Temporal Dynamic Controllability Revisited

Paul Morris and **Nicola Muscettola**
NASA Ames Research Center
Moffett Field, CA 94043
(pmorris|mus)@email.arc.nasa.gov

Abstract

An important issue for temporal planners is the ability to handle temporal uncertainty. We revisit the question of how to determine whether a given set of temporal requirements are feasible in the light of uncertain durations of some processes. In particular, we consider how best to determine whether a network is Dynamically Controllable, i.e., whether a dynamic strategy exists for executing the network that is guaranteed to satisfy the requirements. Previous work has shown the existence of a pseudo-polynomial algorithm for testing Dynamic Controllability. Here, we simplify the previous framework, and present a strongly polynomial algorithm with a termination criterion based on the structure of the network.

Introduction

For some time, Constraint-Based Planning systems (e.g. (Muscettola *et al.* 1998)) have been using 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. Many applications, however, involve 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.

Progress has been made in this area in recent years. In (Vidal & Fargier 1999), 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 (Morris, Muscettola, & Vidal 2001) 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. The method involves repeated tightenings based on a consideration of "triangles" (i.e., node triples) in the network. Termination is guaranteed because the maximum link bound ensures that some domain will become empty after a bounded number of iterations. Thus, the iteration is $O(N^3)$, where N is the number of nodes in the network. However, the apparent low-order polynomial is misleading, because for many applications the link bound (and hence the number of iterations) may be very large in practical terms. In the parlance of complexity theory, the algorithm is pseudo-polynomial like arc-consistency, rather than being a strongly polynomial algorithm. An example of the latter category is the well-known Bellman-Ford algorithm (Cormen, Leiserson, & Rivest 1990), which can determine whether a distance graph has a negative cycle.

The constraint propagation process underlying Bellman-Ford can be viewed as enforcing arc-consistency. What makes the algorithm strongly polynomial is the Bellman-Ford *cutoff*, which restricts the number of iterations based on the number of nodes in the network. In this paper, we derive an analogous cutoff method for Dynamic Controllability checking. We also present several other improvements to the approach of (Morris, Muscettola, & Vidal 2001). The treatment there involves numerous distinct concepts, including diverse reduction and regression operations, that are substantially unified in the present paper. The algorithm also required repeated checks of a special consistency property, which involved recomputation of the AllPairs network after every iteration. In this paper, that is replaced by a standard incremental consistency check. We also show how to reformulate the iterations so they visit a restricted subset of triangles, which reduces the complexity.

In this paper, we revisit the foundations of DC reasoning. Other recent work has focused on combining DC reasoning with preferences (Rossi, Venable, & Yorke-Smith 2004) or probabilities (Tsamardinos, Pollack, & Ramakrishnan 2003).

Background

We restate the basic definitions, as described in (Morris, Muscettola, & Vidal 2001), and summarize the algorithm presented there.

A Simple Temporal Network (STN) (Dechter, Meiri, & Pearl 1991) 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[1] (Cormen, Leiserson, & Rivest 1990). 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 determines 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 (Vidal 2000).

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.

[1]Faster than Floyd-Warshall for sparse graphs, which commonly occur in practical problems.

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.

Classic Algorithm

It was shown in (Morris, Muscettola, & Vidal 2001) that determining Dynamic Controllability is tractable, and an algorithm was presented that ran in polynomial time under the assumption that the sizes of links were bounded above and below. (As discussed in the introduction, this may be called pseudo-polynomial.) We will refer to this in the rest of paper as the *Classic Dynamic Controllability algorithm,* or classic algorithm for short.

The classic algorithm involved 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 was tested by computing the AllPairs Shortest Path graph using Johnson's Algorithm (Cormen, Leiserson, & Rivest 1990). If the network passed the test, the algorithm then analyzed triangles of links and possibly tightened some constraints in a way that was shown not to change the status of the network as DC or non-DC, but made explicit all limitations to the execution strategies due to the presence of contingent links. Thus, we can summarize the classic algorithm as follows.

```
Boolean procedure determineDC()
loop
  if not pseudo-controllable
     return false;
  else for every triangle ABC
     if needed, tighten ABC;
  if no tightenings were found
     return true;
end loop;
end procedure
```

Termination relied on the fact that if quiescence is not reached, then continued tightenings will eventually empty some domain. Thus, the complexity is $O(UN^3)$, taking into account the (constant) bound U on the domain size, the N^3 cost of each round of triangular tightenings, and the $O(EN + N^2 \log N)$ cost of Johnson's Algorithm.

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. The waits were not used directly to compute pseudo-controllability, but could result in additional binary constraints.

In order to describe the tightenings, we introduce the notation A $\overset{\text{[x,y]}}{\Longrightarrow}$ B (or B $\overset{\text{[x,y]}}{\Longleftarrow}$ A) to indicate a contingent link with bounds $[x, y]$ between A and B. We use the similar notation of A $\xrightarrow{\text{[x,y]}}$ B (or B $\xleftarrow{\text{[x,y]}}$ A) for ordinary links.

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

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

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

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

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

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

(*General Reduction*) If $u > x$,
B $\overset{\text{[x,y]}}{\Longleftarrow}$ A $\xrightarrow{<\mathrm{B}, u>}$ C adds A $\xrightarrow{[\text{x},\infty]}$ 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 (Morris, Muscettola, & Vidal 2001) cannot be repeated here due to lack of space. However, some examples may help to give the basic idea.

Example1: A $\overset{\text{[1,2]}}{\Longrightarrow}$ B $\xleftarrow{\text{[1,1]}}$ 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 $\overset{\text{[1,2]}}{\Longrightarrow}$ B $\xrightarrow{\text{[1,1]}}$ C, where B can be observed before executing C, so no addition is needed.

Example2: A $\overset{\text{[1,2]}}{\Longrightarrow}$ B $\xleftarrow{\text{[1,2]}}$ 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 $\overset{\text{[1,3]}}{\Longrightarrow}$ B $\xleftarrow{[-1,1]}$ 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.

The *"Unconditional" Reduction* is so-called because in this situation the full impact of the ternary $<\mathrm{B}, u>$ constraint is captured by the binary $[u, \infty]$ bound without having to split it into cases. By contrast, in the *General Reduction* the inferred binary constraint is weaker than the ternary constraint.

We also note that in the classic algorithm, the tightenings are applied to edges in the AllPairs graph (computed as part of the determination of pseudo-controllability). However, they are valid for any edges.

More Uniform Reductions

To make progress in improving the algorithm, it is helpful to seek a more uniform formulation of the reductions. Although possibly less intuitive, this is easier to work with mathematically.

As mentioned earlier, an ordinary STN has an alternative representation as a *distance graph*, in which a link A $\xrightarrow{\text{[x,y]}}$ B is replaced by two edges A $\xrightarrow{\text{y}}$ B and A $\xleftarrow{-\text{x}}$ 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 compute consistency (Dechter, Meiri, & Pearl 1991).

In this paper we introduce an analogous alternative representation for an STNU called the *labelled distance graph*. This is actually a multigraph (which allows multiple edges between two nodes), but we refer to it as a graph in this paper for simplicity. In the labelled distance graph, each requirement link A $\xrightarrow{\text{[x,y]}}$ B is replaced by two edges A $\xrightarrow{\text{y}}$ B and A $\xleftarrow{-\text{x}}$ B, just as in an STN. For a contingent link A $\overset{\text{[x,y]}}{\Longrightarrow}$ B, we have the same two edges A $\xrightarrow{\text{y}}$ B and A $\xleftarrow{-\text{x}}$ B, but we also have two additional edges of the form A $\xrightarrow{\text{b:x}}$ B and A $\xleftarrow{\text{B:}-\text{y}}$ 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 the reversal in the roles of x and y in the labelled edges. We refer to A $\xleftarrow{\text{B:}-\text{y}}$ B and A $\xrightarrow{\text{b:x}}$ B as *upper-case* and *lower-case* edges, respectively. Note that the upper-case label 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 label corresponds to the contingent link minimum value.

We also provide a representation for a A $\xrightarrow{<\mathrm{B}, t>}$ 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. Also note that this is consistent with the lower bound that would occur in a projection where the contingent link has its maximum value.

We now introduce new 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)
$\mathrm{A} \overset{\mathrm{B:x}}{\longleftarrow} \mathrm{C} \overset{\mathrm{y}}{\longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{\mathrm{B:(x+y)}}{\longleftarrow} \mathrm{D}$

(LOWER-CASE REDUCTION) If $x \leq 0$,
$\mathrm{A} \overset{\mathrm{x}}{\longleftarrow} \mathrm{C} \overset{\mathrm{c:y}}{\longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{\mathrm{x+y}}{\longleftarrow} \mathrm{D}$

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

(NO-CASE REDUCTION)
$\mathrm{A} \overset{\mathrm{x}}{\longleftarrow} \mathrm{C} \overset{\mathrm{y}}{\longleftarrow} \mathrm{D}$ adds $\mathrm{A} \overset{\mathrm{x+y}}{\longleftarrow} \mathrm{D}$

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

(LABEL REMOVAL REDUCTION) If $z \geq -x$,
$\mathrm{B} \overset{\mathrm{b:x}}{\longleftarrow} \mathrm{A} \overset{\mathrm{B:z}}{\longleftarrow} \mathrm{C}$ adds $\mathrm{A} \overset{\mathrm{z}}{\longleftarrow} \mathrm{C}$

It is straightforward to see that the new reductions are sanctioned by the old ones. First note that, as applied to B:x in a wait, the UPPER-CASE and CROSS-CASE REDUCTIONS are simple transliterations to the new notation of the *Simple* and *Contingent Regressions*, respectively. As applied to B:x in the representation of a contingent link (i.e., C=B in this case), the UPPER-CASE REDUCTION follows for $y \geq 0$ from the *Unordered Reduction* using [$-\infty$,y] as the bound. If $y < 0$, then D is after C (i.e. B), so the wait is trivially satisfied. Note that the CROSS-CASE REDUCTION will never be applied to a B:x from a contingent link, since contingent links do not share finishing points. Also, note that NO-CASE REDUCTION is just composition of ordinary edges. (The reason for including this will become clear below.) Finally, the LABEL REMOVAL REDUCTION is a transliteration of the *Unconditional Reduction* to the new notation.

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. We will define the *reduced distance* of a path in the labelled distance graph to be the sum of edge lengths in the path, ignoring any labels. Notice that the reductions preserve the reduced distance.

For each application of these reductions, we will refer to the pair of edges corresponding to the left side pattern as the *participants* in the reduction, and we will call the added edge the *result* of the reduction.

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

Our next change involves the special consistency test that is applied before each iteration in the classic algorithm. Instead of testing for the complex property of pseudo-controllability, we will check for ordinary consistency of the *AllMax* projection, where we define the AllMax projection to be the STN 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.

Note that it is correct to conclude that a network is not DC when the AllMax projection is inconsistent, since this excludes Weak Controllability, which in turn excludes Dynamic Controllability.

We remark that the $\mathrm{B} \neq \mathrm{C}$ restriction in the Cross-Case Reduction is crucial; otherwise, the upper-case and lower-case edges of any contingent link could self-interact, immediately producing an inconsistency.

Baseline Algorithm

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. We will call this the *baseline algorithm*. It follows from the previous discussion that the algorithm will give correct "no" answers. We now consider the opposite direction, and show that it will also still give correct "yes" answers.

Theorem 1 *If the baseline algorithm returns true then the network is dynamically controllable.*

Proof:

We will show the old reductions are either emulated by the new ones, or are unnecessary in the new framework. We also prove that it is unnecessary to directly test whether a contingent link is squeezed.

First, as noted previously, the two regressions are transliterations of the UPPER-CASE and CROSS-CASE REDUCTIONS as applied to waits, and the *Unconditional Reduction* is a transliteration of LABEL REMOVAL; thus, they are emulated. Furthermore, the *Unordered Reduction* is emulated by the UPPER-CASE REDUCTION as it applies to contingent links.

Next consider the Precedes Reduction

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

and suppose that its pattern is satisfied.

After applying both the UPPER-CASE and LOWER-CASE reductions to the labelled distance graph, we reach the following situation:

$$\mathrm{A} \overset{\mathrm{x'}}{\longrightarrow} \mathrm{C} \overset{\mathrm{B:-y'}}{\longrightarrow} \mathrm{A}$$

Then either $y' > x'$, in which case both algorithms detect inconsistency, or $y' \leq x' \leq x$, in which case the LABEL

REMOVAL REDUCTION applies. The result then emulates the *Precedes Reduction*.

Next we show that the checks for contingent-link squeezing that occur in pseudo-controllability testing are unnecessary. Suppose first an upper bound on a contingent link is squeezed, i.e., we have

$$\mathrm{A} \xleftarrow{\mathrm{B}:-\mathrm{y}} \mathrm{B} \xleftarrow{\mathrm{z}} \mathrm{A}$$

where $z < y$. Note that the UPPER-CASE REDUCTION is applicable, giving $\mathrm{A} \xleftarrow{\mathrm{B}:(-\mathrm{y}+\mathrm{z})} \mathrm{A}$, after which AllMax consistency testing detects a negative self-loop. Next consider where the lower bound is squeezed, i.e.,

$$\mathrm{A} \xrightarrow{\mathrm{b}:\mathrm{x}} \mathrm{B} \xrightarrow{\mathrm{z}} \mathrm{A}$$

where $z < -x$. Applying the LOWER-CASE REDUCTION gives $\mathrm{A} \xrightarrow{\mathrm{x}+\mathrm{z}} \mathrm{A}$, after which consistency testing detects a negative self-loop.

The other purpose fulfilled by pseudo-controllability testing was to compute the tight links of the All-Pairs graph. This task is now taken over by the NO-CASE REDUCTION. (Thus, the computation of tight links is interleaved with other reductions.)

It only remains to show that the *General Reduction* is unnecessary. An examination of the correctness proof in (Morris, Muscettola, & Vidal 2001) shows that this reduction is only needed to prevent deadlock, where a cycle exists in which each link has either a positive lower-bound or a positive wait. In the new framework, this task is fulfilled by the AllMax consistency testing, which would detect such a loop as an ordinary negative cycle. Thus, the classic algorithm correctness proof can be adapted to show correctness of the baseline algorithm, without the need for the General Reduction.

□

Cutoff Algorithm

Now that we have a mathematically simplified framework, we can proceed to improve on the baseline algorithm. We wish to obtain a cutoff bound analogous to that of Bellman-Ford, rather than relying on domain exhaustion for the termination guarantee. Relying only on domain exhaustion could cause cycles to be traversed a large number of times before detecting an inconsistency. Consider for example

$$\mathbf{A} \xrightarrow{-2} \mathrm{B} \xrightarrow{\mathrm{c}:1} \mathrm{C} \xrightarrow{-1} \mathrm{D} \xrightarrow{\mathrm{a}:1} \mathbf{A} \xrightarrow{-1} \mathrm{E}$$

with a cycle through A. Repeated applications of of Lower-Case and No-Case reductions derive $\mathrm{D} \xrightarrow{0} \mathrm{E}$, then $\mathrm{C} \xrightarrow{-1} \mathrm{E}$, then $\mathrm{B} \xrightarrow{0} \mathrm{E}$, and then $\mathrm{A} \xrightarrow{-2} \mathrm{E}$. Without a cutoff bound, the cycle could repeat to get $\mathrm{A} \xrightarrow{-3} \mathrm{E}$, and so on. If the domain allowed any C++ int value for AE, then such a cycle could potentially repeat 2^{15} times before the domain of AE would be exhausted.

Suppose an edge is tightened repeatedly as a result of the reductions, as happens to AE in the example above. When such a repetition occurs, then at least one of the edges participating in the reduction must itself have been tightened in the previous round of reductions. This leads us to define the *parent* of the result edge from a reduction to be the most recently tightened participating edge. (Ties are broken arbitrarily.)

Next we observe that if a chain of parents becomes longer than the size of the parent set, it must contain a repeated edge. We claim that if a parent chain has a repeated edge, then the network cannot be Dynamically Controllable. To see this, suppose an edge AB of length x is a repeated edge in some parent chain. This means there is an ancestor occurrence of AB in the parent chain where the length of AB is x' for some $x' > x$. Let $AB = e_0, \ldots, e_n = AB$ be the edges in the chain between the two occurrences of AB. It is not difficult to see that the edges that participate in reductions with the e_i will, in general, form loops on both sides of the AB edge, as illustrated here.[2]

$$\begin{array}{c} \mathrm{A} \xrightarrow{\mathrm{x}'} \mathrm{B} \\ \vdots \\ \mathrm{A} \cdots \mathrm{A} \xrightarrow{\mathrm{x}} \mathrm{B} \cdots \mathrm{B} \end{array}$$

One of the loops may be trivial (consisting of a single point), but since the reductions preserve reduced distance, and since $x' > x$, it follows that at least one of the loops is non-trivial and has a negative reduced distance. Note also that, because of the directionality of AB, Lower-Case reductions can only occur in the left-side loop, while Upper-Case reductions can only occur in the right-side loop. It follows that one of the loops corresponds to a negative cycle in either the AllMax or AllMin projections, so the network cannot be Dynamically Controllable. This establishes the claim.

Now we note that the edges added in the i-th iteration of the baseline algorithm must have parent chains that contain at least i parents. (Otherwise, they would have been added earlier.) Thus, we can use the size of the parent set as a cutoff bound analogous to Bellman-Ford, and terminate with false if that number of iterations is exceeded.

It remains to estimate the complexity of the algorithm given this cutoff. For this analysis, we assume N is the number of nodes, E is the number of edges, and K is the number of contingent links. Note that $K \leq N$ since contingent links do not share finishing points.

The algorithm can be summarized as follows.

```
Boolean procedure determineDC()
 loop from 1 to Cutoff Bound do
  if AllMax projection inconsistent
     return false;
  Perform needed No-Case Reductions;
  Perform needed Upper-Case Reductions;
  Perform any Cross-Case Reductions;
  Perform any Lower-Case Reductions;
  Perform any Label Removal Reductions;
  if no reductions were found above
     return true;
```

[2]The edges in the side loops may themselves be added during the intervening iterations, but will all exist by the time the repetition of AB occurs.

```
 end loop;
return false;
end procedure
```

We can analyze the complexity of each iteration as follows. For the No-Case tightenings we need only look at triangles of ordinary edges. Recall from the proof of Theorem 1 that we only need the No-Case Reduction to compute tight edges that are needed to support the other reductions. This means that we can restrict the No-Case Reduction so that it only applies when the result edge (and hence one of the participating edges) shares a node with either the start or the end timepoint of a contingent link. The No-Case phase of each iteration can thus be expressed as:

```
  for each ordinary edge e do
   for each "special" node B do
       Perform No-Case tightenings
         that involve e and B;
```

where "special" refers to the start or end of a contingent link. The complexity of this can be expressed as $O(E'K)$ where E' is the final number of edges in the networks.

We can also restrict the applicability of the Upper-Case Reduction. Consider

$$\text{A} \xleftarrow{\text{B:x}} \text{B} \longleftarrow \cdots \longleftarrow \text{C}$$

where the path from C to B consists of ordinary edges. Rather than applying the Upper-Case Reduction piecemeal to extend the upper case label backwards over each of the edges from C to B, we can first apply the No-Case reduction repeatedly to consolidate the CB path into a single edge, and then use a single Upper-Case Reduction to extend the upper case label all the way back to C. This means that we need only apply the Upper-Case Reduction in situations where the AB edge is either one of the original upper-case edges resulting from a contingent link, or one of the upper-case edges that results from a Cross-Case Reduction. In both cases, AB connects one special node to another special node, i.e., AB is chosen from a set of edges of size at most K^2. Thus, the complexity of the Upper-Case phase is $O(K^2N)$.

The participants in a Cross-Case Reduction involve a contingent link and another special node. Thus, the complexity of the Cross-Case phase is $O(K^2)$. To complete the picture, the Lower-Case and Label Removal phases have complexity bounds of $O(KN)$, since both involve considering a contingent link in relation to another node.

Reviewing the phases, it is easy to see that at most $O(NK)$ ordinary edges are ever added to the network during the operation of the algorithm, since each result link from a reduction has a special node as its start or end point. The number of upper-case edges is limited to NK because the reductions preserve the property that an upper-case edge always points to the source of its contingent link. (Thus, for each contingent link we can have at most N upper-case edges.) Also recall that the lower-case edges are fixed so there are exactly K of them. Thus, there are at most $O(NK)$ added edges. Note also that (from the second round on) the parent edges are taken from this set.

Thus, $O(E'K) = O(EK + NK^2)$ and, ignoring the consistency testing for now, we see that the No-Case phase dominates the complexity of each iteration. Note also that the parents in the repetition analysis, and hence the cutoff value, can be restricted to a set of size $O(NK)$. Thus, the total complexity can be estimated as $O(E'NK^2)$, which is equivalent to K^2 times the cost of a Bellman-Ford in an STN with E' edges. Finally, the AllMax consistency testing can be done using an incremental Bellman-Ford algorithm as in (Cervoni, Cesta, & Oddi 1994). The complexity, totalled over *all* the iterations, is $O(E'N)$, where E' is the final number of edges in the network. This fits well within the $O(E'NK^2)$ complexity estimate for the reductions.

Using our estimate above for $E'K$, we can expand the complexity as $O(ENK^2 + N^2K^3)$. In a dense graph, where $E \approx N^2$, this reduces to $O(N^3K^2)$; for a sparse graph where $E = O(N)$, we get $O(N^2K^3)$. If $K = O(N)$, we get an overall figure of $O(N^5)$.

In contrast to the pseudo-polynomial algorithm, these estimates do not involve a large "hidden" constant.

Conclusion

We have reformulated Dynamic Controllability testing in a way that provides mathematically simpler operations, and used that to obtain a strongly polynomial algorithm with a cutoff based on the structure of the network. Previously, only a pseudo-polynomial algorithm was known.

Acknowledgements We thank David Smith and anonymous referees for suggestions to improve the presentation.

References

Cervoni, R.; Cesta, A.; and Oddi, A. 1994. Managing dynamic temporal constraint networks. In *Proc. AIPS-94*, 13–20.

Cormen, T.; Leiserson, C.; and Rivest, R. 1990. *Introduction to Algorithms*. Cambridge, MA: MIT press.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proc. of IJCAI-01*.

Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote agent: to boldly go where no AI system has gone before. *Artificial Intelligence* 103(1-2):5–48.

Rossi, F.; Venable, K.; and Yorke-Smith, N. 2004. Controllability of soft temporal constraint problems. In *Proc. CP 2004*, 588–603.

Tsamardinos, I.; Pollack, M. E.; and Ramakrishnan, S. 2003. Assessing the probability of legal execution of plans with temporal uncertainty. In *ICAPS-03 Workshop on Planning under Uncertainty*.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *JETAI* 11:23–45.

Vidal, T. 2000. Controllability characterization and checking in contingent temporal constraint networks. In *Proc. of Seventh Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000)*.