A Structural Characterization of Temporal Dynamic Controllability

Paul Morris

NASA Ames Research Center
Moffett Field, CA 94035, U.S.A.
pmorris@email.arc.nasa.gov

Abstract. An important issue for temporal planners is the ability to handle temporal uncertainty. Recent papers have addressed the question of how to tell whether a temporal network is *Dynamically Controllable*, i.e., whether the temporal requirements are feasible in the light of uncertain durations of some processes. Previous work has presented an $O(N^5)$ algorithm for testing this property. Here, we introduce a new analysis of temporal cycles that leads to an $O(N^4)$ algorithm.

1 Introduction

Many Constraint-Based Planning systems (e.g. [1]) use Simple Temporal Networks (STNs) to test the consistency of partial plans encountered during the search process. These systems produce *flexible* plans where every solution to the final Simple Temporal Network provides an acceptable schedule. The flexibility is useful because it provides scope to respond to unanticipated contingencies during execution, for example, where some activity takes longer than expected. However, since the uncertainty is not modelled, there is no guarantee that the flexibility will be sufficient to manage a particular contingency.

Many applications, however, involve a specific type of temporal uncertainty where the duration of certain processes or the timing of exogenous events is not under the control of the agent using the plan. In these cases, the values for the variables that are under the agent's control may need to be chosen so that they do not constrain uncontrollable events whose outcomes are still in the future. This is the *controllability* problem. By formalizing this notion of temporal uncertainty, it is possible to provide guarantees about the sufficiency of the flexibility.

In [2], several notions of controllability are defined, including *Dynamic Controllability* (DC). Roughly speaking, a network is dynamically controllable if there is a strategy for satisfying the constraints that depends only on knowing the outcomes of past uncontrollable events.

In [3] an algorithm is presented that determines DC and runs in polynomial time under the assumption that the maximum size of links in the STN is bounded. Thus, the algorithm is pseudo-polynomial like arc-consistency, rather than being a strongly polynomial algorithm such as, for example, the Bellman-Ford algorithm [4] for determining consistency of a distance graph. What makes

the latter algorithm strongly polynomial is the Bellman-Ford *cutoff*, which restricts the number of iterations based on the number of nodes in the network. The first strongly polynomial algorithm for DC is presented in [5]. This introduces an algorithm with an $O(N^3)$ inner-loop and an outer loop with an $O(N^2)$ cutoff. Thus, the entire algorithm runs in $O(N^5)$ time. The paper also simplifies the mathematical formulation of the reduction rules.

In this paper, we further simplify the mathematical formulation and introduce a structural characterization of DC in terms of the absence of a particular type of negative cycle. This is analogous to the result characterizing consistency of ordinary STNs in terms of the absence of negative cycles in the distance graph. This leads to a reformulated algorithm for DC with an $O(N^3)$ inner-loop and an $O(N)$ cutoff for the outer loop. Thus, the entire algorithm runs in $O(N^4)$ time.

2 Background

This background section defines the types of controllability, and outlines the previous DC algorithms, essentially following [3, 5].

A Simple Temporal Network (STN) [6] is a graph in which the edges are annotated with upper and lower numerical bounds. The nodes in the graph represent temporal events or *timepoints*, while the edges correspond to constraints on the durations between the events. Each STN is associated with a *distance graph* derived from the upper and lower bound constraints. An STN is consistent if and only if the distance graph does not contain a negative cycle. This can be determined by a single-source shortest path propagation such as in the Bellman-Ford algorithm [4] (faster than Floyd-Warshall for sparse graphs, which are common in practical problems). To avoid confusion with edges in the distance graph, we will refer to edges in the STN as *links.*

A Simple Temporal Network With Uncertainty (STNU) is similar to an STN except the links are divided into two classes, *requirement links* and *contingent links*. Requirement links are temporal constraints that the agent must satisfy, like the links in an ordinary STN. Contingent links may be thought of as representing causal processes of uncertain duration, or periods from a reference time to exogenous events; their finish timepoints, called *contingent timepoints*, are controlled by Nature, subject to the limits imposed by the bounds on the contingent links. All other timepoints, called *executable timepoints*, are controlled by the agent, whose goal is to satisfy the bounds on the requirement links. We assume the durations of contingent links vary independently, so a control procedure must consider every combination of such durations. Each contingent link is required to have positive (finite) upper and lower bounds, with the lower bound strictly less than the upper. Without loss of generality, we assume contingent links do not share finish points. (If desired, they can be constrained to simultaneity by $[0, 0]$ requirement links. It is also known that networks with coincident contingent finishing points cannot be DC.)

Choosing one of the allowed durations for each contingent link may be thought of as reducing the STNU to an ordinary STN. Thus, an STNU deter-

mines a family of STNs corresponding to the different allowed durations; these are called *projections* of the STNU.

Given an STNU with N as the set of nodes, a *schedule* T is a mapping

$$T : N \rightarrow \Re$$

where $T(x)$ is called the *time* of timepoint x. A schedule is *consistent* if it satisfies all the link constraints. The *prehistory* of a timepoint x with respect to a schedule T, denoted by $T\{\prec x\}$, specifies the durations of all contingent links that finish prior to x.

An *execution strategy* S is a mapping

$$S : \mathcal{P} \rightarrow \mathcal{T}$$

where $\mathcal{P}$ is the set of projections and $\mathcal{T}$ is the set of schedules. An execution strategy S is *viable* if $S(p)$, henceforth written S_p, is consistent with p for each projection p.

We are now ready to define the various types of controllability, following [7].

An STNU is *Weakly Controllable* if there is a viable execution strategy. This is equivalent to saying that every projection is consistent.

An STNU is *Strongly Controllable* if there is a viable execution strategy S such that

$$S_{p1}(x) = S_{p2}(x)$$

for each executable timepoint x and projections $p1$ and $p2$. In Strong Controllability, a "conformant" strategy (i.e., a fixed assignment of times to the executable timepoints) works for all the projections.

An STNU is *Dynamically Controllable* if there is a viable execution strategy S such that

$$S_{p1}\{\prec x\} = S_{p2}\{\prec x\} \Rightarrow S_{p1}(x) = S_{p2}(x)$$

for each executable timepoint x and projections $p1$ and $p2$. Thus, a Dynamic execution strategy assigns a time to each executable timepoint that may depend on the outcomes of contingent links in the past, but not on those in the future (or present). This corresponds to requiring that only information available from observation may be used in determining the schedule. We will use *dynamic strategy* in the following for a (viable) Dynamic execution strategy.

It is easy to see from the definitions that Strong Controllability implies Dynamic Controllability, which in turn implies Weak Controllability. In this paper, we are primarily concerned with Dynamic Controllability.

2.1 Previous Algorithms

It was shown in [3] that determining Dynamic Controllability is tractable, and an algorithm was presented that ran in pseudo-polynomial time. We will refer to this as the classic algorithm.

The classic algorithm involves repeated checking of a special consistency property called pseudo-controllability. An STNU is *pseudo-controllable* if it is

consistent in the STN sense and none of the contingent links are squeezed, where a contingent link is *squeezed* if the other constraints imply a strictly tighter lower bound or upper bound for the link. The pseudo-controllability property is tested by computing the AllPairs Shortest Path graph using Johnson's Algorithm [4]. If the network passes the test, the algorithm then analyzes triangles of links and possibly tightens some constraints in a way that has been shown not to change the status of the network as DC or non-DC, but makes explicit all limitations to the execution strategies due to the presence of contingent links.

Some of the tightenings involved a novel temporal constraint called a *wait*. Given a contingent link AB and another link AC, the $<\mathrm{B}, t>$ annotation on AC indicates that execution of the timepoint C is not allowed to proceed until after either B has occurred or t units of time have elapsed since A occurred. Thus, a wait is a ternary constraint involving A, B, and C. It may be viewed as a lower bound of t on AC that is interruptible by B. Note that the annotation resembles a binary constraint on AC.

In order to describe the tightenings, the notation $\mathrm{A} \overset{[x,y]}{\Longrightarrow} \mathrm{B}$ (or $\mathrm{B} \overset{[x,y]}{\Longleftarrow} \mathrm{A}$) indicates a contingent link with bounds $[x, y]$ between A and B. We use the similar notation of $\mathrm{A} \overset{[x,y]}{\longrightarrow} \mathrm{B}$ (or $\mathrm{B} \overset{[x,y]}{\longleftarrow} \mathrm{A}$) for ordinary links.

We can summarize the tightenings, called *reductions*, used in the classic algorithm as follows.

(*Precedes Reduction*) If $u \geq 0,\ y' = y - v,\ x' = x - u$,
$\mathrm{A} \overset{[x,y]}{\Longrightarrow} \mathrm{B} \overset{[u,v]}{\longleftarrow} \mathrm{C}$ adds $\mathrm{A} \overset{[y',x']}{\longrightarrow} \mathrm{C}$

(*Unordered Reduction*) If $u < 0, v \geq 0,\ y' = y - v$,
$\mathrm{A} \overset{[x,y]}{\Longrightarrow} \mathrm{B} \overset{[u,v]}{\longleftarrow} \mathrm{C}$ adds $\mathrm{A} \overset{<\mathrm{B}, y'>}{\longrightarrow} \mathrm{C}$

(*Simple Regression*) If $y' = y - v$,
$\mathrm{A} \overset{<\mathrm{B}, y>}{\longrightarrow} \mathrm{C} \overset{[u,v]}{\longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{<\mathrm{B}, y'>}{\longrightarrow} \mathrm{D}$

(*Contingent Regression*) If $y \geq 0, \mathrm{B} \neq \mathrm{C}$,
$\mathrm{A} \overset{<\mathrm{B}, y>}{\longrightarrow} \mathrm{C} \overset{[u,v]}{\Longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{<\mathrm{B}, y-u>}{\longrightarrow} \mathrm{D}$

(*"Unconditional" Reduction*) If $u \leq x$,
$\mathrm{B} \overset{[x,y]}{\Longleftarrow} \mathrm{A} \overset{<\mathrm{B}, u>}{\longrightarrow} \mathrm{C}$ adds $\mathrm{A} \overset{[u,\infty]}{\longrightarrow} \mathrm{C}$

(*General Reduction*) If $u > x$,
$\mathrm{B} \overset{[x,y]}{\Longleftarrow} \mathrm{A} \overset{<\mathrm{B}, u>}{\longrightarrow} \mathrm{C}$ adds $\mathrm{A} \overset{[x,\infty]}{\longrightarrow} \mathrm{C}$

The tightenings involve new links that are added when the given pattern is satisfied unless tighter links already exist. The extensive motivation for these in [3] cannot be repeated here due to lack of space. However, some examples may help to give the basic idea.

Example1: A $\overset{[1,2]}{\Longrightarrow}$ B $\overset{[1,1]}{\longleftarrow}$ C. Here we must schedule C exactly one time unit before B without knowing when B will occur. This requirement cannot be achieved in practical terms, although the network is initially consistent in the STN sense. The *Precedes Reduction* makes the inconsistency explicit. Contrast this with A $\overset{[1,2]}{\Longrightarrow}$ B $\overset{[1,1]}{\longrightarrow}$ C, where B can be observed before executing C, so no addition is needed.

Example2: A $\overset{[1,2]}{\Longrightarrow}$ B $\overset{[1,2]}{\longleftarrow}$ C. Note that the CB constraint implies C precedes B. This means the agent must decide on a timing for C before information about the timing of B is available, and must do it in a way that the CB constraint is satisfied no matter when B occurs. The only way to accomplish this given our ignorance of B is to constrain C relative to A in such a way that the CB constraint becomes redundant. The *Precedes Reduction* does this by constraining C to happen simultaneously with A.

Example3: A $\overset{[1,3]}{\Longrightarrow}$ B $\overset{[-1,1]}{\longleftarrow}$ C. Here we cannot safely execute C before B until time 2 after A (otherwise if B occurs at 3, the [-1,1] constraint would be violated). After that we can execute C prior to B if we wish, because we know B will finish within one more time unit. Thus, we place a $<\mathrm{B}, 2>$ constraint on AC.

2.2 Labelled Distance Graph and Cutoff Algorithm

We now review the developments in [5], which re-expresses the reductions in a more mathematically concise form.

An ordinary STN has an alternative representation as a *distance graph*, in which a link A $\overset{[x,y]}{\longrightarrow}$ B is replaced by two edges A $\overset{y}{\longrightarrow}$ B and A $\overset{-x}{\longleftarrow}$ B, where the y and $-x$ annotations are called *weights*. Edges with a weight of ∞ are omitted. The distance graph may be viewed as an STN in which there are only upper bounds. This allows shortest path methods to be used to determine consistency, since an STN is consistent if and only if the distance graph does not contain a cycle with negative total distance [6].

Similarly, there is an analogous alternative representation for an STNU called the *labelled distance graph* [5]. This is actually a multigraph (which allows multiple edges between two nodes), but we refer to it as a graph for simplicity. In the labelled distance graph, each requirement link A $\overset{[x,y]}{\longrightarrow}$ B is replaced by two edges A $\overset{y}{\longrightarrow}$ B and A $\overset{-x}{\longleftarrow}$ B, just as in an STN. For a contingent link A $\overset{[x,y]}{\Longrightarrow}$ B, we have the same two edges A $\overset{y}{\longrightarrow}$ B and A $\overset{-x}{\longleftarrow}$ B, but we also have two additional edges of the form A $\overset{b:x}{\longrightarrow}$ B and A $\overset{B:-y}{\longleftarrow}$ B. These are called *labelled edges* because of the additional "b:" and "B:" annotations indicating the contingent timepoint B with which they are associated. Note especially the reversal in the roles of x and y in the labelled edges. We refer to A $\overset{B:-y}{\longleftarrow}$ B and A $\overset{b:x}{\longrightarrow}$ B as *upper-case* and *lower-case* edges, respectively. Observe that the upper-case labelled weight B:-y gives the value the edge would have in a projection where the contingent link

takes on its maximum value, whereas the lower-case labelled weight corresponds to the contingent link minimum value.

There is also a representation for a $\mathrm{A} \xrightarrow{<\mathrm{B}, t>} \mathrm{C}$ wait constraint in the labelled distance graph. This corresponds to a single edge $\mathrm{A} \overset{\mathrm{B}:-\mathrm{t}}{\longleftarrow} \mathrm{C}$. Note the analogy to a lower bound. This weight is consistent with the lower bound that would occur in a projection where the contingent link has its maximum value.

We can now represent the tightenings in terms of the labelled distance graph. The first four categories of tightening from the classic algorithm are replaced by what is essentially a single reduction with different flavors. These are:

(Upper-Case Reduction)
$\mathrm{A} \overset{\mathrm{B}:\mathrm{x}}{\longleftarrow} \mathrm{C} \overset{\mathrm{y}}{\longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{\mathrm{B}:(\mathrm{x}+\mathrm{y})}{\longleftarrow} \mathrm{D}$

(Lower-Case Reduction) If $x \leq 0$,
$\mathrm{A} \overset{\mathrm{x}}{\longleftarrow} \mathrm{C} \overset{\mathrm{c}:\mathrm{y}}{\longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{\mathrm{x}+\mathrm{y}}{\longleftarrow} \mathrm{D}$

(Cross-Case Reduction) If $x \leq 0,\ \mathrm{B} \neq \mathrm{C}$,
$\mathrm{A} \overset{\mathrm{B}:\mathrm{x}}{\longleftarrow} \mathrm{C} \overset{\mathrm{c}:\mathrm{y}}{\longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{\mathrm{B}:(\mathrm{x}+\mathrm{y})}{\longleftarrow} \mathrm{D}$

(No-Case Reduction)
$\mathrm{A} \overset{\mathrm{x}}{\longleftarrow} \mathrm{C} \overset{\mathrm{y}}{\longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{\mathrm{x}+\mathrm{y}}{\longleftarrow} \mathrm{D}$

In place of the *Unconditional* and *General Reductions*, we will have a single reduction:

(Label Removal Reduction) If $z \geq -x$,
$\mathrm{B} \overset{\mathrm{b}:\mathrm{x}}{\longleftarrow} \mathrm{A} \overset{\mathrm{B}:\mathrm{z}}{\longleftarrow} \mathrm{C}$ adds $\mathrm{A} \overset{\mathrm{z}}{\longleftarrow} \mathrm{C}$

It is shown in [5] that the new reductions are sanctioned by the old ones. For example, Upper-Case Reduction follows from a combination of *Unordered Reduction* and *Simple Regression*.

We emphasize that the Cross-Case Reduction does not apply **when the upper and lower labels come from the same contingent link**. (This case violates the $\mathrm{B} \neq \mathrm{C}$ precondition.) This restriction is crucial; otherwise, the upper-case and lower-case edges of any contingent link could self-interact, immediately producing an inconsistency.

With this reformulation, the "Case" (first four) reductions can all be seen as forms of composition of edges, with the labels being used to modulate when those compositions are allowed to occur. In light of this, the *reduced distance* of a path in the labelled distance graph is defined to be the sum of edge weights in the path, ignoring any labels. Thus, the reductions preserve the reduced distance.

Observe that upper-case labels can apply to new edges as a result of reductions (but the targets of the edges do not change and always point to the start node of the contingent link), whereas the lower-case edges are fixed, i.e., the reductions do not produce new ones.

The approach in [5] also modifies the test that is applied before each iteration. Instead of testing for the complex property of pseudo-controllability, it checks for ordinary consistency of the *AllMax* projection, which is defined to be the projection where all the contingent links take on their maximum values. (Similarly, the AllMin projection is where all the contingent links take on their minimum values.) Observe that the distance graph of the AllMax projection can be obtained from the labelled distance graph by (1) deleting all lower-case edges, and (2) removing the labels from all upper-case edges.

Suppose we now take the classic algorithm for Dynamic Controllability, and modify it by replacing the old reductions/regressions with the new, and replacing the pseudo-controllability test with the AllMax consistency test. This modified algorithm correctly determines DC, and furthermore, if the network is DC, quiescence is reached after at most $O(N^2)$ iterations of the outer loop [5]. Thus, the algorithm can be halted at this cutoff bound. We will refer to this as the Quadratic-Cutoff algorithm.

The algorithm can be summarized as follows.

```
Boolean procedure determineDC()
  loop from 1 to Cutoff Bound do
    if AllMax projection inconsistent
       return false;
    Perform applicable Reductions;
    if no reductions were applicable
       return true;
  end loop;
  return false;
end procedure
```

The overall algorithm runs in $O(N^5)$ time. (A more precise $O(N^3K^2)$ bound is given [5] in terms of K, the number of contingent links. Note that $K \leq N$ since the end-points of contingent links are restricted to be distinct.)

2.3 Implicit Precondition

The *Precedes* reduction has an additional implicit precondition, B $\neq$ C, but this is not explicitly stated in [5]. (It should be noted that the results there are not affected by this issue.) This is because the derivation [3] of the *Precedes* reduction is in terms of a triangular network, which assumes three distinct nodes. It is worth pointing out that this B $\neq$ C precondition is necessary. It can easily be seen, for example, that the network A $\overset{[2,4]}{\Longrightarrow}$ B $\overset{[0,0]}{\longrightarrow}$ B has a dynamic strategy (just execute A at time 0), and hence is DC. However, without the precondition, an application of the *Precedes* reduction would produce an inconsistency. Similarly, in the Lower-Case reduction (which is derived from the *Precedes* reduction), there is an implicit A $\neq$ C precondition.

Instead of adding this precondition explicitly, we will make a different modification to the Dynamic Controllability formulation that makes it unnecessary.

Recall that a dynamic strategy may depend on the past, but not on the future *or present*. We change this so that it may depend on the past **or present**. This essentially assumes that observations can be acted upon instantaneously instead of requiring an infinitesimal amount of time. This change is NOT essential to the results in this paper; they could be derived without it. However, the mathematics works out more cleanly with the change. It is also more consistent with the approach used in the dispatchability [8] work.

The effect of this change is that the Lower-Case and Cross-Case reductions must be modified to read as follows (note the $x \leq 0$ is changed to $x < 0$):

(Lower-Case Reduction) If $x < 0$,
$\mathrm{A} \xleftarrow{\mathrm{x}} \mathrm{C} \xleftarrow{\mathrm{c:y}} \mathrm{D}$ adds $\mathrm{A} \xleftarrow{\mathrm{x+y}} \mathrm{D}$

(Cross-Case Reduction) If $x < 0,\ \mathrm{B} \neq \mathrm{C}$,
$\mathrm{A} \xleftarrow{\mathrm{B:x}} \mathrm{C} \xleftarrow{\mathrm{c:y}} \mathrm{D}$ adds $\mathrm{A} \xleftarrow{\mathrm{B:(x+y)}} \mathrm{D}$

We will assume in the remainder of this paper that the Lower-Case and Cross-Case reductions have been modified in this way. The Upper-Case and No-Case reductions do not require modification.

3 Structural Characterization

We now proceed to introduce a new analysis of Dynamic Controllability that leads to a faster algorithm.

3.1 Normal Form STNU

In this subsection, we introduce a new way of simplifying the STNU formulation. First, we recall that in the definition of an STNU [3], the bounds on a contingent link $\mathrm{A} \overset{[x,y]}{\Longrightarrow} \mathrm{B}$ are required to satisfy $0 < x < y < \infty$. An analysis of the proof of correctness in [3] shows that the strict $0 < x$ inequality was only needed because of a weakness of the pseudo-controllability test in detecting a deadlock involving a cycle of waits, and the resulting use of the *General Reduction* for this purpose. In [5], the pseudo-controllability test is replaced by a test of the consistency of the AllMax projection. This can detect a cycle of waits even when contingent links are allowed to have lower bounds of zero. Thus, we can relax the contingent link bound requirement to $0 \leq x < y < \infty$.

This provides an opportunity to recognize that we can restrict our attention to a simpler subclass of STNUs without loss of generality. We will say an STNU is in *normal form* if the lower bound of every contingent link is zero. Now consider a general STNU Γ and any contingent link $\mathrm{A} \overset{[x,y]}{\Longrightarrow} \mathrm{B}$ in Γ where $x > 0$. Suppose we create a new STNU Γ' where the $\mathrm{A} \overset{[x,y]}{\Longrightarrow} \mathrm{B}$ contingent link is replaced by $\mathrm{A} \xrightarrow{\mathrm{[x,x]}} \mathrm{C} \overset{\mathrm{[0,y-x]}}{\Longrightarrow} \mathrm{B}$, where C is a new controllable timepoint. It is not difficult to see that any dynamic strategy for Γ can be easily mapped into a dynamic

strategy for Γ' (just execute C at x units after A) and vice versa (just drop C). Thus, Γ is DC if and only if Γ' is DC. The replacement process can be repeated until every contingent link with a non-zero lower-bound has been eliminated. Thus, for any STNU, there is a normal form STNU that is equivalent in terms of the existence of a dynamic strategy. We will assume in our subsequent analysis that the STNUs are in normal form.

Note that the LABEL REMOVAL reduction assumes a simpler form in a normal form STNU as follows. (This facilitates our subsequent proofs.)

(LABEL REMOVAL) If $z \geq 0$,
$\mathrm{A} \xleftarrow{\mathrm{B:z}} \mathrm{C}$ adds $\mathrm{A} \xleftarrow{\mathrm{z}} \mathrm{C}$

It is also worth commenting that with the normal form assumption, the c:x notation could be "recycled" to mean a c:0 edge followed by a path of ordinary edges of length x. This would allow the LOWER-CASE and CROSS-CASE reductions to be rewritten as follows.

(LOWER-CASE COMPOSITION)
$\mathrm{A} \xleftarrow{\mathrm{x}} \mathrm{E} \xleftarrow{\mathrm{c:y}} \mathrm{D}$ adds $\mathrm{A} \xleftarrow{\mathrm{c:(x+y)}} \mathrm{D}$

(LOWER LABEL REMOVAL) If $z < 0$,
$\mathrm{A} \xleftarrow{\mathrm{c:z}} \mathrm{D}$ adds $\mathrm{A} \xleftarrow{\mathrm{z}} \mathrm{D}$

(CROSS-CASE COMPOSITION) If $\mathrm{B} \neq \mathrm{C}$,
$\mathrm{A} \xleftarrow{\mathrm{B:x}} \mathrm{E} \xleftarrow{\mathrm{c:y}} \mathrm{D}$ adds $\begin{cases} \mathrm{A} \xleftarrow{\mathrm{B:(x+y)}} \mathrm{D} \text{ if } (x+y) < 0 \\ \mathrm{A} \xleftarrow{\mathrm{c:(x+y)}} \mathrm{D} \text{ if } (x+y) \geq 0 \end{cases}$

We will not pursue this notation change here, but it is interesting to note a type of symmetry between lower-case and upper-case labels.

3.2 Path Transformations

An ordinary STN is consistent if and only if its distance graph does not contain a negative cycle. It is tempting to suppose that Dynamic Controllability might be characterized by the absence of cycles of negative reduced distance in the labelled distance graph. However, this is not true in general. For example, the STNU consisting of the single contingent link $\mathrm{A} \overset{[0,4]}{\Longrightarrow} \mathrm{B}$ is DC, but its distance graph contains the cycle $\mathrm{A} \xrightarrow{\mathrm{b:0}} \mathrm{B} \xrightarrow{\mathrm{B:-4}} \mathrm{A}$, which has negative reduced distance. Nevertheless, as we will see, there is indeed a characterization of DC in terms of negative cycles, but it involves a subclass of such cycles. In order to describe this, we require additional concepts involving a notion of path transformation.

Consider a path $\mathcal{P}$ that contains a subpath $\mathcal{Q}$ between two points A and B and suppose $\mathcal{Q}$ matches the left side of a reduction. Note that applying the reduction to $\mathcal{Q}$ yields a new edge e between A to B. Now consider the path $\mathcal{P}'$ obtained from $\mathcal{P}$ by replacing $\mathcal{Q}$ by e. For convenience, we will abuse language

slightly and say $\mathcal{P}$ is *transformed* into $\mathcal{P}'$ by the reduction. (The original path $\mathcal{P}$ is of course still in the network.) Note that $\mathcal{P}'$ has the same reduced distance as $\mathcal{P}$ since the reductions preserve reduced distance.

Armed with this linguistic device, we can define some useful concepts of reducibility of paths. First, a path is *reducible* if it can be transformed into a single edge by a sequence of reductions.

However, a slightly weaker property is more useful for characterizing Dynamic Controllability. Recall that tests of the consistency of the AllMax projection are used to filter non-DC networks in the Quadratic Cutoff algorithm. Note also that the AllMax projection includes edge weights derived from both ordinary edges and upper-case edges, but not from lower-case edges. We may view the reductions as gradually tightening the network by transforming reduced distance in the labelled distance graph into ordinary distance in the AllMax projection. The significant events in this process are the transformations of paths with lower-case edges into paths without lower-case edges. This leads us to define a path as being *semi-reducible* if it can be transformed into a path without lower-case edges by a sequence of reductions. This gives rise to the following theorem. (To simplify its statement, we will informally say an STNU has a negative cycle if its labelled distance graph contains a cyclic path with negative reduced distance.)

Theorem 1. *An STNU is Dynamically Controllable if and only if it does not have a semi-reducible negative cycle.*

Proof. If an STNU is not DC, then there is some sequence of reductions that produces a negative cycle in the AllMax projection, i.e., a lower-case-free negative cycle in the labelled distance graph. If we now unwind that sequence of reductions (applying the reverse transformations to the negative cycle), we arrive at a preimage or *precursor* cycle in the original labelled distance graph. Since the reductions preserve reduced distance, this is also negative, and clearly it is semi-reducible.

Conversely, if there is a semi-reducible negative cycle, then clearly there is a sequence of reductions that produces an inconsistency in the AllMax projection. Thus, the STNU is not DC. □

Observe that the cycle A $\xrightarrow{\text{b:0}}$ B $\xrightarrow{\text{B:}-4}$ A in our earlier example is not semi-reducible since no reductions are applicable. (The Cross-Case reduction does not apply since the b and B labels are from the same contingent link.)

We now look for ways of identifying semi-reducible paths. The following notation will be useful. Consider a specific path $\mathcal{P}$ in the labelled distance graph of an STNU. We will write $e < e'$ in $\mathcal{P}$ if e is an earlier edge than e' in $\mathcal{P}$. If A and B are nodes in the path, we will write $\mathcal{D}_{\mathcal{P}}(\mathrm{A}, \mathrm{B})$ for the reduced distance from A to B in $\mathcal{P}$. We denote the start and end nodes of an edge e by $\mathrm{start}(e)$ and $\mathrm{end}(e)$, respectively.

Now suppose e is a lower-case edge in $\mathcal{P}$. Let e' be some other edge such that $e < e'$ in $\mathcal{P}$. We will say e' is a *drop edge* for e in $\mathcal{P}$ if $\mathcal{D}_{\mathcal{P}}(\mathrm{end}(e), \mathrm{end}(e')) < 0$. We further say e' is a *moat edge* for e in $\mathcal{P}$ if it is a *drop edge* and there is no other drop edge e'' such that $e'' < e'$ in $\mathcal{P}$). (Thus, a moat edge is a closest drop

edge. The metaphor is of a moat in front of a castle.) Note that a lower-case edge can never be a moat edge since it is non-negative. We will also call the subpath of $\mathcal{P}$ from end(e) to end(e') the *extension* of e in $\mathcal{P}$.

The extension subpath turns out to have a very useful property. We will say a path $\mathcal{P}$ has the *prefix/postfix property* if every nonempty proper prefix of $\mathcal{P}$ has non-negative reduced distance and every nonempty proper postfix of $\mathcal{P}$ has negative reduced distance. We will also refer to such a path as a *prefix/postfix path*. Observe that the extension subpath of a lower-case edge always has the prefix/postfix property. (Otherwise there would be a closer drop edge than the moat edge.) The following lemma will be useful.

Lemma 1 (Nesting Lemma). *If two prefix/postfix paths have a non-empty intersection, then one of the paths is contained in the other.*

Proof. The intersection subpath is a postfix of one path and a prefix of the other. It cannot be proper in both cases; otherwise it would be both non-negative and negative, which is a contradiction. Thus, it must be equal to one of the paths, which must then be a subpath of the other. □

The significance of an extension subpath, as we will see, is that it can eventually be used to "reduce away" the lower-case edge from the path. However, there is an exceptional case where we will show this cannot occur. Suppose e is a lower-case edge in a path and e' is a moat edge for e. We will say e' is *unusable* if e' is the upper-case edge from the same contingent link as e. This prepares the way for the following fundamental theorem.

Theorem 2. *A path $\mathcal{P}$ is semi-reducible if and only if every lower-case edge in $\mathcal{P}$ has a usable moat edge in $\mathcal{P}$.*

Proof. First, suppose $\mathcal{P}$ is semi-reducible. Let e be any lower-case edge in $\mathcal{P}$. Then there must be some sequence of transformations on $\mathcal{P}$ that eliminates e, i.e., e must eventually participate in a lower-case or cross-case reduction with some negative edge e' that is derived by a sequence of transformations on $\mathcal{P}$. If we unwind this sequence, we can identify a precursor subpath $\mathcal{Q}$ of $\mathcal{P}$ that will eventually be transformed to e'. Let e'' be the final edge of $\mathcal{Q}$. Since the reductions preserve reduced distance, it follows that $\mathcal{D}_{\mathcal{P}}(\text{end}(e), \text{end}(e'')) = \mathcal{D}_{\mathcal{P}}(\text{end}(e), \text{end}(e')) < 0$. Thus, e'' is a drop edge for e and hence e must have a moat edge e'''.

Next, suppose the moat edge is not usable, i.e., e''' is the upper-case edge that comes from the same contingent link as e. Note that every postfix of the extension (proper or non-proper) of e is negative. It is not hard to see that this rules out any "clearing" of the upper-case label from $\mathcal{Q}$ via the label removal reduction (as modified for a normal form STNU). This implies e' will also have that label. But this prevents application of the cross-case reduction to eliminate e, which is a contradiction. It follows that every lower-case edge in $\mathcal{P}$ has a usable moat edge.

Conversely, suppose that every lower-case edge in $\mathcal{P}$ has a usable moat edge in $\mathcal{P}$. Consider the extension subpaths corresponding to all the lower-case edges

in $\mathcal{P}$. By the Nesting Lemma, these are either nested or disjoint, i.e., they fall into nested groups. We will say an extension is *innermost* if it is not contained in another extension. It is enough to show that we can transform $\mathcal{P}$ to eliminate the lower-case edges of the innermost extensions; the result will then follow by induction since the other extensions will become innermost after the lower-case edges of the extensions nested within them have been eliminated.

Now consider any innermost extension $\mathcal{E}$ of a lower-case edge e. Since all the proper prefixes of $\mathcal{E}$ are non-negative, it follows that any upper-case labels in the interior of $\mathcal{E}$ can be "cleared" by applying no-case, upper-case and label removal reductions in a left-to-right manner. The only possible upper-case edge remaining will be the moat edge e'. Since this is usable, either e' is an ordinary edge, or e' is an upper-case edge from a different contingent link than e. Thus, $\mathcal{E}$ will eventually reduce to an e'' that is either an ordinary edge or an upper-case edge from a different contingent link than e. Since $\mathcal{E}$ has negative reduced distance, the e'' can participate in a reduction that eliminates e. □

We can make two important observations from the converse part of the proof of theorem 2. First, by the nesting lemma, the lower-case edge and moat edge pairs, which fall into nested groups, form a layering of a semi-reducible path. Since the lower-case edges and moat edges behave like left and right parentheses, we call this the *parenthesization* of the path. The second observation is that there is a standard way of performing the transformations, using the parenthesization, that is guaranteed to eliminate the lower-case edges from a semi-reducible path. We call this the *canonical elimination*.

3.3 Complexity of Negative Cycles

Our next task is to analyze the complexity of semi-reducible negative cycles. In the case of an ordinary STN, if it has any negative cycle, then it must have a simple (without any repetitions) negative cycle. This allows the Bellman-Ford algorithm to limit the extent of its propagation. Unfortunately, a similar result does not hold for semi-reducible negative cycles in an STNU. The problem is that (if it is non-simple) there is no guarantee that one of its component cycles will also be both negative and semi-reducible, as seen in the following example. The compound cycle

$$\mathrm{B} \xrightarrow{B:-2} \mathrm{A} \xrightarrow{b:0} \mathrm{B} \xrightarrow{1} \mathrm{D} \xrightarrow{D:-3} \mathrm{C} \xrightarrow{d:0} \mathrm{D} \xrightarrow{3} \mathrm{B} \xrightarrow{B:-2} \mathrm{A} \xrightarrow{b:0} \mathrm{B} \xrightarrow{-2} \mathrm{E} \xrightarrow{4} \mathrm{B}\ ,$$

which is semi-reducible and negative, can be broken into two component cycles $\mathrm{B} \xrightarrow{B:-2} \mathrm{A} \xrightarrow{b:0} \mathrm{B} \xrightarrow{1} \mathrm{D} \xrightarrow{D:-3} \mathrm{C} \xrightarrow{d:0} \mathrm{D} \xrightarrow{3} \mathrm{B}$ and $\mathrm{B} \xrightarrow{B:-2} \mathrm{A} \xrightarrow{b:0} \mathrm{B} \xrightarrow{-2} \mathrm{E} \xrightarrow{4} \mathrm{B}$. However, the first is negative but not semi-reducible, while the second is semi-reducible but not negative. (Note that the CD edge in the first cycle has its moat edge BE in the second cycle.)

The good news is that there are nevertheless some simplifications that we can apply to a semi-reducible negative cycle, and they do lead to a faster DC checking algorithm. We require some additional concepts. First, given a lower-case edge e in a semi-reducible path, we will say e has a *breach* if its extension

contains the upper-case edge from the same contingent link as e. Second, suppose a lower-case edge e repeats in a semi-reducible path. By the nesting lemma, the extensions from the two occurrences of e must be either nested or disjoint. We will say a repetition is *flat* if the two extensions are disjoint. (In the example, the repetition of AB is flat.) We have the following result.

Theorem 3. *If an STNU has any semi-reducible negative cycle, then it has a breach-free semi-reducible negative cycle in which all the repetitions are flat.*

Proof. First, we will show that breaches can be eliminated. Consider any outermost extension $\mathcal{E}$ associated with a lower-case edge e and its moat edge e' and suppose it has a breach edge e''. Then $e'' \neq e'$. (Otherwise the moat edge would not be usable.) Thus, $\mathcal{D}_{\mathcal{P}}(\text{end}(e), \text{end}(e'')) \geq 0$ by the prefix/postfix property, and so $\mathcal{D}_{\mathcal{P}}(\text{start}(e), \text{end}(e'')) \geq 0$. Since e and e'' are the lower-case and upper-case edges of the same contingent link, start(e)=end(e''). Now observe that if we tighten the cycle by deleting the portion between start(e) and end(e''), we will not affect the moat edges of any remaining lower-case edges. (Since $\mathcal{E}$ does not lie inside any other extension.)

Now suppose by induction that we have eliminated breaches in all extensions that contain a given extension $\mathcal{E}$. We can apply the same breach elimination process as before. This may tighten some extension $\mathcal{E}'$ containing $\mathcal{E}$ such that the former moat edge for $\mathcal{E}'$ is no longer the closest drop edge. However, since $\mathcal{E}'$ has no breaches, the new moat edge will still be usable. Thus, we can eliminate the breach from $\mathcal{E}$, while preserving the property that every lower-case edge has a usable moat edge. By induction, we can eliminate all breaches while preserving this property. This leads to a new tighter (thus, still negative) cycle in which every lower-case edge still has a usable moat edge. Thus, it is still semi-reducible by theorem 2.

Next suppose we have a breach-free semi-reducible negative cycle $\mathcal{P}$, and consider a repetition that is not flat, i.e., we have occurrences of lower-case edges e_1 and e_2 with associated extensions $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively, such that $\mathcal{E}_1$ contains $\mathcal{E}_2$, and $e_1 = e_2$. By the prefix/postfix property, $\mathcal{D}_{\mathcal{P}}(\text{start}(e_1), \text{start}(e_2)) \geq 0$. In this case, we can tighten the cycle by deleting the subpath between start(e_1) and start(e_2). Since the cycle is breach-free, every lower-case edge will still have a usable moat edge (by a similar argument as previously). Thus, the cycle will still be semi-reducible. □

The significance of theorem 3 is that if the repetitions are all flat, then the depth of nesting of the extensions cannot be greater than K, where K is the number of contingent links. We now fashion a DC checking algorithm that takes advantage of this. The idea is that each iteration of a propagation phase will decrement the depth of nesting by eliminating the innermost extensions. Thus, at most K iterations will be required to detect some semi-reducible negative cycle if an STNU is not DC. The propagation phase essentially simulates the canonical elimination mentioned earlier: we propagate forward from each lower-case edge over breach-free and lower-case-free paths looking for moat edges. For each one we find, we add a new edge corresponding to the reduction of the extension to a single edge.

The algorithm can be summarized as follows.

```
Boolean procedure fastDCcheck()
  loop from 1 to K do
    if AllMax projection inconsistent return false;
    loop for each lower-case edge e do
      Propagate forward from end(e) over allowed paths
      loop for each moat edge e' found do
        add reduced edge from start(e) to end(e')
      end loop;
    end loop;
  end loop;
  if AllMax projection inconsistent return false;
  return true;
end procedure
```

We now estimate the complexity of this algorithm. For this, we let N be the number of nodes, E be the number of edges, and K be the number of contingent links. First, we observe that we need only propagate over the shortest paths among the allowed paths. (The only consequence will be possible earlier discovery of tighter reduced edges.) Second, a Bellman-Ford propagation that determines consistency of the AllMax projection can be used to provide a potential function as in Johnson's algorithm [4]. Thus, the shortest path propagations from the lower-case edges can use the $O(E + N \log N)$ Dijkstra algorithm. The overall cost of the algorithm is then $O(K(EN + K(E + N \log N))) = O(KEN + K^2E + K^2N \log N)$. At most KN edges are added during the course of the algorithm. Thus, E is bounded by $E_0 + NK$, where E_0 is the original number of edges. This gives an overall estimate of $O(KE_0N + K^2E_0 + K^3N + K^2N^2 + K^2N \log N)$. Using $K \leq N$ and $E_0 \leq N^2$, we can simplify that to $O(N^4)$, which compares favorably with the previous $O(N^5)$ algorithm.

4 Execution

It should be pointed out that fastDCcheck merely determines the status of an STNU. It does not provide a network suitable for the execution algorithm described in [3]. However, once DC has been confirmed, it is an easier matter to prepare the network for execution. Due to space limitations, we can only outline the approach without providing detailed proofs.

Successful execution requires that no contingent link bounds are squeezed due to propagation when a timepoint is executed or a contingent link finishes. To ensure that contingent link upper-bounds are not squeezed, we see from [3] that the key requirement is that waits need to be regressed along both ordinary and lower-case edges as far as they will go. This means that a regressWaits algorithm analogous to fastDCcheck that works backwards from upper-case edges (using Upper and Cross Case reductions), adds new waits and ordinary edges (the latter using Label Removal), and uses an AllMin propagation in each iteration

to construct the potential function, can be used to regress the waits. An argument that is symmetrically similar to the drop/moat edge analysis can be used to show that quiescence is reached within K iterations.

Once the waits have been regressed, we need to ensure that contingent link lower-bounds are also not squeezed. For this, we observe that propagations during execution are only along ordinary edges. (The waits merely introduce delays.) Thus, we need to ensure that paths that begin with lower-case edges and continue with ordinary edges are transformed to bypass the lower-case edges via the Lower Case reduction. This can be achieved by applying a modified fastDCcheck algorithm where the "allowed" paths are restricted to ordinary edges. With that restriction, the proof methods of this paper can be adapted to show that quiescence is reached within K iterations. It can also be shown that the edges added in this step will not disturb the quiescence of the wait regression.

Both of these post-processing steps run in similar time to fastDCcheck. Thus, the combined algorithm is still $O(N^4)$.

5 Conclusion

We have reformulated Dynamic Controllability testing in a way that provides mathematically simpler operations, characterized DC in terms of the absence of a certain type of negative cycle, and used that to obtain a linear cutoff leading to an $O(N^4)$ algorithm. Previously, only an $O(N^5)$ algorithm was known.

Acknowledgement We thank Nicola Muscettola for discussions that contributed to these results.

References

1. Muscettola, N., Nayak, P., Pell, B., Williams, B.: Remote agent: to boldly go where no AI system has gone before. Artificial Intelligence **103**(1-2) (1998) 5–48
2. Vidal, T., Fargier, H.: Handling contingency in temporal constraint networks: from consistency to controllabilities. JETAI **11** (1999) 23–45
3. Morris, P., Muscettola, N., Vidal, T.: Dynamic control of plans with temporal uncertainty. In: Proc. of IJCAI-01. (2001)
4. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. MIT press, Cambridge, MA (1990)
5. Morris, P., Muscettola, N.: Dynamic controllability revisited. In: Proc. of AAAI-05. (2005)
6. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49** (1991) 61–95
7. Vidal, T.: Controllability characterization and checking in contingent temporal constraint networks. In: Proc. of Seventh Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000). (2000)
8. Tsamardinos, I., Muscettola, N., Morris, P.: Fast transformation of temporal plans for efficient execution. In: Proc. of Fifteenth Nat. Conf. on Artificial Intelligence (AAAI-98). (1998)