Temporal Dynamic Controllability Revisited

Content Areas : Constraint Satisfaction, Temporal Reasoning, Uncertainty

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 greatly simplify the previous framework, and present a true polynomial algorithm with a cutoff based only on the number of nodes.

1 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 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.

Although we are not aware of deployed systems that fully confront these issues, progress has been made in recent years on the theoretical front. In [Vidal and Fargier, 1999], several notions of controllability are defined, including *Dynamic Controllability*. Roughly speaking, a network is dynamically controllable (DC) if there is a strategy for satisfying the requirements that depends only on knowing the outcomes of past uncontrollable events.

In [Morris *et al.*, 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. Note that 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. Looked at another way, if we allow the link sizes to grow, and measure the problem size in terms of the decimal representations of the link sizes, the complexity is $O(10^D N^3)$, where D is a bound on the lengths of the decimal representations of the link sizes. In the parlance of complexity theory, the algorithm is pseudo-polynomial like arc-consistency, rather than being a true polynomial algorithm. An example of the latter category is the well-known Bellman-Ford algorithm [Cormen *et al.*, 1990] for determining consistency of an STN.

Note that the constraint propagation process underlying Bellman-Ford can be viewed as enforcing arc-consistency. What makes the algorithm a true polynomial algorithm 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 *et al.*, 2001]. The treatment there involves numerous distinct concepts, including diverse reduction and regression operations, that are substantially simplified 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.

2 Background

We restate the basic definitions, as described in [Morris *et al.*, 2001], and summarize the algorithm presented there.

A Simple Temporal Network (STN) is a graph in which the edges are labelled with upper and lower numerical bounds. The nodes in the graph represent temporal events or *time-points*, while the edges correspond to constraints on the durations between the events. Formally, an STN may be described as a 4-tuple $< N, E, l, u >$ where N is a set of nodes, E is a set of edges, and $l : E \rightarrow \Re \cup \{-\infty\}$ and $u : E \rightarrow \Re \cup \{+\infty\}$

are functions mapping the edges into extended Real Numbers that are the lower and upper bounds of the interval of possible durations. Each STN is associated with a *distance graph* [Dechter *et al.*, 1991] 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, and this can be determined by a single-source shortest path propagation such as in the Bellman-Ford algorithm [Cormen *et al.*, 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, *contingent links* and *requirement links*. Contingent links may be thought of as representing causal processes of uncertain duration; 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.

Thus, an STNU is a 5-tuple $< N, E, l, u, C >$, where N, E, l, u are as in a STN, and C is a subset of the edges: the contingent links, the others being called requirement links. Each contingent link is required to satisfy $0 < l(e) < u(e) < \infty$. Multiple contingent links with the same finishing points are not allowed.

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 a fixed STNU $< N, E, l, u, C >$, a *schedule* T is a mapping

$$T : N \rightarrow \Re$$

where $T(x)$, written T_x here, is called the *time* of time-point x. A schedule is *consistent* if it satisfies all the link constraints. From a schedule, we can determine the durations of all contingent links that finish prior to a timepoint x. (This may be viewed as a partial mapping from C to $\Re$.) We call this the *prehistory* of x with respect to T, denoted by $T_{\prec x}$.

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 $S(p)$ is consistent (w.r.t. p) for each projection p.

We are now ready to define the various types of controllability, essentially 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 $p2$. Thus, a Strong execution strategy assigns a fixed time to each executable timepoint irrespective of the outcomes of the contingent links.

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.

It was shown in [Morris *et al.*, 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 graph. The algorithm also involved repeated tightenings of the network that, while preserving the status of DC or non-DC, made explicit the implicit constraints resulting from the contingent links. Thus, we can summarize the classic algorithm as follows.

```
Boolean procedure determineDC()
loop
  if not pseudo-controllable
     return false;
  if no more tightenings exist
     return true;
  else
     implement some tightening;
end loop;
end procedure
```

The guarantee of termination relied on the fact that if quiescence is not reached, then continued tightenings will eventually empty some domain.

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 either any positive amount of time has elapsed after B has occurred or until at least 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 that did play a role in that computation.

In order to describe the tightenings, we introduce the notation A $\overset{[x,y]}{\Longrightarrow}$ B to indicate a contingent link with bounds $[x, y]$ between A and B. It will also be convenient to write this backwards as B $\overset{[x,y]}{\Longleftarrow}$ A. We use the similar notation of A $\xrightarrow{[x,y]}$ B and B $\xleftarrow{[x,y]}$ A for ordinary links.

We can summarize the tightenings, called *reductions*, used in the classic algorithm as follows. They show new links that are added when the given pattern is satisfied unless tighter links already exist.

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

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

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

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

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

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

The “Unconditional” Reduction is so-called because in this situation the full impact of the ternary $<B, u>$ constraint is capturned 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.

3 Simplifications

As mentioned earlier, an ordinary STN has an alternative representation as a *distance graph*, in which a link A $\xrightarrow{[x,y]}$ B is replaced by two edges A $\xrightarrow{y}$ B and A $\xleftarrow{-x}$ 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 *et al.*, 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 $\xrightarrow{[x,y]}$ B is replaced by two edges A $\xrightarrow{y}$ B and A $\xleftarrow{-x}$ B, just as in an STN. For a contingent link A $\overset{[x,y]}{\Longrightarrow}$ B, we have the same two edges A $\xrightarrow{y}$ B and A $\xleftarrow{-x}$ B, but we also have two additional edges of the form A $\xrightarrow{b:x}$ B and A $\xleftarrow{B:-y}$ 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 $\xleftarrow{B:-y}$ B and A $\xrightarrow{b:x}$ B as *upper-case* and *lower-case* edges, respectively. Note that the upper-case label B:-y is 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 $\xrightarrow{<B,t>}$ C constraint in the labelled distance graph. This corresponds to a single edge A $\xleftarrow{B:-t}$ 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 simplified 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) If $y \geq 0$,
A $\xleftarrow{B:x}$ C $\xleftarrow{y}$ D adds A $\xleftarrow{B:(x+y)}$ D

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

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

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

As one might expect, if a reduction adds an edge where another edge with the same start, end and label already exists, then the tighter of the two edges is retained and the other discarded.

It is straightforward to see that the new reductions are sanctioned by the old ones. (We will defer for the moment the consideration of the reverse direction.) 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, the Upper-Case Reduction follows from the Unordered Reduction using [$-\infty$,y] as the bound. 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. Finally, note that No-Case Reduction is just composition of ordinary edges. (The reason for including this will become clear below.)

Observe that upper-case labels can "move" in the sense that they 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.

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

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

This is simply a transliteration of the Unconditional Reduction to the new notation.

Our next simplification 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. 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 if the AllMax projection is inconsistent, since this excludes Weak Controllability, which in turn excludes Dynamic Controllability.

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.

Next consider the Precedes Reduction

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

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:

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

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.

We note that the Unordered Reduction is directly emulated by the Upper-Case 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

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

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

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

where $z < -x$. Applying the Lower-Case Reduction gives $\mathrm{A} \xrightarrow{\mathrm{x+z}} \mathrm{A}$, 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 *et al.*, 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 correctness proof for the classic algorithm can be adapted to show correctness of the baseline algorithm, without the need for the General Reduction.

□

4 Cutoff Algorithm

Now that we have a simplified framework, we can proceed to improve on the baseline algorithm. Our first task is to obtain a cutoff bound analogous to that of Bellman-Ford, rather than depending for termination on the domain size, which could be large.

We will focus on the "Case" (including No-Case) reductions, because those are ones that can be applied repetitously. We regard such a reduction as a repetition if the particular node triple and configuration of labels (if any) has occurred before. For example, if the Lower-Case reduction is applied to

$$\mathrm{A} \xleftarrow{-1} \mathrm{B} \xleftarrow{\mathrm{b:10}} \mathrm{C}$$

and later (after BA has tightened to -2), it is applied to

$$\mathrm{A} \xleftarrow{-2} \mathrm{B} \xleftarrow{\mathrm{b:10}} \mathrm{C}$$

we consider that a repetition.

Note that if a repetition occurs, then at least one of the participating edges must have a tighter weight (or labelled weight) than its previous value. This leads us to define the *parent* of the result edge from a repetition to be the participating edge that tightened. If both participating edges have been tightened, one is chosen arbitrarily. Also, in a reduction that is occurring for the first time, the parent can be chosen arbitrarily.

Next we observe that during the operation of the baseline algorithm, the labelled distance graph (actually a multigraph, as noted earlier) has at most $N^2 + NK + K$ edges, where N is the number of nodes and K is the number of contingent timepoints. Here N^2 is a bound on the number of ordinary edges. 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.

The bound means that if a parent chain becomes longer than $N^2 + NK + K$, then it must contain a repeated edge. Note the "Case" reductions have a linearity property where a tightening of the parent is transmitted to the child without any attenuation. Moreover, a reduction that is applicable will still be applicable if a parent is tightened further. This implies that any repetition in the parent chain will lead to continued repetitions until some domain is exhausted. Thus, the network is not DC in this situation.

Next 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. We conclude that we can use N^2+NK+K as a cutoff analogous to Bellman-Ford, and terminate with false if that number of iterations is exceeded.

This leads to the following algorithm for Dynamic Controllability, which we term the *basic-cutoff algorithm*.

```
Boolean procedure determineDC()
loop for iter from 1 to (N^2 + NK + K)
  if AllMax projection inconsistent
     return false;
  Perform any No-Case tightenings;
  for each Upper-Case edge e do
    for each node B do
       Perform Upper-Case tightenings
         that involve e and B;
       Perform Cross-Case tightenings
         that involve e and B;
  for each Lower-Case edge e do
    for each node B do
       Perform Lower-Case tightenings
         that involve e and B;
  if no tightenings were found above
     return true;
  for each Upper-Case edge e do
     Perform applicable Label Removals;
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. This phase has complexity $O(N^3)$. The Upper-Case phase has complexity $O(KN^2)$ since there are at most KN upper-case edges. The Lower-Case phase clearly has complexity $O(KN)$, and the Label Removals also have complexity $O(KN)$. Since the $O(KN^2)$ and $O(KN)$ terms are dominated by $O(N^3)$, the overall complexity of each iteration, excluding AllMax consistency testing, is $O(N^3)$. The total complexity is then $O(N^5)$, since there are at most $N^2 + NK + K$ iterations. The AllMax consistency testing can be done using an incremental Bellman-Ford algorithm as in [Cervoni *et al.*, 1994]. The amortized complexity is $O(E'N)$, where E' is the final number of edges. Note that E' is bounded by $(N^2 + KN + K)$. Thus, this is dominated by the $O(N^5)$ complexity derived above.

5 Refined Algorithm

We can do better if we can either reduce the scope of the inner loop, or reduce the size of the set from which the parents are taken (which would reduce the cutoff bound). As it turns out, we can do both by exploiting the following observation. Notice that the AllMax consistency testing takes full account of upper-case edge values (with the labels removed), but ignores lower-case edges. Thus, if we can "eliminate the need" for the lower-case edges, the consistency testing takes care of the remaining interactions.

To make this more precise, define the *reduced distance* of any path in the labelled distance graph to be that obtained by summing the numerical values on the edges without regard to any labels. Notice that new paths created by reductions preserve the reduced distance. The reductions (in particular the lower-case and cross-case ones) may be viewed as converting reduced distances to distances in the AllMax projection by providing an alternative path that is free of lower-case edges. This suggests that we might be able to restrict the reductions so that they only add edges that are needed to support the creation of the alternative paths.

We will show this can be achieved by restricting the Upper-Case and No-Case Reductions so that they only apply when D is a contingent timepoint, i.e., the target of a contingent link. With these restrictions, applications of the two reductions behave essentially as a propagation forward from each contingent timepoint. This reduces the scope of all the reductions to only adding edges that emanate from the start or end of a contingent link. (The Lower-Case and Cross-Case Reductions already satisfy this restriction.) We have the following theorem.

Theorem 2 *Any effect on AllMax distances of the unrestricted reductions can also be achieved by the restricted reductions.*

Proof:

Using the unrestricted reductions, consider any application α of a Cross-Case or Lower-Case Reduction (the only ones that can tighten AllMax distances) during the course of the algorithm. Suppose the reduction is applied to A $\longleftarrow$ C $\xleftarrow{\text{c:y}}$ D where CA has a negative (possibly upper-labelled) value.

If we "unwind" all the reductions preceding α back to the start, we can identify a path in the original labelled distance graph that is the preimage or *precursor* of the CA edge. (Since the lower-case edges are fixed, the precursor of the DC edge is itself.)

We will focus on the propagation forward from C to A supported by the restricted reductions. We need to show that this will allow the DC lower-case edge to be bypassed on some subpath of the DA path. Notice that if the propagation is not "blocked" and reaches A then the α application can again be applied to achieve this.

Consider first the special case where there are no lower-case edges between C and A. Then repetitions of the No-Case Reduction will allow propagation forward from C until we reach some upper-case edge B'A'. At that point, we have the following situation:

$$\mathrm{A}' \xleftarrow{\mathrm{B}':\mathrm{x}} \mathrm{B}' \xleftarrow{\mathrm{y}} \mathrm{C} \xleftarrow{\mathrm{c:z}} \mathrm{D}$$

Suppose u is the lower bound of the A'B' contingent link. Note that if $x + y \geq -u$, then we can apply an upper-case reduction followed by label removal to eliminate the B' label from the path and continue. If $x + y < -u$ and B'$\neq$C, we can apply an upper-case followed by a cross-case reduction to immediately bypass DC. Otherwise, since the unrestricted propagations were able to eliminate B', there must be some node E between C and B' such that $x + EB' \geq -u$ and hence CE must be negative. In this case, we can apply a lower-case reduction immediately to bypass DC. Thus, in all cases we can convert the reduced distance on the DA path to an AllMax distance.

Next consider the general case where there may be lower-case edges between C and B. These must have been bypassed in the unrestricted case by applications of the lower-case or cross-case reductions. We will call the node that participates as the "A" node in those applications the *annihilator* of the lower-case edge. It is easy to see that in the precursor path, the pairs consisting of a lower-case edge and its annihilator must be "bracketed" in the sense that two such pairs must either be nested or disjoint. Then the innermost pair of each nested group will consist of a lower-case edge and its annihilator without any intermediate lower-case edges. Thus, we can apply the method for the special case above repeatedly in an "inside-out" fashion to eliminate all the lower-case edges from the path.

□

The restricted application of tightenings allows us to modify the no-case and upper-case phases of the basic-cutoff algorithm as follows.

```
Boolean procedure determineDC()
loop for iter from 1 to (N^2 + NK + K)
  if AllMax projection inconsistent
     return false;
! Perform restricted No-Case tightenings;
  for each Upper-Case edge e do
!   for each contingent node B do
       Perform Upper-Case tightenings
         that involve e and B;
       Perform Cross-Case tightenings
         that involve e and B;
  for each Lower-Case edge e do
    for each node B do
       Perform Lower-Case tightenings
         that involve e and B;
  if no tightenings were found above
     return true;
  for each Upper-Case edge e do
     Perform applicable Label Removals;
end loop;
return false;
end procedure
```

We call the modified algorithm the *refined-cutoff* algorithm for Dynamic Controllability.

Note that this reduces the complexity of the no-case phase to $O(E'K)$, where $E' = E + NK$ is a bound on the final number of edges in the network. The upper-case phase now has complexity $O(NK^2)$. The dominant term is $O(E'K)$, which also dominates the other phases in the iteration. Moreover, all of the parents in the parent chain analysis are now chosen from 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.

6 Closing Remarks

Other recent work has focused on combining DC reasoning with preferences [Rossi *et al.*, 2004] or probabilities [Tsamardinos *et al.*, 2003]. In this paper, we have revisited the foundation to place it on a simpler and more efficient footing.

References

[Cervoni *et al.*, 1994] R. Cervoni, A. Cesta, and A. Oddi. Managing dynamic temporal constraint networks. In *Proc. AIPS-94*, pages 13–20, 1994.

[Cormen *et al.*, 1990] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT press, Cambridge, MA, 1990.

[Dechter *et al.*, 1991] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, May 1991.

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

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

[Rossi *et al.*, 2004] F. Rossi, K.B. Venable, and N. Yorke-Smith. Controllability of soft temporal constraint problems. In *Proc. CP 2004*, pages 588–603, 2004.

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

[Vidal and Fargier, 1999] T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11:23–45, 1999.

[Vidal, 2000] T. Vidal. 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.