijkstra's single-*source*-shortest-paths (SSSP) algorithm can be run on the new graph to generate all of the updated $\mathcal{D}(Z, X)$ values. This procedure, which is analogous to *SinkDijkstra,* is called *SourceDijkstra,* since it uses Z as its single source.

Since the only kinds of edges that are added to (or removed from) the *AllMax* graph, $\mathcal{G}_x^*$, during the execution of the time-points in an STNU are edges that involve Z, the FAST-EX algorithm uses *SinkDijkstra* and *SourceDijkstra*, in alternation, to compute the needed updates to the matrix, $\mathcal{D}_x^*$. The *SinkDijkstra* algorithm updates the values needed by the *SourceDijkstra* algorithm, and vice-versa.

5.3 Data structures used by FAST-EX

The FAST-EX algorithm uses multiple hash tables [4] to store the edges that belong to the *AllMax* graph.[28] In particular, for each time-point X, the hash table, $\mathit{Ins}(X)$, stores all of the edges in the *AllMax* graph that point at X (i.e., that have X as their destination). For example, an edge, $Y \xrightarrow{\delta} X$, would be stored in the hash table $\mathit{Ins}(X)$ with a key of Y, and a value of δ. Similarly, all of the edges in the *AllMax* graph that emanate from X (i.e., that have X as their source) are stored in a hash table, $\mathit{Outs}(X)$. Although each edge is stored in two hash tables, this bit of redundancy enables fast access.

Now, when a time-point X is executed at some time t, any edge whose destination is X must be moved from $\mathit{Ins}(X)$ to $\mathit{Ins}(Z)$, with its weight appropriately adjusted. Similarly, any edge whose source is X must be moved from $\mathit{Outs}(X)$ to $\mathit{Outs}(Z)$. These kinds of changes are also made to the FAST-EX algorithm's store of *ordinary* edges, as follows.

Prior to execution, the FAST-EX algorithm stores the *ordinary* edges from $\mathcal{F}^*$ in an N-by-N matrix, called the *ordinary matrix,* `Om`. In particular, for any time-points, X and Y, the entry, $\texttt{Om}(X, Y)$, equals the length of the ordinary edge from X to Y in $\mathcal{F}^*$. (If no such edge exists, then $\texttt{Om}(X, Y) = \infty$.) Because the FAST-EX algorithm re-directs edges in the process of collapsing the rigid component, $\mathcal{R}$, the entries in the `Om` matrix are similarly re-directed. For example, suppose that $\texttt{Om}(X, Y) = 22$ is an initial entry in the `Om` matrix, representing an ordinary edge, $X \xrightarrow{22} Y$. Now suppose that X is subsequently executed at time 5. Redirecting the above edge to emanate from Z yields the edge, $Z \xrightarrow{27} Y$. If the current entry, $\texttt{Om}(Z, Y)$, is greater than 27, representing a weaker constraint, it must be strengthened: $\texttt{Om}(Z, Y) := 27$. If the current entry is less than or equal to 27, representing a stronger constraint, no change to $\texttt{Om}(Z, Y)$ is made. Similar remarks apply to re-directing entries of the form, $\texttt{Om}(Y, X)$. Note that no additional propagation is done during this re-direction process.

Similarly, the *upper-case* edges from $\mathcal{F}^*$ are stored in a K-by-N matrix, called `UC`. For example, if $Y \xrightarrow{C:-w} A$ is the strongest upper-case edge from Y to A labeled by C, then $\texttt{UC}(C, Y) = -w$.[29] Note that no new upper-case edges are generated during the process of executing the network. In addition, to facilitate the search for replacement edges when upper-case edges are removed from the network, the FAST-EX algorithm does not do the

[28] If desired, the hash tables used by FAST-EX can be replaced by vectors and arrays.

[29] C is used as an index into the `UC` matrix instead of A, since multiple contingent links could have the same activation time-point, A.

kind of re-directing of entries in the $\texttt{UC}$ matrix that it does for the $\texttt{Om}$ matrix. Implications of this minor point are addressed in the next section.

The FAST-EX algorithm also uses hash tables to keep track of the execution status of each time-point, contingent or executable. And, of course, the *SinkDijkstra* and *SourceDijkstra* algorithms use priority queues.

5.4 Dealing with the removal of upper-case edges

The main contribution of the FAST-EX algorithm is its more efficient processing of the *removal* (or *weakening*) of edges from the *AllMax* graph, $\mathcal{G}_x^*$, that occurs whenever a contingent time-point is executed. Recall that for each contingent link, (A, x, y, C), the upper-case edges labeled by C (available in the $\texttt{UC}$ matrix) are stored—without any alphabetic labels—in the *AllMax* graph, using the *Ins* and *Outs* hash tables.[30] (Being an STN graph, the *AllMax* graph cannot distinguish ordinary and upper-case edges.) Now, before a contingent time-point C can execute, its activation time-point, A, must have already executed. Thus, A must have already joined the rigid component, $\mathcal{R}$, and all edges in the *AllMax* graph (i.e., in the *Ins* and *Outs* hash tables) that involve A must have already been re-directed to involve Z. Furthermore, since each upper-case edge labeled by C necessarily points at A (in the STNU graph), each edge in the *AllMax* graph that derives from an upper-case edge labeled by C must have already been re-directed to point at Z. Thus, removing the upper-case edges labeled by C from the STNU graph corresponds to removing (ordinary) edges from the *AllMax* graph that point at Z. This paves the way for the use of the *SinkDijkstra* procedure to compute the necessary updates to $\mathcal{D}$. However, before that can be done, the FAST-EX algorithm must deal with the possibility that the removal of an edge from some Y to Z in the *AllMax* graph might effectively *uncover* some previously longer edge that, due to the first edge's removal, now becomes the shortest edge from Y to Z.

Consider the following example. Suppose that, due to the contingent time-point C having just executed, an upper-case edge, $Y \xrightarrow{C:-9} A$, is to be removed from the STNU graph. Now, because A must have executed previously—say, at time 8—this edge, stripped of its label, might be in the *AllMax* graph as an ordinary edge from Y to Z of length -17. Removing this edge from the *AllMax* graph is necessary. But what should it be replaced by? Well, an entry, $\texttt{Om}(Y, Z) = -14$, in the matrix of ordinary edges would give rise to an edge, $Y \xrightarrow{-14} Z$, in the *AllMax* graph. Alternatively, any upper-case edge of the form, $Y \xrightarrow{C_i:-w_i} A_i$, for which the activation time-point A_i has already executed, but the contingent time-point C_i has not, also gives rise to an ordinary edge from Y to Z in the *AllMax* graph. (The length of that edge is $-w_i - a_i$, where a_i is the time at which A_i executed.) Whichever of these edges leads to the shortest edge from Y to Z in the *AllMax* graph is the one that needs to be inserted into the *Ins* and *Outs* hash tables as a replacement for the edge that was removed. The relevant entries are $\texttt{UC}(C_i, Y)$ for each activated-but-unexecuted contingent time-point C_i, as well as $\texttt{Om}(Y, Z)$. In short, the length of the replacement edge is given by: $\min\{\texttt{Om}(Y, Z), \min\{\texttt{UC}(C_i, Y) - a_i \mid C_i \notin \mathcal{R}, A_i \in \mathcal{R}\}\}$, where a_i denotes the execution time for A_i, the activation time-point for the activated-but-not-yet-executed contingent time-point C_i. Since there are at most K entries in the $\texttt{UC}$ matrix that need to be considered, finding the strongest replacement edge can be done in linear time.

Once the strongest replacement edge is found, then the *SinkDijkstra* procedure described earlier can be used to compute the updates to all of the $\mathcal{D}_x^*(X, Z)$ values.

[30] Only the shortest edge between each pair of time-points is stored in the *Ins* and *Outs* hash tables.

- Hash tables:

$\mathcal{U}_x$	: the unexecuted executable time-points
$\mathcal{U}_c$	: the unexecuted contingent time-points
$\mathcal{R}$	: the rigid component, containing executed time-points, initially only Z
$Ins(X)$	: for each X, the edges coming into X in $\mathcal{G}_x^*$
$Outs(X)$	: for each X, the edges leaving X in $\mathcal{G}_x^*$

- The N-by-N *AllMax* matrix, $\mathcal{D}_x^*$. Only the entries involving Z are used by FAST-EX.
- The *ordinary* edges from $\mathcal{F}^*$ are collected into an N-by-N matrix, called the *ordinary* matrix, `Om`.
- The *upper-case* edges from $\mathcal{F}^*$ are collected into a K-by-N matrix, called the *upper-case* matrix, `UC`.
- The variable, `now`, is initialized to $-\infty$.

Fig. 18 The initialization of data structures by the FAST-EX algorithm

```
IF $\mathcal{U}_x$ and $\mathcal{U}_c$ are both empty, THEN done, ELSE:
 1. Generate the next real-time execution decision, $RD$, as in Defn. 41.
 2. Observe the outcome of $RD$: $(\texttt{now}', \texttt{NewExec})$, where $\texttt{now}'$ is the time of the latest execution event and
    NewExec is the set of time-points that executed at $\texttt{now}'$. For convenience, let $t = \texttt{now}'$.
 3. For each newly executed contingent time-point $C$ (if any), remove each upper-case edge,
    $Y \xrightarrow{C:-w} A$, labeled by $C$, from the STNU graphs (i.e., from the UC matrix) and replace the cor-
    responding edge from $Y$ to $Z$ from the AllMax graph (i.e., the $Ins$ and $Outs$ hash tables) with the
    strongest replacement edge—that is, the edge, $Y \xrightarrow{\delta} Z$, where $\delta$ is the minimum of $\texttt{Om}(Y, Z)$ and
    one of at most $K$ entries in the UC matrix, as described in Sec. 5.4.
 4. For each newly executed time-point, $X \in \texttt{NewExec}$, add the lower-bound edge, $X \xrightarrow{-t} Z$.
 5. Run the SinkDijkstra procedure to update all entries of the form, $\mathcal{D}_x^*(V, Z)$, for any $V$.
 6. For each newly executed time-point, $X \in \texttt{NewExec}$, add the upper-bound edge, $Z \xrightarrow{t} X$.
 7. Run the SourceDijkstra procedure to update all entries of the form, $\mathcal{D}_x^*(Z, V)$, for any $V$.
 8. For each newly executd time-point, $X \in \texttt{NewExec}$, effectively remove $X$ from the network re-orienting
    any edges involving $X$ so that they instead involve $Z$, as discussed in Section 5.1. If $X$ is executable,
    move it from $\mathcal{U}_x$ to $\mathcal{R}$; otherwise, move it from $\mathcal{U}_c$ to $\mathcal{R}$.
 9. Go to the next iteration with $\texttt{now} := \texttt{now}'$.
```

Fig. 19 Pseudo-code for one iteration of the FAST-EX algorithm

5.5 Putting it all together

Given a dynamically controllable STNU; the corresponding STNU graphs $\mathcal{G}^*$ and $\mathcal{G}_{\text{ou}}^*$, which include the edges from $\mathcal{F}^*$; the corresponding *AllMax* graph $\mathcal{G}_x^*$; and the corresponding distance matrix, $\mathcal{D}^* = \mathcal{D}_x^*$, the FAST-EX algorithm initializes the structures shown in Fig. 18.

After initialization, the FAST-EX algorithm iteratively runs through the steps listed in Fig. 19. Note that the time-points in `NewExec` discussed in Steps 4, 6 and 8 of Fig. 19 may be contingent, executable or both.

5.6 Analysis of the FAST-EX algorithm

Theorem 10 *Given a dynamically controllable STNU $\mathcal{S}$, and the corresponding APSSRP matrix $\mathcal{D}^* = \mathcal{D}_x^*$, the FAST-EX algorithm correctly updates the STNU and* AllMax *graphs, and the matrix $\mathcal{D}_x^*$. Thus, by Theorem 8, FAST-EX will successfully execute the network, assuming that all contingent links satisfy their specified durational bounds. And the FAST-EX algorithm operates in $O(N^3)$ time overall: $O(N)$ updates at $O(N^2)$-time per update.*

Proof The proof addresses the correctness of FAST-EX, and then its time complexity.

Correctness of the FAST-EX algorithm. The FAST-EX algorithm uses the execution strategy $\hat{R}$ specified in Definition 41 which, by Theorem 8, is guaranteed to satisfy the constraints in the network in any situation, assuming that the contingent links satisfy their durational bounds, and assuming that the matrix $\mathcal{D}_x^*$ is correctly updated. Since the execution decisions computed by $\hat{R}$ depend only on the entries of $\mathcal{D}_x^*$ that involve Z, it suffices to show that the FAST-EX algorithm correctly computes the updates to those entries.

As in the Proof of Theorem 8, the FAST-EX algorithm maintains the graphs, $\mathcal{G}_{\mathrm{ou}}^*$ and $\mathcal{G}_x^*$. It uses the `Om` and `UC` matrices to respectively represent the ordinary and upper-case edges from the set $\mathcal{F}^*$, which provides the initial contents of $\mathcal{G}_{\mathrm{ou}}^*$; and it uses the *Ins* and *Outs* hash tables to represent the edges in $\mathcal{G}_x^*$. Inserting edges that correspond to execution constraints is straightforward: ordinary edges are inserted into both the `Om` matrix and the *Ins* and *Outs* hash tables. The correctness of the FAST-EX implementation of the removal of upper-case edges from $\mathcal{G}_{\mathrm{ou}}^*$ relies on the following observations. First, the edges in $\mathcal{G}_x^*$ are, by definition, the edges from $\mathcal{G}_{\mathrm{ou}}^*$, but without any upper-case labels. Therefore, removing an upper-case edge, $Y \xrightarrow{C:-w} A$, from $\mathcal{G}_{\mathrm{ou}}^*$ can only affect, if at all, the edge from Y to A in $\mathcal{G}_x^*$. Now there are at most $K + 1$ edges in $\mathcal{G}_{\mathrm{ou}}^*$ from Y to A: one ordinary edge and at most K upper-case edges—one for each contingent link. Therefore, whenever the FAST-EX algorithm removes an upper-case edge, $Y \xrightarrow{C:-w} A$, from the `UC` matrix, it need only look for the strongest replacement edge from among one entry in the `Om` matrix and at most K entries in the `UC` matrix. Thus, the FAST-EX algorithm correctly updates its representations of the $\mathcal{G}_{\mathrm{ou}}^*$ and $\mathcal{G}_x^*$ graphs.

After updating the $\mathcal{G}_{\mathrm{ou}}^*$ and $\mathcal{G}_x^*$ graphs, the FAST-EX algorithm does not re-compute the entire matrix $\mathcal{D}_x^*$ from scratch. Instead, it restricts attention to the entries of the form, $\mathcal{D}_x^*(Z, X)$ and $\mathcal{D}_x^*(X, Z)$, for each time-point X. Since all edges that are inserted or removed involve Z (see the next paragraph), the FAST-EX algorithm uses the *SinkDijkstra* and *SourceDijkstra* algorithms to correctly compute the necessary updates.

Before a contingent time-point C can execute, its corresponding activation time-point A must have already executed. Therefore, before any upper-case edges labeled by C are removed from $\mathcal{G}_{\mathrm{ou}}^*$, A will have already joined the rigid component involving Z; and each ordinary edge in the *AllMax* graph that derives from an upper-case edge labeled by C will have already been re-oriented in the *Ins* and *Outs* hash tables to effectively point at Z instead of A. Re-orienting edges in this way is well known to result in an equivalent STN graph [7,8,33]. Doing so ensures that all upper-case edges to be removed from the network effectively point at Z, thereby ensuring that the *SinkDijkstra* and *SourceDijkstra* algorithms will subsequently correctly update the entries of $\mathcal{D}_x^*$ that involve Z.

Time complexity of the FAST-EX algorithm The time complexity of the FAST-EX algorithm is dominated by its management of the removal of upper-case edges. For each edge removal, there are at most $K + 1$ possible replacement edges, as described above, each of which is examined in constant time. Since there are at most KN upper-case edges in the network at the start of execution, and no such edges are ever inserted during execution, the *total time* needed to find replacement edges for all of those upper-case edges when they are subsequently removed is $O(N^3)$: NK edges at $O(K)$ time per replacement.

The only other modifications to the STNU graphs involve the insertion of execution constraints for each newly executed time-point, which happens N times. When any time-point is executed, the *SinkDijkstra* and *SourceDijkstra* algorithms are run once each: $O(N \log N)$ per iteration. Thus, the overall time required for *all* of the runs of *SinkDijkstra* and *SourceDijkstra* is $O(N^2 \log N)$.

Since all other computations (e.g., generating each decision) can be done in linear time, the overall worst-case complexity of the FAST-EX algorithm is dominated by the $O(N^3)$-time replacement of upper-case edges when they are removed from the network. □

6 Conclusions

This paper presents a comprehensive, rigorous, and yet streamlined treatment of the theoretical foundations of STNUs and dynamic controllability, thus filling an important hole in the literature on STNUs. The presented theory combines work from a variety of sources, while also introducing novel approaches and proofs. The proof of the fundamental theorem of STNUs provides an execution strategy that is then efficiently implemented as the FAST-EX algorithm for managing the execution of a dynamically controllable network, the fastest execution algorithm presented so far.

While this paper was under review, Morris [17] presented an $O(N^3)$-time DC-checking algorithm that represents the new state of the art. Like all existing DC-checking algorithms, the new algorithm works by checking whether an STNU graph has any SRN loops and, thus, relies on the fundamental theorem. In addition to providing a DC-checking algorithm, Morris' paper also clarifies the relationships between the dynamic controllability of STNUs and prior work on the *dispatchability* of temporal networks [21,27,29].

There are many important avenues for future work involving STNUs and dynamic controllability. For example, there has been substantial interest recently in *incremental* DC-checking algorithms (i.e., algorithms that check whether inserting a new constraint into a DC network preserves its dynamic controllability) [14,24,25,28]. A thorough emprical evaluation and comparison of these incremental algorithms would be extremely useful for those seeking to incorporate STNUs into their applications.

A variety of extensions to the STNU framework have also been presented. For example, Rossi et al. [26] augmented STNUs to include soft temporal constraints; Morris et al. [20] considered STNUs with preferences and probabilities; Conrad and Williams [3] considered STNUs with *choice nodes;* Effinger et al. [6] presented temporally-flexible reactive programs; Venable et al. [31] studied STNUs with disjunctive constraints; and Hunsberger et al. [15] extended STNUs to include the *observation nodes* from prior work on the *Conditional Temporal Problem* (CTP) [30].

Other work has shown how to reduce the dynamic controllability problem for STNUs to a *reachability* problem for *Timed Game Automata* (TGAs), thereby clarifying the surprising relationships between these two very different formalisms [2]. Although that work does not provide faster DC-checking algorithms for STNUs, it has subsequently been extended to provide the first sound-and-complete DC-checking algorithm for a much wider class of temporal networks that allow disjunctive constraints and constraints conditioned on real-time observation of boolean propositions [1].

References

1. Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., Roveri, M.: Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In: Cesta, A., Combi, C., Laroussinie, F. (eds.) Proceedings of the 21st International Symposium on Temporal Representation and Reasoning (TIME-2014). IEEE (2014)
2. Cimatti, A., Hunsberger, L., Micheli, A., Roveri, M.: Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the 28th National Conference on Artificial Intelligence (AAAI-2014). AAAI Press (2014)
3. Conrad, P.R., Williams, B.C.: Drake: An efficient executive for temporal plans with choice and uncertainty. J. Artif. Intell. Res. **42**, 607–659 (2011)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2009)
5. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. **49**, 61–95 (1991)

6. Effinger, R., Williams, B., Kelly, G., Sheehy, M.: Dynamic controllability of temporally-flexible reactive programs. In: Gerevini, A., Howe, A., Cesta, A., Refanidis, I. (eds.) Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 09). AAAI Press (2009)
7. Gerevini, A., Perini, A., Ricci, F.: Incremental algorithms for managing temporal constraints. Technical Report IRST-9605-07, IRST (1996)
8. Hunsberger, L.: Group decision making and temporal reasoning. Ph.D. Thesis, Harvard University, Available as Harvard Technical Report TR-05-02 (2002)
9. Hunsberger, L.: A practical temporal constraint management system for real-time applications. In: Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N. (eds.) Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-2008). IOS Press, Amsterdam (2008)
10. Hunsberger, L.: Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In: Proceedings of the 16th International Symposium on Temporal Representation and Reasoning (TIME-2009), pp. 155–162. IEEE Computer Society (2009)
11. Hunsberger, L.: A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In: Proceedings of the 17th International Symposium on Temporal Representation and Reasoning (TIME-2010), Los Alamitos, CA, USA, pp. 121–128. IEEE Computer Society (2010)
12. Hunsberger, L.: A faster execution algorithm for dynamically controllable stnus. In: Sanchez, C., Venable, B.K., Zimanyi, E. (eds.) Proceedings of the 20th International Symposium on Temporal Representation and Reasoning (TIME-2013). IEEE Computer Society, Washington (2013)
13. Hunsberger, L.: Magic loops in simple temporal networks with uncertainty. In: Filipe, J., Fred, A. (eds.) Proceedings of the Fifth International Conference on Agents and Artificial Intelligence (ICAART-2013). SCITEPRESS (2013)
14. Hunsberger, L.: New techniques for checking dynamic controllability of simple temporal networks with uncertainty. Springer (Forthcoming)
15. Hunsberger, L., Posenato, R., Combi, C.: The dynamic controllability of conditional stns with uncertainty. In: Proceedings of the Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx) Workshop Associated with the ICAPS-2012 Conference, pp. 121–128 (2012)
16. Morris, P.: A structural characterization of temporal dynamic controllability. In: Principles and Practice of Constraint Programming (CP 2006), Volume 4204 of Lecture Notes in Computer Science, pp. 375–389. Springer, Berlin (2006)
17. Morris, P.: Dynamic controllability and dispatchability relationships. In: Integration of AI and OR Techniques in Constraint Programming—11th International Conference, CPAIOR 2014, Volume 8451 of Lecture Notes in Computer Science, pp. 464–479. Springer, Berlin (2014)
18. Morris, P., Muscettola, N., Vidal, T.: Dynamic control of plans with temporal uncertainty. In: Nebel, B. (ed.) 17th International Joint Conference on Artificial Intelligence (IJCAI-01), pp. 494–499. Morgan Kaufmann, Los Altos (2001)
19. Morris, P.H., Muscettola, N.: Temporal dynamic controllability revisited. In: Veloso, M.M., Kambhampati, S. (eds.) The Twentieth National Conference on Artificial Intelligence (AAAI-2005), pp. 1193–1198. The MIT Press, Cambridge (2005)
20. Morris, R., Morris, P., Khatib, L., Yorke-Smith, N.: Temporal constraint reasoning with preferences and probabilities. In: Brafman, R., Junker, U. (eds.) Proceedings of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling, pp. 150–155 (2005)
21. Muscettola, N., Morris, P., Tsamardinos, I.: Reformulating temporal plans for efficient execution. In Cohn, A.G., Schubert, L.K., Shapiro, S.C. (eds.) Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98), pp. 444–452. Morgan Kaufman, Los Altos (1998)
22. Nilsson, M., Kvarnström, J., Doherty, P.: Incremental dynamic controllability revisited. In: Borrajo, D., Kambhampati, S., Oddi, A., Fratini, S. (eds.) Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-2013). AAAI Press (2013)
23. Nilsson, M., Kvarnström, J., Doherty, P.: Classical dynamic controllability revisited: a tighter bound on the classical algorithm. In: Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART-2014), pp. 130–141. SCITEPRESS (2014)
24. Nilsson, M., Kvarnström, J., Doherty, P.: Efficientidc: a faster incremental dynamic controllability algorithm. In: Chien, S., Fern, A., Ruml, W., Do, M. (eds.) Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS-2014), pp. 199–207. AAAI Press (2014)
25. Nilsson, M., Kvarnström, J., Doherty, P.: Incremental dynamic controllability in cubic worst-case time. In: Cesta, A., Combi, C., Laroussinie, F. (eds.) Proceedings of the 21st International Symposium on Temporal Representation and Reasoning (TIME-2014). IEEE (2014)
26. Rossi, F., Venable, K.B., Yorke-Smith, N.: Uncertainty in soft temporal constraint problems: a general framework and controllability algorithms for the fuzzy case. J. Artif. Intell. Res. **27**, 617–674 (2006)

27. Shah, J., Stedl, J., Robertson, P., Williams, B.C.: A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In: Boddy, M., Fox, M., Thiébaux, S. (eds.) Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007). AAAI Press (2007)
28. Stedl, J., Williams, B.C.: A fast incremental dynamic controllability algorithm. In: Proceedings of the ICAPS Workshop on Plan Execution: A Reality Check, pp. 69–75 (2005)
29. Tsamardinos, I., Muscettola, N., Morris, P.: Fast transformation of temporal plans for efficient execution. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), pp. 254–261. The MIT Press, Cambridge (1998)
30. Tsamardinos, I., Vidal, T., Pollack, M.E.: CTP: a new constraint-based formalism for conditional, temporal planning. Constraints **8**(4), 365–388 (2003)
31. Venable, K.B., Volpato, M., Peintner, B., Yorke-Smith, N.: Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In: Proceedings of COPLAS 2010: ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems, pp. 50–59 (2010)
32. Vidal, T., Fargier, H.: Handling contingency in temporal constraint networks: from consistency to controllabilities. J. Exp. Theor. Artif. Intell. **11**(1), 23–45 (1999)
33. Wetprasit, R., Sattar, A.: Qualitative and quantitative temporal reasoning with points and durations (an extended abstract). In: Fifth International Workshop on Temporal Representation and Reasoning (TIME-98), pp. 69–73 (1998)