Execution of Temporal Plans with Uncertainty

Paul Morris[*]
Caelum Research Corporation
M/S 269-1 NASA Ames Research Center
Moffett Field, CA 94035
pmorris@ptolemy.arc.nasa.gov

Nicola Muscettola
Mail Stop 269-2
NASA Ames Research Center
Moffett Field, CA 94035
mus@ptolemy.arc.nasa.gov

Abstract

Simple Temporal Networks (STNs) have proved useful in applications that involve metric time. However, many applications involve events whose timing is uncertain in the sense that it is not controlled by the execution agent. In this paper we consider execution algorithms for temporal networks that include events of uncertain timing. We present two such algorithms. The first retains maximum flexibility, but requires potentially costly updates during execution. The second surrenders some flexibility in order to obtain a fast execution comparable to that available for ordinary STNs.

Introduction

Simple Temporal Networks (Dechter, Meiri, & Pearl 1991) have proved useful in Planning and Scheduling applications that involve quantitative reasoning about time (e.g. (Bienkowski & Hoebel 1998; Muscettola *et al.* 1998)) because they allow fast checking of temporal consistency after each plan step. However this formalism does not adequately address an important aspect of real execution domains: the occurrence time of some events may not be under the complete control of the execution agent. For example, when a spacecraft commands an instrument or interrogates a sensor, a varying amount of time may intervene before the operation is completed. In cases like this, the execution agent does not have freedom to select the precise time delay between events in accord with the timing of previously executed events. Instead, the value is selected by Nature independently of the agent's choices. This can lead to constraint violations during execution even if the Simple Temporal Network appeared consistent at plan generation time. The problem of control of temporal networks with uncertainty was first addressed formally in (Vidal & Ghallab 1996; Vidal & Fargier 1997), and was also studied in (Morris & Muscettola 1999).

The previous work has been primarily concerned with algorithms to determine various flavors of *controllability* of temporal networks with uncertainty. Controllability is a property analogous to consistency in ordinary (without uncertainty) temporal networks. This paper focuses instead on execution algorithms for temporal networks with uncertainty and considers issues of flexibility and efficiency.

A method in widespread use for executing ordinary temporal networks is to "harden" the network. That is, a specific solution is chosen that rigidly fixes the time of execution of all the events in the network. Note that this approach is inapplicable to networks with uncertainty because of the unpredictable choices made by Nature (which may not agree with the chosen solution), and is often undesirable even for ordinary temporal networks because it allows no flexibility for dealing with unmodelled contingencies that may nevertheless occur in practice. However, retaining flexibility into execution imposes a burden of propagation to ensure that the windows of later timepoints are appropriately narrowed as earlier timepoints are executed. Since a short cycle time is paramount during execution, methods have been developed (Muscettola, Morris, & Tsamardinos 1998; Tsamardinos, Muscettola, & Morris 1998) to reformulate the network in order to reduce the amount of propagation. A central issue addressed in this paper is to what extent these methods are applicable to networks with uncertainty.

Background

We review the definitions of Simple Temporal Network (Dechter, Meiri, & Pearl 1991), Minimum Dispatchable Network (Muscettola, Morris, & Tsamardinos 1998), and Simple Temporal Network with Uncertainty (Vidal & Fargier 1997).

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 *timepoints*, 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 $l : E \rightarrow \mathbb{R} \cup \{-\infty\}$ and $u : E \rightarrow \mathbb{R} \cup \{+\infty\}$ are functions mapping the edges into extended Real Numbers. Figure 1 shows an example of an STN. Figure 2 shows the corresponding *distance graph* (Dechter, Meiri, & Pearl 1991), which is an alternate representation useful for mathematical analysis. Note that lower bounds are negated to give the lengths of the reverse edges. Edges of infinite length are omitted. (Given a distance graph, one can also find a corresponding STN, so the representations are interchangeable.) An STN is consistent

[*] Current Affiliation: RIACS.

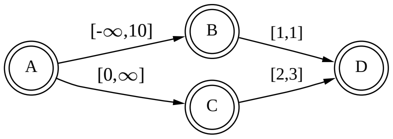

Figure 1: Simple Temporal Network.

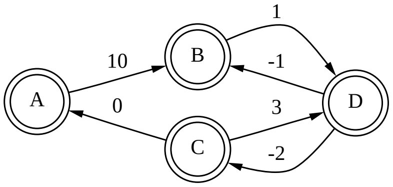

Figure 2: Distance Graph.

if and only if the distance graph does not contain a negative cycle.

In (Dechter, Meiri, & Pearl 1991), it is shown how time windows (upper and lower time bounds) can be determined for each timepoint by propagations in the distance graph starting from any designated initial timepoint. The upper bound corresponds to the shortest-path distance from the initial timepoint to the given timepoint, while the lower bound is the negation of the shortest-path distance in the opposite direction. Shortest-path distances are efficiently determined by propagation methods such as the Bellman-Ford algorithm (Cormen, Leiserson, & Rivest 1990). A solution (i.e., a globally consistent assignment of time values to timepoints) can be incrementally constructed without backtracking by progressively extending a locally consistent assignment, with propagation of time windows occurring after each extension.

An execution algorithm is essentially the same as an incremental construction of a solution. However, there are two important distinctions, as discussed in (Muscettola, Morris, & Tsamardinos 1998). First, to be eligible for execution, a timepoint must be *live*, i.e., the current time must lie between its upper and lower bounds. (Note that we maximize flexibility by delaying fixing the time of a timepoint until it is actually executed.) Second, suppose the shortest-path distance (in the distance graph) from a timepoint A to a timepoint B is negative. It follows that in all valid solutions, the time assigned to B must precede that assigned to A. We say B is an *enabling condition* for A. Notice that a naive incremental execution algorithm may execute A before B, with the resulting propagation forcing B into the past. To correct this, it is necessary to determine all the enabling conditions ahead of time, and defer executing timepoints until they are enabled. (Note that a deadlock cannot occur since a consistent STN has no negative cycles.)

The modified execution algorithm is still costly because it requires a full propagation after each timepoint is executed. A better approach (Muscettola, Morris, & Tsamardinos 1998) is to convert the network to an equivalent *minimum dispatchable* network for which *local propagation* (to immediate neighbors only) is sufficent to ensure a valid execution. Conversion to a minimum dispatchable network is obtained by first computing the All-Pairs Shortest-Path graph (Cormen, Leiserson, & Rivest 1990) (henceforth we abbreviate this to the All-Pairs graph), and then eliminating *dominated* edges. An edge AB is dominated if there is a timepoint C such that $AC + CB = AB$ and either both AB and CB are non-negative or both AB and AC are negative. In the former case, AB is dominated by CB; in the latter, by AC. If an edge is dominated, then removing it from the All-Pairs graph does not change the set of valid executions.

Controllability problems (Vidal & Ghallab 1996; Vidal & Fargier 1997) arise in STNs where the edges, which we will call *links*, are divided into two classes, *contingent links* and *requirement links*. Contingent links may be thought of as representing causal processes; their finishing times are controlled by Nature, subject to the limits imposed by the bounds on the contingent links. All other timepoints, which we will call *control points*, are controlled by the agent, whose goal is to satisfy the bounds on the requirement links.

Thus, a *Simple Temporal Network with Uncertainty* (Vidal & Fargier 1997) (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 edges in C are called the *contingent links*, and the other edges are the *requirement links*. We require $0 < l(e) < u(e) < \infty$ for each contingent link e.

Each set of allowable durations for the contingent links may be thought of as reducing the STNU to an ordinary STN. Thus, an STNU determines a family of STNs, as in the following definition.

Suppose $\Gamma = < N, E, l, u, C >$ is an STNU. A *projection* (Vidal & Ghallab 1996) of Γ is a Simple Temporal Network derived from Γ where each requirement link is replaced by an identical STN link, and each contingent link e is replaced by an STN link with equal upper and lower bounds $[b, b]$ for some b such that $l(e) \leq b \leq u(e)$.

A *boundary* projection (Vidal & Fargier 1997) is one where, for each contingent link e, the above b is chosen so that $b = l(e)$ or $b = u(e)$. Because of convexity, the minimum shortest path distances over all projections can be calculated using only the boundary projections. This makes it feasible for many algorithms to essentially ignore the non-boundary projections.

Execution Of STNU

Execution of an STNU is complicated by the necessity of updating the network as new information is received. The most common scenario is that the precise timing of a contingent link finishing point is observed when the contingent process finishes. This is analogous to the timing of a control point becoming fixed when it is selected for execution. We call this the *standard* update scenario. However, other update scenarios are possible. For example, in a spacecraft application, the predicted time of closest encounter to a comet may be progressively refined as the event approaches. On the other hand, the precise time of the encounter might not be available until some time after the event has occurred due to delays involving communication or analysis.

In this paper we will consider two approaches to executing STNUs. The first, considered in this section, is a general

```
1.   A = Set of all control points
     current_time = 0
2. Arbitrarily pick a control point TP in A that is
   live and enabled in all boundary projections.
3. Set TP execution time to current_time, and remove
   TP from A.  Halt if A is now empty.
4. Advance current time, propagating all updates
   until some point in A is live and enabled.
5. Go to 2.
```

Figure 3: Execution Algorithm.

method that preserves maximum flexibility, but entails significant computational cost. The second approach, considered in a later section, surrenders some flexibility in order to achieve greater efficiency.

In (Morris & Muscettola 1999), the concept of *Waypoint Controllability* is introduced as a practical approach to ensuring controllability. It is shown that a Waypoint Controllable network may be effectively decomposed into (1) an induced STN involving only the waypoints, and (2) several STNUs corresponding to the subnetworks *between* the waypoints. This decomposition may impact the choice of execution algorithm. The point to note is that the subnetworks can be expected to be quite limited in size, which makes it feasible to consider methods with asymptotic high complexity. This lends plausibility to the algorithm to be considered in this section, which is exponential in the number of interacting contingent links.

A simple execution algorithm could proceed as if the network were an ordinary STN and simply fail if propagation causes the flexibility of a contingent link to be restricted. The following algorithm does better by excluding current values for control points that would lead to future restrictions on the uncontrollables.

We can formulate an execution algorithm in a manner analogous to that for ordinary STNs. For this, we must (1) propagate time windows, (2) determine whether a timepoint is live and enabled, and (3) repropagate with updated information. In the STNU case, updated information may result from observations and refined predictions, as well as executions. Note also that an update may arise from simple passage of time, where a contingent link finishing point is observed to have not yet occurred. Figure 3 summarizes the overall algorithm.

Basic Propagation

The propagation in the execution algorithm is closely related to the propagation algorithm for determining Waypoint Controllability that was introduced in (Morris & Muscettola 1999). The latter is similar to a Bellman-Ford distance propagation in an ordinary STN, except it effectively propagates in all boundary projections simultaneously. This is accomplished by splitting each distance value propagated across a contingent link into two values corresponding to the different boundary projections. For example, a contingent link $A \rightarrow B$ with bounds $[1, 3]$ corresponds to an STN link with bounds $[1, 1]$ or $[3, 3]$ in different boundary projections. This means a distance value of 0 at A produces propagated values of 1 or 3 at B. The algorithm carries forward both values, but tags them according to which selection was used. In this case, we could write the set of propagated tagged values as $\{1^t, 3^T\}$, where t and T denote the $[1, 1]$ and $[3, 3]$ selections, respectively. Note that the values may split again at subsequent contingent links, leading to values involving multiple choices. Thus, the propagated values acquire tags reflecting all the boundary choices involved in their creation.

We will use the expression v^T to denote a value v with tag T. (We may write v, without superscript, if the tag is empty.) A tagged value $v_1^{T_1}$ *subsumes* a tagged value $v_2^{T_2}$ if $v_1 \leq v_2$ and $T_1 \subseteq T_2$. Subsumed values are subject to deletion. Observe that we can propagate distance values just like in an ordinary STN; the only difference is that sets of tagged values are propagated to each node instead of simple values. It is convenient to use a simplified notation for sets of tagged values, writing $1^t 3^T$ instead of $\{1^t, 3^T\}$.

Windows and Updates

For execution purposes, we need to propagate both upper and lower bounds, i.e., a time window. Each bound consists of a set of labelled values, corresponding to the values in the different boundary projections. Consider, for example, a contingent link $A \rightarrow B$ with bounds $[1, 3]$. A time window of $[0, 4]$ at A then propagates to a time window of $[1^t 3^T, 5^t 7^T]$ at B, where t and T indicate the $[1, 1]$ and $[3, 3]$ choices, respectively. The reader should pay particular attention to the interval $[3, 5]$ lying between the maximum of the lower-bound values and the minimum of the upper-bound values, which corresponds to the intersection of the windows from all the boundary projections.

When an update occurs, we need to repropagate. Suppose, in the above example, the lower bound for the contingent link is increased from 1 to 2. This affects both the lower and upper bounds at B. The update of the lower bound at B is an increase from $1^t 3^T$ to $2^t 3^T$, which is a potential tightening of the intersection window. The change to the upper bound at B is also an increase, from $5^t 7^T$ to $6^t 7^T$. However, an increase in the upper bound is a potential loosening of the intersection window, not a tightening.

The need to propagate predictive updates efficiently requires a high degree of incrementality. In ordinary STNs, it is difficult to achieve incremental propagation of loosened bounds. However, the tags provide sufficient information to identify the values that need revision. A reasonable degree of incrementality is obtained by simply deleting values with tags involving the updated links, and repropagating from those links. (More selective deletion is possible by including additional information in the tags. For example, each contingent link gives rise to both a positive and negative edge in the distance graph, and information about which of those participated in a propagation could be used to advantage. We omit the details.)

Liveness and Enablement

We next consider how to select a control timepoint for execution. First, the timepoint must be live in all boundary projections. This is equivalent to the current time lying between the maximum of the lower-bound values and the minimum of the upper-bound ones. As an example, consider the network

```
    [1,100]      [-1,50]
 A =======> B <------- C
```

where A ==> B is a contingent link, B <-- C is a requirement link, and A is the start point. The initial window at C is given by $[-49^{-t}50^{-T}, 2^{+t}101^{+T}]$, so C is not live. Suppose B is observed to occur at time 25. A new propagation produces a window of $[-25, 26]$ for C , which makes it live. On the other hand, suppose it is observed that B has *not* occurred by time 50. The bounds for the contingent link can be narrowed to $[50, 100]$. This update propagates to give $[0^{-t}50^{-T}, 51^{+t}101^{+T}]$ for C. Notice that C is now live in all projections even though B has not finished yet.

The second condition for execution is that the timepoint be enabled in all boundary projections. This poses a minor complication compared to an ordinary STN execution because the set of projections may contract as updates occur. We can still compute in advance all the paths that could possibly be required for enablement. (Note that each path will have a set of tagged values as its distance.) However, as projections are excluded due to updates, some of the paths may cease to be enabling. This requires repeated updating and checking of the enablement distances to see if they are still negative in some projection.

It is easy to see that the algorithm will never choose an execution time that is inconsistent with some projection. However, an execution can still fail, even for a network where all projections are consistent. This can happen if a situation occurs where immediate execution and deferred execution each exclude some of the projections. An example of this is the impossible task where we need to make preparations tightly in advance before some event whose timing is unknown.

Consider, for example, the network

```
    [1,100]      [1,50]
 A =======> B <------ C
```

and assume the standard update scenario. Note that C must occur before B. However, any time we choose for C before time 50 will be inconsistent with some late occurrence of B. On the other hand, deferring the choice until time 50 will fail if B occurs early. For instance, the updated window for C at time 25, if B has not occurred earlier, will be $[-25^{-t}50^{-T}, 24^{+t}99^{+T}]$. Note that C is not live, so it cannot be executed. If B now occurs, the execution has failed.

Dispatchability

We next consider the extent to which the methods of (Muscettola, Morris, & Tsamardinos 1998) can speed up the execution algorithm, in the framework where maximum flexibility is retained.

The transformation to the All-Pairs graph is straightforward. However, the distances must be computed using tagged values. This can be costly, but is done offline before the execution begins.

A complication arises in the removal of dominated edges. In this framework, an edge may only be removed if it is dominated in *all* projections. This presents fewer opportunities for removal, and seems to exclude the possibility of arriving at a *minimum* dispatchable network. If interacting contingent edges are few in practice, as seems likely with the waypoint controllability approach, there should nevertheless be substantial savings from using a pruned dispatchable network.

A second complication is with respect to the arbitrary updates envisaged in this framework. The proof of theorem 1 in (Muscettola, Morris, & Tsamardinos 1998), upon which the dominance analysis is based, relies on two assumptions about the type of update that occurs during execution: first, the update only occurs for timepoints that are enabled; second, the update narrows the window to a single value. The upshot of this is that local propagation will be in general be insufficient for predictive updates to contingent links. Full propagation can be used instead. On the positive side, the updates arising from executions and sharp observations (that collapse the interval to a single point) do satisfy the criteria, and for those, local propagation is sufficient.

Safe Networks

We now turn to an approach that sacrifices some flexibility in order to avoid the exponential complexity of the algorithm in the previous section. This applies to scenarios where the precise finishing time of a contingent link becomes known as soon as it occurs. The key idea is as follows. Suppose we pretend that an STNU is an ordinary STN, and the contingent links are just ordinary links. Suppose also that the network has the property that, for every valid execution and for each contingent link, the only effective propagations to the contingent link finishing point are those that propagate through the contingent link itself. In this case, the propagations do not affect the possible durations of the contingent links, and Nature's flexibility is not impaired. Consequently, we can pretend that the agent is making all the choices, and carry out the execution as if the network was an ordinary STN. In particular, simple values can be propagated; no tags are needed. Furthermore, we can apply the methods of (Muscettola, Morris, & Tsamardinos 1998) to compute a minimum dispatchable network, and use only local propagation during execution.

An STNU with the above felicitous property will be called a *safe* network. Our first task is to obtain a characterization of the property that is more easily checked. It turns out that the dominance relation introduced in (Muscettola, Morris, & Tsamardinos 1998) is relevant here. The characterization will be in terms of the STN obtained from a given STNU by ignoring the distinction between contingent links and requirement links. We call this the *associated STN* of the given STNU. Note that in the distance graph of the associated STN, the forward edge arising from a contingent link will have positive length, while the corresponding reverse edge will have negative length. These edges will be referred to as "contingent edges."

The following terminology will also be useful. In an STN distance graph, an edge from A to B is a *tight edge* if there is no path from A to B whose distance is less than the length of the edge.

To avoid certain technical difficulties, it is necessary to exclude networks where two contingent links have the same finishing point. This does not entail any loss of generality because the coincident finishing points can be separated and connected by a $[0, 0]$ requirement link instead. (Semantically, there really are two points that are independently controlled by Nature.)

This leads to the following characterization.

Theorem 1 (Safety Theorem) *An STNU is safe if and only if in the distance graph of the associated STN the following three properties hold:*

(1) The forward and reverse edges arising from each contingent link are tight edges. (Thus, they survive unchanged in the All-Pairs graph.)

(2) In the All-Pairs graph, the forward edge arising from each contingent link $A \to B$ dominates all non-negative edges whose destination is B.

(3) In the All-Pairs graph, the reverse edge (from B to A) arising from each contingent link $A \to B$ dominates all negative edges whose source is B.

Proof: First suppose (1) is violated. Then one of the edges arising from a contingent link $A \to B$ is squeezed. Consider what happens when A is executed. The propagation along the squeezing path will supersede the propagation along the squeezed contingent edge. Thus, the network is not safe.

It remains to show that if (1) holds, then (2) and (3) together are equivalent to safety. This is a straightforward consequence of the methods of (Muscettola, Morris, & Tsamardinos 1998). Since (1) holds, we need only consider dispatching executions in the All-Pairs graph. Let $A \to B$ be a contingent link. By Theorem 1 of (Muscettola, Morris, & Tsamardinos 1998), the only effective propagations to B are those involving non-negative edges whose destination is B and negative edges whose source is B. Condition (2) states that the only effective non-negative edge propagation is the one occurring along the forward contingent edge. Similarly condition (3) states that the only effective negative edge propagation is the one occurring along the reverse contingent edge. Thus, (2) and (3) together state that the only effective propagations to B are those occurring along the two contingent edges, which is the requirement for safety. □

It follows from the theorem that for a safe STNU, the associated STN can be converted into a minimum dispatchable network in which the contingent links survive.[1] Moreover, for a contingent link $A \to B$ in this minimum dispatchable network, the forward contingent edge will be the only non-negative edge remaining whose destination is B, and the reverse contingent edge will be the only negative edge remaining whose source is B. This makes safety easy to check by first converting to the minimum dispatchable network. With minor modifications, the algorithm for constructing a minimum dispatchable graph in (Tsamardinos, Muscettola, & Morris 1998) can thus be used to verify safety in $O(EN + N^2 \log(N))$ time.

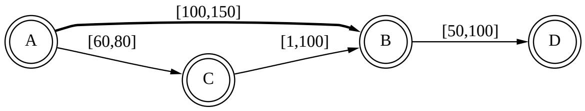

Figure 4: Safe Network.

As an example of a safe network, consider figure 4, where $A \to B$ is a contingent link. Notice that no matter what time C is executed (within its window), the duration of $A \to B$ will not be further restricted. Also D does not affect the duration since it is constrained to occur after B. It is easy to verify, using the Triangle Rule of (Muscettola, Morris, & Tsamardinos 1998), that the dominance relations required by the theorem hold in the All-Pairs graph. Note that a minimum dispatchable graph exists that is identical to the original network except the link $C \to B$ is deleted.

Predictive updates that narrow the temporal bounds of contingent links can be accommodated within this framework. It is not hard to see that these cannot make a safe network unsafe. However, each such update requires full rather than local propagation, since it does not satisfy the criteria of dispatching executions.

Potential Safety

Few networks encountered in practice are likely to be safe. However, many networks that are not initially safe can be made so by judicious tightening of the constraints.

Consider for example the network

```
   [1,100]      [-50,50]
A =======> B <---------
|                     |
|                     |
|       [0,150]       |
 -------------------> C
```

which is not safe since if C is executed at time 10, B will be forced to occur before time 60.

Notice that the network is already in All-Pairs form. We can use the criteria of the Safety Theorem to determine how to make it safe. The only edge at issue is the one of length 50 from C to B arising from the C --> B link. This violates condition (2). There are two ways to remove this violation.

First, we could make the edge be dominated by tightening up the bounds of the A --> C link as dictated by the Triangle Rule of (Muscettola, Morris, & Tsamardinos 1998). This requires that the distance graph edges satisfy $CA + AB = CB$, which can be accomplished by tightening the A --> C bounds to $[50, 150]$ (increasing the lower bound). It is easy to see the network is now safe: if B occurs before C, there is no problem (since B then propagates to C rather than vice versa); otherwise the propagation from C to B merely ratifies the situation that B has not yet occurred. For example, if B has not occurred before C is executed at time 60, then the propagation requires B to occur

[1] If contingent links with coincident finishing points were allowed, this would not necessarily be the case because then there could be mutually dominating contingent links.

within $[10, 110]$, which is not restrictive given the current time of 60.

The second way to remove the violation is to eliminate the non-negative status of the edge. This can be done by tightening the C --> B bounds to $[-50, -1]$ (reducing the upper bound). Note that the $[-50, -1]$ link is equivalent to a $[1, 50]$ link in the opposite direction. This forces C to occur after B, which also makes the network safe.

With both methods, we must take care to ensure the tightening does not squeeze the contingent link, or produce an inconsistency. For example, if A --> C had bounds $[0, 100]$ then the second method would not work since it would restrict the contingent link to $[1, 99]$.

It is easy to see that this leads to a general algorithm for making a network safe, or determining that this is not possible. Each violation of the Safety Theorem can potentially be repaired in two ways. After a repair is attempted, the All-Pairs graph must be recomputed to determine whether a contingent link has been squeezed, or an inconsistency has been introduced. If so, the repair fails.

A network for which the repair algorithm succeeds will be called *potentially safe*. Since each choice of possible repairs can be verified in polynomial time, we see that the problem of determining potential safety is in $\mathcal{NP}$.

Examples suggest that the property of being potentially safe is very similar to that of *Dynamic Controllability*, as defined in (Vidal & Fargier 1997). The precise relationship between them remains to be determined. Note that an $\mathcal{NP}$ algorithm for Dynamic Controllability has not been presented in the literature.

Observe that in making safe a potentially safe network, we are giving up flexibility in advance. This complicates the question of predictive updates of contingent links. We could accommodate them, as discussed previously, by doing a full propagation, but this would not take advantage of the flexibility-enhancing effect of narrowing a contingent link. To do that, we would need to redo the repairs required for safety, which might be too costly in practice during execution.

Unfortunately, not all networks that can be effectively executed can be made safe. Consider the network

```
  [1,200]     [-1,100]
A =======> B <---------
|                      |
|                      |
|      [0,100]         |
 --------------------> C
```

The following execution algorithm is effective:

```
if B occurs before time 99
then execute C at 1 unit later
else execute C at the fixed time 100
```

The first possibility for making this safe would be to increase the A --> C lower bound to 100 in an attempt to satisfy the Triangle Rule. However, this would squeeze the contingent link, forcing an increase in the lower bound to 99. The other possibility is to decrease the C --> B upper bound to -1, so that C occurs after B. However, that also squeezes the contingent link, forcing a decrease in the upper bound to 99.

This raises the possibility of using the safety analysis to synthesize the conditional execution strategy noted above. For example, the given network could be specialized to two "cases" where the contingent link bounds are restricted to ranges of $[1, 99]$ and $[99, 200]$, respectively. Each of these can be made safe separately, leading to the two branches of the strategy. This is a topic for future work.

Acknowledgments

We thank Thierry Vidal for useful discussions, and also thank the referees for their suggestions.

References

Bienkowski, M. A., and Hoebel, L. J. 1998. Integrating AI components for a military planning application. In *Proc. of Fifteenth Nat. Conf. on Artificial Intelligence (AAAI-98)*.

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., and Muscettola, N. 1999. Managing temporal uncertainty through waypoint controllability. In *Proc. of Sixthteenth Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*.

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.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proc. of Sixth Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*.

Tsamardinos, I.; Muscettola, N.; and Morris, P. 1998. Fast transformation of temporal plans for efficient execution. In *Proc. of Fifteenth Nat. Conf. on Artificial Intelligence (AAAI-98)*.

Vidal, T., and Fargier, H. 1997. Contingent durations in temporal CSPs: From consistency to controllabilities. In *Proc. of IEEE TIME-97 International Workshop*.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proc. of 12th European Conference on Artificial Intelligence (ECAI-96)*, 48–52.