International Journal on Artificial Intelligence Tools
Vol. 16, No. 1 (2007) 17–44
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

OPTIMIZATION OF BOUNDS IN TEMPORAL FLEXIBLE PLANS WITH DYNAMIC CONTROLLABILITY*

BENJAMIN W. WAH

Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
wah@uiuc.edu
http://manip.crhc.uiuc.edu

DONG XIN

Department of Computer Science
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
dongxin@uiuc.edu

Received 29 August 2005
Accepted 11 August 2006

A temporal flexible planning problem that involves contingent and requirement events can be formulated as a simple temporal network with uncertainty (*STNU*). An *STNU* is controllable when there is a strategy for executing the requirement events (or actions) in such a way that all the conditions involving contingent events can be satisfied in all situations. The most interesting and useful controllability property is dynamic controllability in which the remaining actions in an *STNU* can always be scheduled under all possible feasible durations of future contingent events when all the past contingent events are known. In this paper, we propose and study a novel problem of assigning bounds on the duration of each requirement link in order for the resulting *STNU* to be dynamically controllable and to minimize the total cost over the allowed durations of all requirement links. We first prove the NP hardness of the problem with a linear cost function. We then formulate the dynamic controllability of an *STNU* as the constraints in a nonlinear optimization problem. Finally, we present methods for reducing the number of constraints in order to make the problem tractable and to demonstrate the computational performance of our methods.

Keywords: Constraints; dynamic controllability; nonlinear programming; NP-hardness; simple temporal network with uncertainty.

*Research supported by National Science Foundation Grant IIS 03-12084.
An early version of this paper appeared in the *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, 2004. This paper has been substantially revised to include new proofs and experimental results.

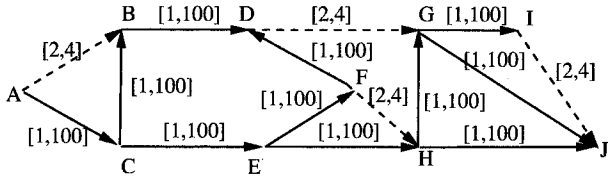
1. Introduction

A temporal flexible planning problem can be formulated in a simple temporal network with uncertainty[12] (*STNU*) whose links can be classified into *contingent* and *requirement links*, and the duration of each link is modeled by a lower bound and an upper bound. A contingent link represents a causal process/action of uncertain duration whose completion time is controlled by nature, whereas a requirement link represents a process/action whose completion time is controlled by a planner. For example, the arrival time of Bob riding a bus is a contingent event because it depends on the traffic condition. If Bob chooses to walk to the destination, then his arrival time is a requirement event because it is under his control.

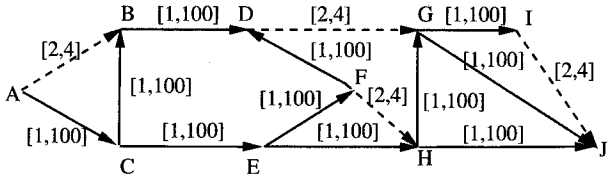
Formally, an *STNU* defines the partial order of events in an application and is described by a 5-tuple $\Gamma = \langle V, E, L, U, C \rangle$. Here, V is the set of nodes that represent labels in the horizon; E is the set of links that represent events of some finite duration; L: $E \rightarrow \mathbb{R} \cup \{-\infty\}$ and U: $E \rightarrow \mathbb{R} \cup \{+\infty\}$ are functions that map a link into two real numbers that represent, respectively, the lower and upper bounds on the duration that the corresponding action will take; C is the subset of links that are contingent links; and $E \backslash C$ is the subset of links that are requirement links.

The problem of constraint satisfaction for an *STNU* has been characterized by *controllability*.[11,10] A network is controllable if there is a strategy for executing each requirement event (or action) in such a way that all the conditions involving contingent events are satisfied in all situations. In *strong controllability*, the actions can always be scheduled under all possible times at which contingent events can happen. In *weak controllability*, the actions can always be scheduled under all possible times of contingent events if those times were specified ahead of time. Last, in *dynamic controllability*, the remaining actions in a network can always be scheduled under all possible feasible durations of future contingent events when all the past contingent events are known. It is easy to see from these definitions that strong controllability implies dynamic controllability, and that dynamic controllability implies weak controllability. That is, if an *STNU* is strongly (*resp.* dynamically) controllable, then it must be dynamically (*resp.*, weakly) controllable. Since strong controllability is too rigid for planning under contingent events and weak controllability does not guarantee a strategy that can handle all contingencies, the most interesting and useful controllability property is dynamic controllability.

In this paper, we propose and study a new optimization problem of assigning bounds on requirement links in order for the resulting *STNU* to be dynamically controllable and the total cost over the allowed durations of all requirement links to be minimized. Define a dynamically controllable *STNU* $\Gamma = \langle V, E, L, U, C \rangle$, where $[L(e), U(e)]$ are the loose bounds of $e \in E$; and $[\ell(e), u(e)]$ are the desired bounds of e. Assuming a cost function f: $\ell(e_1), u(e_1), \ldots, \ell(e_n), u(e_n) \rightarrow \mathbb{R}$, where $n = |E|$, the goal is to find $\Gamma' = \langle V, E, \ell, u, C \rangle$, the optimal *STNU* with respect to



(a) Original dynamically controllable *STNU* with loose bounds



(b) Dynamically controllable *STNU* with optimized bounds

Fig. 1. An example of a dynamically controllable *STNU* with optimized bounds that minimizes a linear cost function on the interval in each requirement link. A solid arrow represents a requirement link, whereas a dashed arrow represents a contingent link.

f over $\ell(e_i)$, $u(e_i), e_i \in E$, that solves the following:

$$
\begin{aligned}
(P_{stnu}): \quad & \min \ f(\ell(e_1), u(e_1), \ldots, \ell(e_n), u(e_n)) & \\
& \text{subject to } L(e) \leq \ell(e) \leq u(e) \leq U(e) & e \in E \backslash C \\
& \qquad \ell(e) = L(e), \quad u(e) = U(e) & e \in C \\
& \text{and } \Gamma' \text{ is dynamically controllable.} & (1)
\end{aligned}
$$

P_{stnu} is interesting in practice. A larger interval on a requirement link always makes the resulting *STNU* more likely to be dynamically controllable. However, such flexibility may incur additional costs on the application because some resources may have to be made available earlier. For instance, the cost of allowing a camera onboard a satellite to be turned on at any time in the interval [10,20] may be higher than that if the camera were allowed to be turned on in [18,20]. Hence, it is imperative to reduce the interval as much as possible as long as the resulting *STNU* is dynamically controllable.

Figure 1a illustrates an instance of P_{stnu}. A dashed arrow between nodes i and j represents a contingent event with labels i and j in the horizon and whose duration is between a lower bound L and an upper bound U. The duration of a contingent event is controlled by nature, and the plan executer does not know its value until the contingent event finishes. On the other hand, a solid arrow between nodes i and j represents a requirement event whose duration can be controlled by the plan executer, given that it is within the lower and upper bounds.

For example, the completion of requirement event AC in Figure 1a triggers requirement events CB and CE. On the other hand, requirement event BD is triggered

by the simultaneous completion of contingent event AB and requirement event CB. Suppose AB has a duration of 3 time units, which is within the interval [2,4]. This means that AC must have a duration of 1 or 2 time units and that CB must have a corresponding duration of 2 or 1 time units in order for CB to complete at time 3. It should be clear that it is not possible to have two contingent events incident on a node because it is not possible to synchronize their occurrences. Moreover, given the maximum duration of contingent event AB, it is clear that the upper bounds on the duration of requirement events AC and CB are excessive.

By following the definition on dynamic controllability, it is easy to verify that the network in Figure 1a is dynamically controllable. Assuming a linear cost function of $\sum_{e \in E \setminus C}(u(e) - \ell(e))$, the *STNU* has an objective value of 1,089. By reducing the bounds on each requirement link, Figure 1b shows a modified *STNU* that is dynamically controllable yet has an objective value of 18.

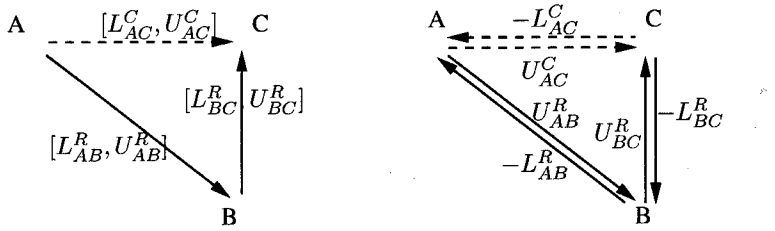
Previous studies on *STNU*s have focused on algorithms for checking controllability[7,5,10] and for successfully scheduling the tasks of an *STNU*.[6,9] Section 2 reviews existing algorithms for checking dynamic controllability. Although such algorithms have polynomial complexity,[7] we prove in Section 3 that P_{stnu} is NP hard, even when a linear cost function is involved. In this paper, we formulate P_{stnu} as a constrained optimization problem and solve it by existing nonlinear programming methods. This approach is flexible because it can incorporate additional constraints, such as resource constraints[8] and general cost functions, in the formulation. Since existing methods for checking dynamic controllability are procedural,[7] we first define in Section 4 the constraints that specify the conditions for dynamic controllability in our constrained formulation. We show a naïve formulation that leads to a problem with $\mathcal{O}(N^2)$ variables and $\mathcal{O}(N^3)$ constraints for an N-node *STNU*, which is intractable when N is large. To reduce the complexity, we propose in Section 5 new methods for eliminating unnecessary variables and implied constraints. Finally, Section 6 presents our experimental results.

2. Dynamic Controllability

Given an *STNU* with independent contingent events, an algorithm for checking dynamic controllability must consider every combination of possible contingent events.[7] We assume that the lower bounds on the duration of all contingent links are positive because the influence of a contingent event should only propagate forward in time.[5] We first review some key concepts[7] before formulating the requirements as constraints. The dynamic controllability of an *STNU* is classified into local and global dynamic controllability.

2.1. *Local dynamic controllability*

By treating an N-node *STNU* as C_3^N triangles, the local dynamic controllability of the *STNU* can be established by examining each triangle in two steps.[7]



(a) Triangle ABC (b) The corresponding distance graph

Fig. 2. A triangle in an *STNU* and its corresponding distance graph

The first step treats each contingent link as a requirement link and examines one at a time each triangle in the network and its associated *distance graph* (Figure 2).[4] A directed link in the triangle corresponds to two directed edges in the distance graph, whose weights are derived from the upper and negative lower bounds of the corresponding links. To allow an action of a requirement link in the triangle to be schedulable, the bounds of the corresponding link in the distance graph must be the shortest paths. For instance, in Figure 2, it follows that $[L^R_{BC}, U^R_{BC}] \subseteq [L^C_{AC} - U^R_{AB}, U^C_{AC} - L^R_{AB}]$, where $[L^R_r, U^R_r]$ (*resp.*, $[L^C_c, U^C_c]$) denote the lower and upper bounds of requirement link r (*resp.*, contingent link c).

Second, a triangle with at least one contingent link is classified into one of the following three categories. (If a triangle has two contingent links, then it will be considered twice, with each contingent link in turn playing the role of a requirement link.) Refer to Figure 2a in the following discussion.

a) If $U^R_{BC} < 0$, then B must follow C in its occurrence, and ABC is in the *follow case*. In this case, contingent link AC acts like a requirement link because C has already occurred at the time when B is scheduled to occur. The condition for dynamic controllability is always satisfied for this triangle.

b) If $L^R_{BC} \geq 0$ and $U^R_{BC} > 0$, then B must occur before or simultaneously with C, and ABC is in the *precede case*. Since the information about the occurrence of C is not available to B when B is scheduled, the bounds of AB must be tightened to $[U^C_{AC} - U^R_{BC}, L^C_{AC} - L^R_{BC}]$ in order for B to be controllable.

c) If $L^R_{BC} < 0$ and $U^R_{BC} \geq 0$, then B can occur before or after C, and ABC is in the *unordered case*. If C has not occurred when B is considered for scheduling, then B cannot be scheduled at any time before $U^C_{AC} - U^R_{BC}$ after A has occurred. Further, if $U^C_{AC} - U^R_{BC} \leq L^C_{AC}$, then L^R_{AB} can be tightened to $U^C_{AC} - U^R_{BC}$. On the other hand, if $U^C_{AC} - U^R_{BC} > L^C_{AC}$, then L^R_{AB} can be tightened to L^C_{AC}, and a wait annotation $\langle C, w_{ABC} \rangle$ is placed on AB for contingent link AC. We call w_{ABC} a *triangular wait* in this paper, where:

$$w_{ABC} = U^C_{AC} - U^R_{BC} \quad \text{(triangular wait on AB for contingent link AC).} \tag{2}$$

Moreover, w_{ABC}, L^R_{AB}, and L^C_{AC} are related by the following property:

$$L^R_{AB} \geq \min(L^C_{AC}, w_{ABC}). \tag{3}$$

Intuitively, w_{ABC} defines a threshold: at any time before the threshold, B cannot occur until C has occurred; whereas at any time after the threshold, B can occur independent of C. It is obvious that the wait on a link must be within the lower and upper bounds defined on the link. The only exception is when the link is a contingent link. In this case, the wait on the contingent link must be the same as its defined lower bound. For simplicity, we refer to the lower and upper bounds on the wait defined in (2) as the *lower-bound* and *upper-bound wait constraints*, or collectively, the *wait-bound constraints*.

2.2. *Global dynamic controllability*

In global dynamic controllability, wait information is propagated throughout an *STNU* by *regression*.[7] In wait regression, a wait that regresses to other links is called a *source wait*, whereas a wait regressed from a source wait is called a *target wait*. Note that a regression wait can be regressed from multiple source waits.

Consider regressing $\langle C, w_{ABC}\rangle$, a wait on AB, to AD, where AC is the contingent link causing the wait.

a) If DB is a requirement link with upper bound U^R_{DB}, or DB is a contingent link and $w_{ABC} < 0$, then the wait regressed to AD is $\langle C, w_{ABC} - U^R_{DB}\rangle$. That is,

$$w_{ADC} = w_{ABC} - U^R_{DB} \tag{4}$$

for requirement link DB, or contingent link DB with $w_{ABC} < 0$.

b) If DB is a contingent link with lower bound L^C_{DB} and $w_{ABC} \geq 0$, then the wait regressed to AD is $\langle C, w_{ABC} - L^C_{DB}\rangle$. This is,

$$w_{ADC} = w_{ABC} - L^C_{DB} \tag{5}$$

for contingent link DB with $w_{ABC} \geq 0$.

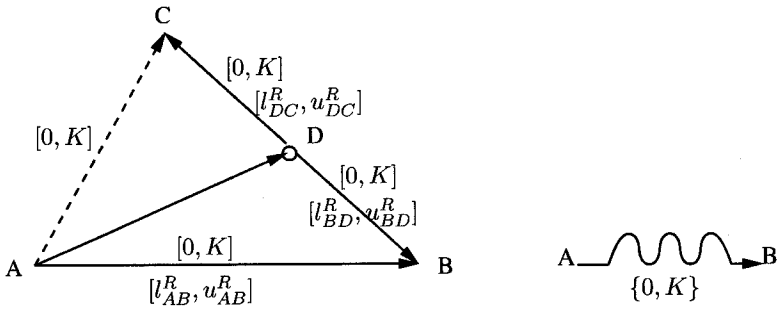
To distinguish the wait regressed to AD from a *triangular wait*, we call it a *regression wait* in this paper. The actual wait on AD will be the maximum of its regression and triangular waits.

After tightening the bounds of each link, an *STNU* is dynamically controllable if and only if $U^R_r \geq L^R_r$ for every requirement link r and $[L^C_c, U^C_c]$ has not been tightened for every contingent link c.[7]

3. NP-Hardness of P_{stnu}

In this section, we show that P_{stnu} with a linear cost function is NP-hard. The proof is done by constructing a mapping from the 3-coloring[3] of graph G to an instance of P_{stnu} represented by S_c. We map the nodes of G to those of S_c in such a way that the color of a node in G or the difference in colors between two nodes in G is represented as the bounds on a corresponding link in S_c.

To map from a discrete color value in G to two continuous values that represent the lower and upper bounds of a link in S_c, we introduce a basic *STNU* with a linear cost function. The goal is to construct S_c using this basic *STNU* as a building block



(a) Basic *STNU* (b) Notation of the basic *STNU*

Fig. 3. Basic STNU with bounds of $[0,0]$ or $[K,K]$ on AB when it is dynamically controllable.

in such a way that the possible bounds of a link in S_c are either $[0,0]$ or $[K,K]$ when S_c is dynamically controllable and its cost is minimized.

Figure 3a shows the basic *STNU*, where AC is a contingent link with upper bound K, and AB, DB, DC, and AD are requirement links. We further denote $[0,K]$ to be the original loose bounds, and $[\ell^R, u^R]$ to be the desired tightened bounds when the goal is to minimize the following linear cost function:

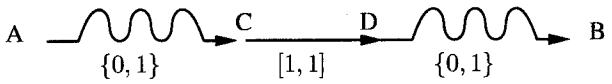
$$f = 2 \cdot (u^R_{AB} - \ell^R_{AB}) + (u^R_{DC} - \ell^R_{DC}) + (u^R_{DB} - \ell^R_{DB}) + \ell^R_{DB} - K. \tag{6}$$

Consider ADC. Since the lower bound of DC is nonnegative, ADC is in the precede case, according to the classification in local dynamic controllability. Hence, the desired bounds of DC must be either $[0,K]$ if C and D are different nodes, or $[0,0]$ if C and D are the same nodes. If the desired bounds on DC is $[0,K]$, then the desired bounds on AD, AB, and DB must be $[0,0]$ in order to minimize f in (6). On the other hand, if the desired bounds on DC is $[0,0]$, then the sum of the durations of DB and AB has to be no less than K in order to make the network dynamically controllable. To minimize (6), we have to choose the bounds of DB to be $[0,K]$ and, consequently, the bounds on AB to be $[K,K]$. In short, in order to achieve the minimum cost in (6) to be 0 and to have the basic *STNU* dynamically controllable, the bounds on AB must be either $[0,0]$ or $[K,K]$. Figure 3b shows a shorthand notation of this basic *STNU*.

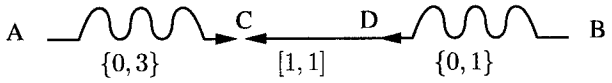
The basic *STNU*s can be compounded into more complex structures, with a cost equal to the sum of the costs of the basic *STNU*s. Figure 4a shows a compound *STNU* whose bounds are $[1,1]$, $[2,2]$, or $[3,3]$. It is made up of three basic *STNU*s whose bounds are either $[0,0]$ or $[1,1]$. Figure 4b shows another compound *STNU* whose bounds are $[-2,-2]$, $[-1,-1]$, $[1,1]$, or $[2,2]$.

Based on the compound *STNU*s in Figure 4, we can transform in polynomial time the 3-coloring of a graph G to the determination of the bounds on the links of S_c in such a way that solve P_{stnu} and that achieves a minimum cost of zero.

Theorem 3.1. The problem of P_{stnu} with a linear cost function is NP-hard.



(a) Compound *STNU* with bounds [1,1], [2,2], or [3,3] on AB



(b) Compound *STNU* with bounds [−2,−2], [−1,−1], [1,1], or [2,2] on AB

Fig. 4. Compound *STNU*s that are dynamically controllable.

Proof. We first present a polynomial procedure to transform an arbitrary graph $G = (V_G, E_G)$ to an *STNU* S_c. Given G, we create S_c as follows. We first add a source node s to S_c. For each $v_i \in V_G$, we add v_i to S_c and use a compound *STNU* in Figure 4a to connect s and v_i in S_c. Similarly, for each $(v_i, v_j) \in E_G$, we use the compound *STNU* in Figure 4b to connect v_i and v_j in S_c. The transformation takes $O(|V_G| + |E_G|)$ time. The cost function of P_{stnu} over S_c is the sum of the costs of the corresponding compound *STNU*s.

To complete the proof, we show that the 3-coloring of G can be reduced to solving P_{stnu} with a linear cost function over S_c. The proof is done in two parts.

First, we prove that, if G is 3-colorable, then S_c with a linear cost function is dynamically controllable and achieves a minimum cost of zero. This happens when each of the compound *STNU*s in S_c is dynamically controllable and achieves a minimum cost of zero. In that case, the color of v_i in G corresponds to the bounds $[1,1]$, $[2,2]$, or $[3,3]$ of (s, v_i) in S_c, and the difference in colors assigned to v_i and v_j in G corresponds to the bounds $[-2,-2]$, $[-1,-1]$, $[1,1]$, or $[2,2]$ of (v_i, v_j) in S_c.

Next, we prove that, if S_c with a linear cost function is dynamically controllable and achieves a minimum cost of zero, then G is 3-colorable. The assumption implies that all the compound *STNU*s in S_c are dynamically controllable and has a zero cost. Each node v_i in S_c corresponds to a bound $[1,1]$, $[2,2]$, or $[3,3]$, and each edge in S_c between v_i and v_j corresponds to the bounds $[-2,-2]$, $[-1,-1]$, $[1,1]$, or $[2,2]$. Since S_c is dynamically controllable, these bounds constitute a consistent assignment of S_c. Consequently, G is 3-colorable and the color on v_i of G corresponds to the bounds $[1,1]$, $[2,2]$, or $[3,3]$.

The two parts of the proof reduce the 3-coloring of G to solving P_{stnu} with a linear cost function over S_c. Since the 3-coloring of G is NP-hard, it implies that solving P_{stnu} is also NP-hard. □

Theorem 3.1 shows that, although checking dynamic controllability has polynomial complexity, P_{stnu} is NP-Hard even when a linear cost function is used. To solve P_{stnu}, we formulate in the next section the dynamic controllability of an *STNU* as constraints and solve P_{stnu} as a nonlinear constrained optimization problem.

4. Naive Formulation

To formulate the conditions for dynamic controllability of an *STNU*, we can simply develop a constraint for each property in Section 2. Appendix A presents the complete list of constraints that must be satisfied. We group these constraints into the *shortest-path*, *precede*, and *wait* constraints. The latter includes constraints on triangular-wait, regression-wait, and wait-bound. In our formulation, we represent the desired bounds on a link as $[\ell, u]$, and the wait value on AB for contingent link AC as variable w_{ABC} or simply w.

The first two sets of constraints are needed for any simple temporal network without contingent links. The bound constraints in (9) specify that the desired bounds of a requirement link must be within its loose bounds, and those of a contingent link are fixed. The shortest-path constraints in (10) are applied on every triangle in the *STNU*.[4]

Next, we follow the discussions in Section 2 in order to develop the constraints for handling dynamic controllability. Since the condition for dynamic controllability in the follow case is always satisfied, no constraints are needed. The constraints for the precede (*resp.*, triangular-wait) case are given in (11) (*resp.*, (12)). We do not include the constraints for the unordered case because these constraints are implied. We formally state this property as follows and leave the proof to Appendix B.

Lemma 4.1. *If the shortest-path and the precede constraints are satisfied, then removing the conditions for the unordered case will not change the space of feasible solutions of* P_{stnu}.

The constraints for wait regression (or global dynamic controllability) are formulated in (13). Finally, (14)–(16) show the constraints for the various cases of the wait-bounds constraints where the duration of a wait cannot exceed the lower and upper bounds on the corresponding link.

In contrast to the algorithm for checking dynamic controllability in which the bounds and the type of each triangle in S_c are known a priori, the bounds in P_{stnu} are variables. Hence, the conditions for the different cases of a constraint that involve variables must be expanded into new constraints in the formulation. Because the steps involved for each class of constraints are similar, we only illustrate the construction of the constraints in (13) for regressing a wait through a contingent link with bounds $[\ell^C, u^C]$.

Assume wait w_2 obtained by regressing wait w_1 through a contingent link. Then:

$$w_2 \geq \begin{cases} w_1 - \ell^C & \text{if } w_1 \geq 0 \\ w_1 - u^C & \text{otherwise.} \end{cases} \tag{7}$$

To represent (7) as constraints without requiring a condition that depends on w_1 (a variable in the constraint), we generate a linear constraint that is true for both

Table 1. Complexity of the naïve formulation for an N-node $STNU$ with C contingent links.

Type of Const.	# of Const.	Type of Var.	# of Var.
Bound	$\mathcal{O}(N^2)$	Bound	$\mathcal{O}(N^2)$
Shortest-Path	$\mathcal{O}(N^3)$	Wait	$\mathcal{O}(CN)$
Precede	$\mathcal{O}(CN)$	Auxiliary	$\mathcal{O}(C^2+CN)$
Triangular-Wait	$\mathcal{O}(CN)$		
Regression-Wait	$\mathcal{O}(CN^2)$		
Wait-Bound	$\mathcal{O}(CN)$		

cases and introduce three new constraints using an auxiliary variable α:

$$\begin{aligned} w_2 &\geq w_1 - u^C; \\ \alpha &\geq 0; \quad \alpha \geq w_1; \\ \alpha \cdot (w_2 - w_1 + \ell^C) &\geq 0. \end{aligned} \tag{8}$$

In addition, we add $\alpha(\alpha - w_1)$ to the objective in order to ensure that the α chosen is either w_1 when $w_1 \geq 0$ or 0 when $w_1 < 0$. A similar approach can be applied to convert (16) in Appendix A.

The two constraints defined in (7) with respect to the sign of w_1 are not always needed when we can infer the sign of w_1. For example, since the wait on a link must be within its initial bounds $[L_1, U_1]$, we know that $w_1 \geq 0$ if $L_1 \geq 0$, and that $w_1 < 0$ if $U_1 < 0$.

After specifying the constraints, we can now develop a naïve constrained formulation for solving P_{stnu} of $STNU$ S_c in two steps. First, we generate the precede and wait constraints for each contingent link in S_c and add the new links involved in the constrained formulation to S_c. Next, we consider every link in the updated S_c as a requirement link and formulate new constraints to ensure every bound to be the shortest path in the corresponding distance graph.

Table 1 shows that the naïve formulation leads to $\mathcal{O}(N^2)$ variables and $\mathcal{O}(N^3)$ constraints for an N-node $STNU$. As demonstrated in Section 6, such a formulation is usually too large to be solved when N is large. In the next section, we present methods for reducing the number of redundant variables and implied constraints.

5. Reduced Formulation

Because our naïve formulation treats an $STNU$ as fully connected and enumerates all its possible triangles, it leads to many redundant variables and implied constraints. However, these redundancies are usually too complex to be reduced by existing pre-solving and linear-reduction techniques.[1]

In this section, we examine those redundancies on the wait, precede, and shortest-path constraints in our naïve formulation and introduce methods to eliminate them. Instead of looking for redundancies directly, we avoid generating implied constraints by analyzing the relations among the constraints. In contrast to our naïve formulation that adds new links in $STNU$ S_c in an unrestricted fashion, our

reduced formulation only adds essential links when formulating the conditions for dynamical controllability. These links are used to propagate existing bound constraints throughout S_c. They are divided into two classes: those involved in the wait and precede constraints and those involved in the shortest-path constraints. Because the shortest-path constraints are weaker than the wait and precede constraints, they are formulated on the updated network after the links on the wait and precede constraints have been added. Note that the amount of reductions is problem dependent. In the worst case, if S_c is fully connected, then all the shortest-path constraints are necessary and cannot be reduced.

5.1. *Reductions on wait constraints*

These reductions are based on eliminating redundant wait constraints and those implied by the shortest-path and triangular-wait constraints. Before we show the reductions, we describe four observations.

First, recall that a wait on a link is the maximum of its triangular and regression waits, and that the bounds on the wait are defined by its wait-bound constraints. A wait constraint is redundant and does not have to be explicitly included when one of the following conditions is true.

- A triangular (*resp.* regression) wait is redundant when it is guaranteed to be less than the regression (*resp.* triangular) wait on the same link.
- A wait on a link is redundant when it satisfies the wait-bound constraint on the link, and its regressed wait on other target links does not introduce tighter bounds on those target links.

Hence, the wait-bound constraints on a link are redundant when both the associated triangular and regression waits on the link are redundant.

Second, the wait-bound constraints of wait w on a requirement link r may be simplified by examining their relation to $[L^R, U^R]$, the loose bounds on r. As is indicated in (3), $\ell^R \geq \min(\ell^C, w)$, where ℓ^R is the desired lower bound of r and ℓ^C is the lower bound of contingent link c causing wait w on r. There are two special cases in which the wait-bound constraints on r can be simplified.

- The lower-bound wait constraint is redundant when $L^R \geq \ell^C$. The constraint is implied because $\ell^R \geq L^R \geq \ell^C \geq \min(\ell^C, w)$.
- If $\ell^C \geq U^R$, then we conclude from (15) that $\ell^C \geq w$. Hence, from (3), we conclude that $\ell^R \geq w$. Since $\ell^R \leq w$ according to (10) and (12), we can simplify the wait-bound constraint in this case to $w = \ell^R$.

Third, when the wait on a requirement link satisfies (15) and when the shortest-path constraints in (10) are enforced, the regression of this wait through a requirement link or the regression of a negative wait through a contingent link (both defined in (4)) also satisfies (15). For instance, if wait w_{ABD} satisfies (15), namely, $w_{ABD} \leq u^R_{AB}$, and if $u^R_{AB} \leq u^R_{AC} + u^R_{CB}$ is satisfied due to (10) on ABC, then

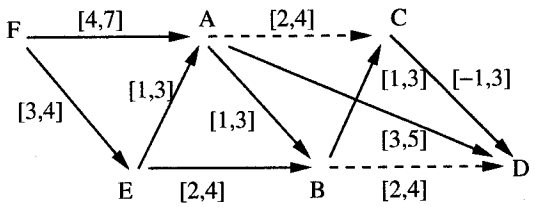


Fig. 5. Reductions on wait constraints.

the regression of w_{ABD} on AB to w_{ACD} on requirement link AC also satisfies (15), namely, $w_{ACD} = w_{ABD} - u^R_{CB} \leq u^R_{AB} - u^R_{CB} \leq u^R_{AC}$. With the redundant constraints identified above, since a triangular wait is always no larger than the upper bound of the link where the wait applies, (15) is only needed on waits that are positive and that can be regressed by a contingent link.

Fourth, assume that A′B′ in S_c corresponds to AB in the original distance graph. If A′B′ is eliminated, then the shortest-path constraints ensure that the upper bound of A′B′ must equal to the minimum of the upper bounds of all possible paths from A′ to B′, where the upper bound of a path is the sum of the upper bounds of all the links along the path. Similarly, the lower bound of A′B′ must equal to the maximum of the lower bounds of all possible paths from A′ to B′. Note that the path corresponding to the upper bound of A′B′ and that corresponding to the lower bound of A′B′ may not be identical. This observation allows us to formulate the wait-bound constraints with only nearby nodes and to propagate their effect throughout S_c by triangles with shared links.

Based on these four observations, we now present our reductions on redundant wait constraints. We first check S_c for dynamic controllability and derive better bounds on all possible links, including those that do not exist in S_c, in our naïve formulation. Next, we derive the wait (triangular-wait, regression-wait, and wait-bound) constraints on every link in S_c. To find the redundant constraints, we first classify each node B for contingent link AC and link AB into three subsets, where AB can be a requirement or a contingent link:

$$\begin{cases} \{B \text{ such that } U_{AB} \leq L^C_{AC}\} & \text{(pre-wait set)} \\ \{B \text{ such that } L_{AB} \geq L^C_{AC}\} & \text{(post-wait set)} \\ \text{otherwise} & \text{(unordered-wait set).} \end{cases}$$

As an illustration, for contingent link AC in Figure 5, its pre-wait, post-wait, and unordered-wait sets are, respectively, {E,F}, {D}, and {B}.

Our classification allows us to ignore the wait constraints on nodes in the pre-wait (*resp.* post-wait) set because the wait on a link between A and any node B in the pre-wait (*resp.* post-wait) set is no larger (*resp.* no less) than L^C_{AC}. That is, these nodes are guaranteed to occur before (*resp.* after) some time point. This is true because $L^C_{AC} = \ell^C_{AC}$ for contingent link AC, and the nodes defined in the pre-wait and post-wait sets match exactly those defined in the second observation

above. Hence, the lower-bound wait constraint between A outside the post-wait set and any node B in the post-wait set is redundant. Similarly, the wait-bound constraints between A outside the pre-wait set and any node B in the pre-wait set can be simplified to $\ell^R_{AB} = w_{ABC}$, where ℓ^R_{AB} is the lower bound of requirement link AB and w_{ABC} is its wait.

Although our initial classification leads to a small unordered-wait set, it is incomplete in the sense that some nodes in the pre-wait (*resp.* post-wait) set may not form triangles with contingent link c in the precede (*resp.* follow) cases and whose triangular-wait constraints are necessary (see Section 2.1). According to the proof of Lemma 4.1, we know that the triangular-wait constraints in both the precede and follow cases are redundant. Hence, our approach is to first find nodes that form triangles with contingent link c in the precede (*resp.* follow) case in the pre-wait (*resp.* post-wait) set and whose triangular-wait constraints are redundant. We then identify those remaining nodes in the pre-wait (*resp.* post-wait) set whose triangular-wait constraints are necessary and migrate them to the unordered-wait set. The migration procedure is presented in that related to guard migrations (*resp.* pre-migrations) later in this subsection.

On the other hand, regression-wait constraints in the pre-wait set and post-wait set are more complex for two reasons. First, if waits in the pre-wait and post-wait sets are found by regression, then the bounds on which these waits apply may be tightened. Second, if waits in the pre-wait and post-wait sets are regressed to other links, then the bounds on the target links may also be tightened. In both cases, the corresponding regression-wait constraints may be redundant. Assuming that triangular waits have been simplified according to the second and third observations stated above, the effect of regression waits on bounds only depends on waits that are positive and that can be regressed through a contingent link (that is, the end point of a link where the wait applies is the end point of a contingent link). Figure 6 summarizes the post-migration and pre-migration procedures presented below.

a) *Post-migrations* are used to find nodes in the post-wait set whose waits may affect the bounds of links. For each F in the post-wait set, let F be the end point of contingent link EF. We first consider link CF, where AC is a contingent link. If $L^R_{CF} < 0$ (*i.e.*, AFC is not in the follow case), then the triangular wait on AF and its further regression through EF might affect the bounds on AE. In this case, we will migrate both F and E to the unordered-wait set (Line 3 in Figure 6). Next, let P be a node in the unordered-wait set that is the starting point of a contingent link. If $L^R_{PF} \geq 0$, then the regression of w_{APC} to w_{AFC} and further to w_{AEC} is equivalent to the regression of w_{APC} to w_{AEC} through requirement link PE (this statement is implied by the precede constraints because triangle FPE, with FE a contingent link, is in the precede case). Hence, we only migrate E and F to the unordered-wait set if $L^R_{PF} < 0$ (Line 5), and the regression of w_{APC} to AF and that of w_{AFC} through EF will be included in the formulation. This step is repeated until the unordered-wait set does not change. As an illustration, node D in Figure 5 will be migrated to the unordered-wait set, since D is the end point of contingent link BD and $L^R_{CD} < 0$.

```
1.  procedure wait-reduction(AC) // AC is a contingent link //
2.    repeat
3.      for each F in the post-wait set where F is the end point of a cont. link EF do
4.        if (L^R_CF < 0) then migrate both E and F to the unordered-wait set; end_if
5.      end_for
6.      for each P in the unordered-wait set where P is starting point of cont. link do
7.        if (L^R_PF < 0) then migrate both E and F to the unordered-wait set; end_if
8.      end_for
9.      for each F in the pre-wait set where F is the end point of a cont. link EF do
10.       if (0 ≤ U^R_AF ≤ L^C_AC) then migrate E and F to unordered-wait set; end_if
11.     end_for
12.     mark A as visited, and all the other nodes as not visited;
13.     call GuardDFS(C);
14.   until (the unordered-wait set does not change)
15. end_procedure

16. procedure GuardDFS(P) // P is the current node in the DFS traversal //
17.   if P has been visited then return; end_if
18.   mark P as visited;
19.   if P is in pre-wait set then migrate P to unordered-wait set; return; end_if
20.   for each S where S is a direct descendant of P do call GuardDFS(S); end_for
21. end_procedure
```

Fig. 6. The wait reduction algorithm

b) *Pre-migrations* are used to find nodes in the pre-wait set whose waits may affect other nodes in the pre-wait set. For each F in the pre-wait set, if $0 \leq U^R_{AF} \leq L^C_{AC}$ (which means that w_{AFC} can be positive) and F is the end point of contingent link EF, we migrate both E and F to the unordered-wait set (Line 7).

c) *Guard-migrations* are used to find nodes in the pre-wait set whose waits will eliminate triangular-wait constraints for the remaining nodes in the pre-wait set. This step is achieved by finding *guard* nodes that are in the pre-wait set and that appear first in each path starting from a node, say C. Hence, for each P that is not directly connected to C, there are guard nodes on every possible path between P and C. Based on the last observation in this subsection, the upper bound of the non-existent link PC must equal to the upper bound of one of the paths from P to C, where the upper bound of a path is the sum of the upper bounds of all the links along the path. Assuming guard node D on a path from P to C, one can easily verify that the triangular wait w_{APC} on AP is equal to the regression wait of w_{ADC} through requirement link PD. Note that PD cannot be a contingent link because it has been ruled out by pre-migrations. This regression will not introduce new constraints if all the constraints formulated on w_{ADC} are satisfied. In this way, the constraints formulated on the guard nodes will ensure that the rest of the nodes in the pre-wait set can be safely ignored.

To find the guard nodes, we treat S_c as an undirected graph and apply the following modified depth-first search (DFS). Let C be the root and A be already visited. Consider any node B visited during the traversal.

(1) If B is in the post-wait set or the unordered-wait set, then mark B as visited and continue (Line 12).
(2) Otherwise, B is a guard node. In this case, the DFS migrates B to the unordered-wait set (Line 14), and mark B as visited. At this point, the DFS does not continue from the current node but returns to the parent of the current node before continuing (Line 17).

In the last step, when a guard node is migrated, the formulation of wait constraints on the node may add new links to S_c (if they do not already exist), and the updated network will lead to different guard nodes found by the search. Hence, we repeat the search until no new links are added. As an illustration, the DFS will classify E in Figure 5 as a guard node and put it into the unordered-wait set, while leaving F in the pre-wait set.

After migrating nodes from the pre-wait and post-wait sets to the unordered-wait set, the following lemma states that only nodes in the unordered-wait set are needed in formulating the wait constraints. (See proof in Appendix C.)

Lemma 5.1. *If the shortest-path, precede, and wait constraints on the unordered-wait set are satisfied, then excluding the updated pre-wait and post-wait sets in the formulation involving wait (triangular-wait, regression-wait, and wait-bound) constraints will not change the space of feasible solutions of* P_{stnu}.

5.2. *Reductions on precede constraints*

Similar to the reductions on wait constraints, we classify each node B of S_c into three subsets with respect to contingent link AC and another link CB that can be a requirement or a contingent link:

$$\begin{cases} \{B \text{ such that } L_{CB} \geq 0\} & \text{(post-wait set)} \\ \{B \text{ such that } U_{CB} < 0\} & \text{(pre-wait set)} \\ \text{otherwise} & \text{(unordered-wait set).} \end{cases}$$

For any B in the post-wait set, ABC must be in the follow case, and no precede constraints will be generated. For any B in the unordered-wait set, since the type of ABC is undetermined, nonlinear precede constraints will need to be generated.

Similar to the reductions on wait constraints in Section 5.1, reductions are applied on each node in the pre-wait set by finding guard nodes using the following modified DFS algorithm and by migrating them to the unordered-wait set. The algorithm treats S_c as an undirected graph, and takes C as the root and A as visited. Consider any node B in the process of traversal:

(1) If B is in the post-wait set or the unordered-wait set, mark B as visited and continue;

(2) Otherwise, B is a guard node. In this case, mark B as visited; migrate B to the unordered-wait set; and return to the parent of the current node before continuing.

As is stated in the last subsection, when a guard node is migrated to the unordered-wait set, the formulation of wait and precede constraints on the node may add new links to S_c, and the updated network will lead to different guard nodes found by the search. Hence, we repeat the search for wait and precede reductions until no new guard nodes are found.

Likewise, we have the following lemma. We omit its proof because it is similar to the second part of the proof for Lemma 5.1.

Lemma 5.2. *If the shortest-path constraints are satisfied, then excluding nodes in the post-wait and pre-wait sets in formulating the precede constraints will not change the space of feasible solutions of* P_{stnu}.

5.3. *Reductions on shortest-path constraints*

In our naïve formulation, there are a large number of shortest-path constraints because all possible triangles in S_c are considered. To reduce such constraints, we only formulate them between neighboring nodes in the reduced formulation, and propagate the constraints on bounds throughout the network.

Our algorithm for reducing the shortest-path constraints works by recursively finding new candidate nodes in S_c, formulate the corresponding shortest-path constraints, and eliminate the nodes from S_c until no nodes are left. The process of elimination works as follows. For any A in the distance graph, we identify nodes that are connected to A by existing links in its *adjacent set* and generate the shortest-path constraints on those triangles made up by A and any two nodes in its adjacent set. We assume that nodes in the adjacent set will be connected by existing links, or new links will be created otherwise. After generating the constraints, A will be eliminated from the network, and the procedure is repeated on the remaining graph.

The following lemma states that the shortest-path constraints will only need to be formulated on nodes in the adjacent set in each step. (See proof in Appendix D.)

Lemma 5.3. *If nodes are eliminated one by one from* S_c *using the above steps, then the shortest-path constraints between any two nodes in* S_c *are always satisfied.*

Since shortest-path constraints are formulated on triangles in S_c, the number of such constraints is proportional to the number of triangles. We have seen that when a node is removed from S_c, additional links may be added to S_c, leading to an increase in the number of triangles. To get a reduced formulation, we like to add as few links as possible and remove nodes in a proper order from the distance graph. For example, in Figure 7, if B is removed first, then A and D are in the adjacent set of B, and BAC, BAD, and BCD will be considered in formulating

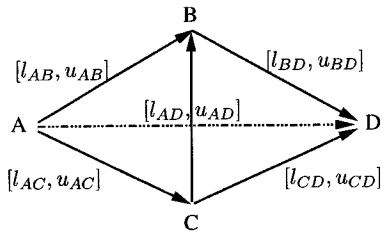


Fig. 7. The order of node removals affects the number of shortest-path constraints. (AD is a link added to the network.)

the shortest-path constraints after adding AD. Moreover, shortest-path constraints will be formulated on ACD, leading to four triangles to be considered. On the other hand, if A is removed first, then only the shortest-path constraints for two triangles (ABC followed by BCD) will be considered, and those of ABD and ACD are redundant.

We have developed a heuristic algorithm to identify a proper order for removing nodes. We define a heuristic value $v_A = \mathcal{L}_A / C_2^{\mathcal{S}_A}$ for node A, where $\mathcal{S}_A$ is the size of the adjacent set of A and $\mathcal{L}_A$ is the number of existing links in the adjacent set. If $v_A = 1$, then A is an ideal node to be removed; that is, removing A does not add any new links. In each step, the algorithm removes the node with the maximum v.

As an illustration, using the reduction methods proposed, the naïve formulation of the *STNU* in Figure 1 with 116 variables and 1,173 constraints can be reduced to 53 variables and 263 constraints. In particular, in the naïve formulation, the number of bound variables, wait variables, auxiliary variables, bound constraints, shortest-path constraints, precede constraints, triangular-wait constraints, regression-wait constraints, and wait-bound constraints are, respectively, 82, 32, 2, 123, 720, 38, 32, 224, and 36. In contrast, in the reduced formulation, the corresponding numbers are, respectively, 42, 9, 2, 63, 156, 18, 9, 4, and 13.

6. Experimental Results

In our experiments, we generate our *STNU*s randomly using the code developed by Cherkassky *et al.*[2] We choose the GRID family that closely approximates *STNU* s found in natural planning problems, each with L horizontal layers, H vertical layers, one source node, and $L \cdot H + 1$ nodes. The links are either horizontal or vertical, with a contingent link chosen randomly among the horizontal links and at most one contingent link in each layer. We restrict the lower bounds to be negative and the upper bounds to be positive in each vertical link, and choose the bounds on requirement links to be so loose that the generated *STNU*s are guaranteed to be dynamically controllable.

To test the scalability of our proposed methods, we generate five groups of configurations by varying the number of nodes from 40 to 200, where 200 is a reasonably large number of nodes that we can currently solve within several minutes

Table 2. Topology of the *STNU*s tested in terms of the number of nodes in the GRID network (*Nodes*), number of horizontal and vertical layers (*Layers* and *Height*), total number of links (*Links*), and number of contingent links (*Ctg.*).

ID	Nodes	Layers	Height	Links	Ctg.
1-a	41	20	2	60	12
1-b	41	10	4	70	8
1-c	41	8	5	72	6
2-a	81	40	2	120	25
2-b	81	20	4	140	17
2-c	81	16	5	144	14
3-a	121	60	2	180	36
3-b	121	30	4	210	26
3-c	121	24	5	216	22
4-a	161	80	2	240	52
4-b	161	40	4	280	33
4-c	161	32	5	288	28
5-a	201	100	2	300	63
5-b	201	50	4	350	42
5-c	201	40	5	360	37

of CPU time. In each group, we vary its configuration by changing its height to 2, 4, and 5, respectively. Table 2 summarizes the configuration of the *STNU*s tested. We also vary the random seed in generating contingent links and generate ten random networks in each configuration. We observe that a network with more levels will involve more interactions among its nodes. Hence, a network of height five is expected to be more difficult to solve than one of height two.

Using an objective of minimizing a linear cost function $\sum_{e \in E \setminus C}(u(e) - \ell(e))$ on the total duration of all requirement links, we formulate the optimization of bounds for both the naïve and the reduced formulations as nonlinear programming problems and solve them by SNOPT.[a]

Table 3 presents the complexities of our two formulations and the average CPU times taken by SNOPT over 10 random networks for each configuration.

In the naïve formulation, the majority of the constraints are shortest-path constraints, where the number of such constraints (C_3^N) only depends on N and not on the topology of the *STNU*. On the other hand, the number of other constraints vary with topology. For example, it increases with the number of levels in the network. Since our reduction techniques are only applicable on linear constraints, the number of nonlinear constraints remains unchanged in the reduced formulation.

We also observe that reductions perform better when the height of the graph is small. This is true because most nodes in the same layer with the same starting and end points will be in the unordered-wait set when we restrict every vertical link to

[a]SNOPT is a state-of-the-art nonlinear programming solver that is available at the NEOS server *http://www-neos.mcs.anl.gov*.

Table 3. Statistics on the number of variables and the number of constraints in our naïve and reduced formulations. The variables in each formulation are classified into bound, wait, and auxiliary, whereas the constraints are classified into bound, shortest-path, precede, triangular-wait (tri.-wait), regression-wait (reg.-wait), and wait-bound. The precede, regression-wait, and wait-bound constraints are further classified into linear (Lin.) and nonlinear (NLin.). For each configuration, we show its average and standard deviation of the solution time in seconds over 10 random networks by SNOPT on the NEOS Server.

ID	Variables				Constraints										Results (sec.)	
	Bound Var.	Wait Var.	Aux Var.	Total	Bound Const.	Shortest-Path	Precede Lin.	Precede NLin	Tri.-Wait	Reg.-Wait Lin.	Reg.-Wait NLin.	Wait-Bound Lin.	Wait-Bound NLin.	Total	Avg. Time	Std. Dev.
Naive Formulations																
1-a	1616	468	15	2099	2424	63960	444	24	468	17784	3	499	24	85630	–	–
1-b	1623	319	28	1971	2435	63960	296	49	319	12152	3	376	49	79642	–	–
1-c	1626	265	31	1923	2439	63960	232	55	265	10077	4	328	54	77416	–	–
2-a	6430	1975	31	8436	9645	511920	1888	50	1975	154050	6	2037	50	681622	–	–
2-b	6445	1358	61	7865	9668	511920	1363	103	1358	105986	9	1480	103	631993	–	–
2-c	6450	1153	68	7672	9676	511920	1120	117	1153	89965	9	1289	116	615368	–	–
3-a	14447	4331	47	18826	21670	1727880	4494	72	4331	511128	11	4426	72	2274089	–	–
3-b	14466	3165	95	17727	21700	1727880	3067	159	3165	373517	15	3355	159	2133019	–	–
3-c	14475	2641	101	17219	21713	1727880	2580	178	2641	311732	12	2845	177	2069761	–	–
4-a	25656	8268	68	33992	38484	4095840	7922	104	8268	1306344	16	8404	104	5465487	–	–
4-b	25693	5310	117	31121	38539	4095840	5107	200	5310	839074	17	5545	200	4989835	–	–
4-c	25702	4547	130	30380	38554	4095840	4404	231	4547	718489	16	4808	228	4867119	–	–
5-a	40073	12616	83	52772	60109	7999800	12792	126	12616	2498086	19	12782	126	10596461	–	–
5-b	40114	8517	151	48783	60171	7999800	8481	256	8517	1686405	23	8820	256	9772733	–	–
5-c	40126	7363	171	47660	60189	7999800	7186	298	7363	1457874	23	7705	296	9540734	–	–
Reduced Formulations																
1-a	213	48	15	276	319	856	72	24	48	11	3	79	24	1438	1.194	0.554
1-b	402	71	28	502	604	2944	94	49	71	27	3	128	49	3972	16.066	19.417
1-c	469	72	31	574	704	4064	91	55	72	41	4	136	54	5224	19.993	7.065
2-a	462	115	31	609	693	2124	181	50	115	20	6	178	50	3418	3.110	1.316
2-b	861	159	61	1081	1291	6574	216	103	159	70	9	281	103	8810	36.237	13.259
2-c	1032	164	68	1265	1549	9522	212	117	164	94	9	301	116	12088	128.255	13.737
3-a	674	159	47	881	1011	2889	246	72	159	37	11	254	72	4755	8.212	4.257
3-b	1344	258	95	1698	2016	10786	357	159	258	115	15	448	159	14317	100.734	19.787
3-c	1601	274	101	1977	2402	15250	370	178	274	121	12	477	177	19266	251.766	37.446
4-a	919	228	68	1216	1379	4038	352	104	228	55	16	365	104	6643	13.276	6.147
4-b	1769	333	117	2219	2653	14043	465	200	333	127	17	567	200	18608	154.296	17.704
4-c	2139	363	130	2632	3208	20490	497	231	363	153	16	623	228	25812	486.370	64.073
5-a	1144	280	83	1508	1717	5005	434	126	280	65	19	446	126	8223	19.260	8.738
5-b	2240	422	151	2814	3360	17809	588	256	422	178	23	726	256	23622	221.225	19.806
5-c	2700	447	171	3319	4051	25752	599	298	447	220	23	789	296	32478	837.102	199.752

have a negative lower bound and a positive upper bound. When the height of the network increases, reductions become less effective due to increases in the size of the unordered-wait set in wait reductions as well as in precede reductions.

Using the naïve formulations, SNOPT was not able to find any solution within 6,000 seconds, while all the reduced formulations can be solved within 1,000 seconds. The results show that our reduction techniques allow large problems to be solved within a reasonable amount of time.

7. Conclusions

In this paper, we have proposed and studied a novel problem of minimizing a cost function on the bounds of durations in a simple temporal network with uncertainty (*STNU*) in order for the *STNU* to be dynamically controllable. We show that the optimization problem is NP-hard, even for a linear cost function on the difference between the lower and upper bounds of the duration of each event. We also show that it is intractable to solve a naïve nonlinear constrained formulation that directly formulates its constraints from existing algorithms for checking dynamic controllability. By examining the redundancies in the constraints of the naïve formulation, we propose new methods for eliminating implied constraints. Our experimental results on some benchmarks demonstrate that our reduced formulations are effective and can be solved by existing nonlinear programming solvers.

Appendix A. A Complete List of Constraints

The constraints are described with respect to Figure 8, which is the same as Figure 2 except that the fixed bounds in Figure 2 are replaced by variable bounds. The constraints are derived based on the procedure in Section 2 for checking the dynamic controllability of each of the C_3^N triangles in the *STNU*. For simplicity, we have represented the constraints in (11), (13), and (16) without the use of auxiliary variables. However, as is discussed in Section 4, auxiliary variables will need to be introduced in order to transform these constraints and to remove their dependence on unknown variables in their conditions.

Constraints on bounds:

$$\begin{aligned} L^R_{AB} \le \ell^R_{AB} \le u^R_{AB} \le U^R_{AB} \\ L^R_{BC} \le \ell^R_{BC} \le u^R_{BC} \le U^R_{BC} \\ L^C_{AC} = \ell^C_{AC} \le u^C_{AC} = U^C_{AC}. \end{aligned} \tag{9}$$

Shortest-path constraints:

$$\ell^R_{AB} + \ell^R_{BC} \;\le\; \ell^C_{AC} \;\le\; \left\{ \begin{matrix} u^R_{AB} + \ell^R_{BC} \\ \ell^R_{AB} + u^R_{BC} \end{matrix} \right\} \;\le\; u^C_{AC} \;\le\; u^R_{AB} + u^R_{BC}. \tag{10}$$

Precede constraints:

$$\begin{cases} \ell^R_{BC} \cdot (\ell^R_{AB} + u^R_{BC} - u^C_{AC}) \geq 0 \\ \ell^R_{BC} \cdot (u^R_{AB} + \ell^R_{BC} - \ell^C_{AC}) \leq 0 \end{cases} \quad \text{if } L^R_{BC} < 0 \text{ and } U^R_{BC} \geq 0 \tag{11}$$

$$\begin{cases} \ell^R_{AB} = u^C_{AC} - u^R_{BC} \\ u^R_{AB} = \ell^C_{AC} - \ell^R_{BC} \end{cases} \quad \text{if } L^R_{BC} \geq 0.$$

Triangular-wait constraints:

$$w_{ABC} \geq u^C_{AC} - u^R_{BC} \quad \text{where } w_{ABC} \text{ is the wait on AB for contingent link AC.} \tag{12}$$

Regression-wait constraints (regressing wait w_1 to w_2):

$$w_2 \geq \begin{cases} w_1 - u^R & \text{through requirement link with upper bound } u^R \\ \begin{cases} w_1 - \ell^C \text{ if } w_1 \geq 0 \\ w_1 - u^C \text{ if } w_1 < 0 \end{cases} & \text{through contingent link with bounds } [\ell^C, u^C]. \end{cases} \tag{13}$$

Wait-bound constraints:

(1) Wait-bound of wait w on contingent link c with lower bound ℓ^C where the wait applies:

$$w = \ell^C. \tag{14}$$

(2) Upper bound of wait w on requirement link r with upper bound u^R where the wait applies:

$$w \leq u^R. \tag{15}$$

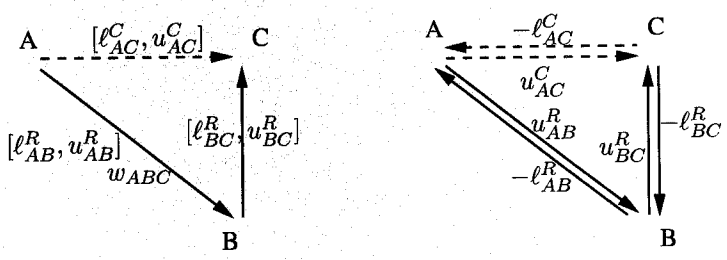
(3) Lower bound of wait w on requirement link r with initial bounds $[L^R, U^R]$ where the wait applies. For the contingent link c causing the wait, L^C is its lower bound, and $\ell^C = L^C$ and $u^C = U^C$.

$$\begin{cases} w = \ell^R & \text{if } U^R \leq L^C \\ \begin{cases} (\ell^R - \ell^C) \geq 0 & \text{if } w - \ell^C \geq 0 \\ (\ell^C - w)(\ell^R - w) \geq 0 & \text{if } w - \ell^C < 0 \end{cases} & \text{otherwise.} \end{cases} \tag{16}$$

Appendix B. Proof of Lemma 4.1

A triangle must be in the follow or the precede case when it is not in the unordered case. Assuming that the shortest-path and precede constraints are satisfied, we show that including the conditions for the unordered case will only lead to redundant triangular-wait constraints in the follow and precede cases. The proof is based on showing the following two propositions.

(a) The triangular waits in the follow and precede cases are redundant with respect to local dynamic controllability. Referring to Figure 8, consider triangular wait w_{ABC} defined in (2). It suffices to show that this triangular wait does not change the space of feasible solutions defined by the wait-bound constraints in



(a) Triangle ABC (b) The corresponding distance graph

Fig. 8. A triangle network with variable bounds in an *STNU* and its distance graph.

(14)–(16). We first prove the statement in the follow case, and the proof for the precede case is much simpler. In each case, we consider the cases when AB is a requirement link and when AB is a contingent link.

(i) Triangle ABC is in the follow case (*i.e.*, $u^R_{BC} < 0$).

First, assume AB to be a requirement link. From (2) and the shortest-path constraints in (10), we have:

$$w_{ABC} = u^C_{AC} - u^R_{BC} \leq u^R_{AB}. \tag{17}$$

This proves that the upper bound of the wait-bound constraint in (15) is not affected by the additional triangular-wait constraint. For the lower bound of the wait-bound constraint in (16), since ABC is in the follow case, we have $U^R_{AB} > L^C_{AC}$ (otherwise, ABC will be in the precede case). We need to consider (16) for the case when $U^R_{AB} > L^C_{AC}$. Because $u^R_{BC} < 0$, we have:

$$w_{ABC} = u^C_{AC} - u^R_{BC} \geq u^C_{AC} \geq \ell^C_{AC}. \tag{18}$$

By combining (18) with the shortest-path constraints in (10) and noting that $u^R_{BC} < 0$, we obtain:

$$w_{ABC} = u^C_{AC} - u^R_{BC} \geq \ell^R_{AB}, \quad \ell^C_{AC} \leq \ell^R_{AB} + u^R_{BC} \leq \ell^R_{AB}. \tag{19}$$

Under the condition that $U^R_{AB} > L^C_{AC}$, the inequalities in (19) prove that $w_{ABC} \geq \ell^R_{AB} \geq \ell^C_{AC}$. Hence, the lower bound of the wait-bound constraint in (16) is implied and is not affected by the additional triangular-wait constraint.

Second, assume AB to be a contingent link. Since ABC is in the follow case (with AC a contingent link), ACB must be in the precede case (with AB a contingent link). Using the precede constraint in (11) that $\ell^R_{AB} = u^C_{AC} - u^R_{BC}$, the triangular-wait constraint on AB is:

$$w_{ABC} = u^C_{AC} - u^R_{BC} = \ell^R_{AB}. \tag{20}$$

Hence, the wait-bound constraint in (14) for contingent link AB is not affected by the additional triangular-wait constraint.

(ii) Triangle ABC is in the precede case (*i.e.*, $\ell^R_{BC} \geq 0$), and the condition follows

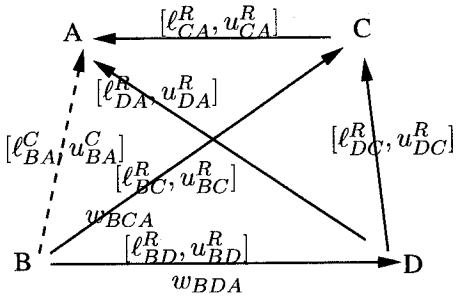


Fig. 9. Regressing wait w_{BCA} through a requirement link.

as in (20). Regardless of whether AB is a contingent or a requirement link, the wait-bound constraints are always satisfied.

(b) The triangular wait is redundant in the follow and precede cases with respect to global dynamic controllability. Since the wait on a link is the maximum of its triangular and regression waits, it suffices to show that the regression of a source triangular wait is no larger than any target triangular wait. Here, the source triangular wait is the triangular wait to be regressed from, and the target triangular wait is the triangular wait on the regressed target link.

Consider the waits in Figure 9. Here, w_{BCA} and w_{BDA} are the triangular waits in BCA and BDA, respectively, and BA is the contingent link causing the wait. From (2), we have:

$$w_{BCA} = u^C_{BA} - u^R_{CA}; \qquad w_{BDA} = u^C_{BA} - u^R_{DA}. \tag{21}$$

Let w_{BCA} be the source triangular wait in the follow or the precede case to be regressed, and w_{BDA} be an arbitrary target triangular wait. Assuming the regression wait of w_{BCA} to be w'_{BDA}, we show that $w_{BDA} \geq w'_{BDA}$. We consider two cases where DC is a requirement link and where it is a contingent link.

(i) Assuming DC to be a requirement link, the value of the regression wait of w_{BCA} is:

$$w'_{BDA} = w_{BCA} - u^R_{DC} = u^C_{BA} - u^R_{CA} - u^R_{DC}. \tag{22}$$

Using the shortest-path constraints in (10) on ADC where $u^R_{DC} + u^R_{CA} \geq u^R_{DA}$, we have:

$$w'_{BDA} = u^C_{BA} - u^R_{CA} - u^R_{DC} \leq u^C_{BA} - u^R_{DA} = w_{BDA}. \tag{23}$$

(ii) Assume DC to be a contingent link with C as the end point, and $w_{BCA} \geq 0$. (We do not need to consider the regression of a negative wait through a contingent link because, as is discussed in (4), it is the same as that through a requirement link). From the regression-wait constraints in (13), we have:

$$w'_{BDA} = w_{BCA} - \ell^R_{DC} = u^C_{BA} - u^R_{CA} - \ell^R_{DC}. \tag{24}$$

First, consider BCA to be in the follow case ($u^R_{CA} < 0$), which implies that DAC (with DC a contingent link) is in the precede case. Using the precede constraints in (11), we have $\ell^R_{DC} = u^R_{DA} - u^R_{CA}$ and

$$w'_{BDA} = u^C_{BA} - u^R_{CA} - \ell^R_{DC} = u^C_{BA} - u^R_{DA} = w_{BDA}. \tag{25}$$

Second, consider BCA to be in the precede case ($\ell^R_{CA} > 0$). From the shortest-path constraint in (10) on ADC, we have $\ell^R_{DA} \geq \ell^R_{DC} + \ell^R_{CA} > 0$. (Here we use the assumption that ℓ^R_{DC} is positive. Note that DC is a contingent link in this derivation, although it is marked as a requirement link in Figure 9.) Hence, both BCA and BDA are in the precede cases. Combining (2) with the precede constraints in (11) on these two triangles:

$$w_{BCA} = u^C_{BA} - u^R_{CA} = \ell^R_{BC}, \quad w_{BDA} = u^C_{BA} - u^R_{DA} = \ell^R_{BD}. \tag{26}$$

By the shortest-path constraints in (10) on BDC, we have $\ell^R_{BC} \leq \ell^R_{BD} + \ell^R_{DC}$, and the regression of w_{BCA} is:

$$w_{BDA} = \ell^R_{BD} \geq \ell^R_{BC} - \ell^R_{DC} = w_{BCA} - \ell^R_{DC} = w'_{BDA}. \tag{27}$$

In conclusion, we have shown that triangular waits in both the precede and follow cases are redundant because they satisfy all the wait-bound constraints. The regressions of these triangular waits are also redundant, since the corresponding regression waits are no larger than the triangular wait on the target link.

Appendix C. Proof of Lemma 5.1

The proof consists of showing that the space of feasible solutions will not change by excluding the post-wait and pre-wait sets in the wait-constraint formulation.

(a) There are four parts in proving that the post-wait set can be excluded from the formulation.

First, we prove that all the triangular waits in the post-wait set satisfy the wait-bound constraints. For any node F in the post-wait set, consider triangular wait w_{AFC} on link AF.

(1) Let AF be a contingent link. Then the wait-bound constraint for the wait on AF is $w_{AFC} = \ell^C_{AF}$, according to (14). From post migrations, we know that $\ell^R_{CF} \geq 0$. Given that AC is a contingent link, AFC is in the follow case, and AF can be taken as a requirement link. In this case, we have already proved in (20) that the triangular wait $w_{AFC} = \ell^R_{AF}$. Note that there are two contingent links AC and AF in this triangle. Hence, the triangle will be considered twice, with each contingent link playing the role of a requirement link.

(2) Let AF be a requirement link. By the definition of the post-wait set, $\ell^R_{AF} \geq \ell^C_{AC}$, and the lower-bound wait constraint is satisfied according to the second observation in Section 5.1. By combining the shortest-path constraints in (10) on AFC, it is easy to verify that the upper-bound wait constraint is satisfied.

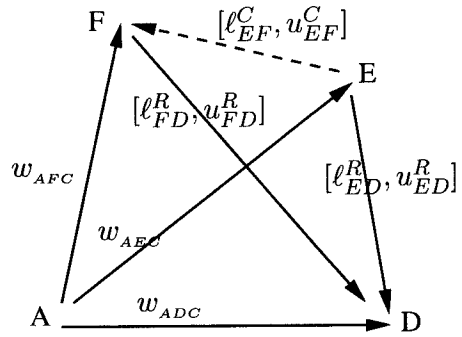


Fig. 10. Regressing waits from the unordered-wait set to the post-wait set.

Second, we prove that the regression of triangular waits in the post-wait set will not increase any other waits. For any node E, consider node F in the post-wait set.

(1) Let EF be a requirement link. Using the shortest-path constraints in (10) on AFE, one can show that the regression of triangular wait w_{AFC} is no larger than the triangular wait w_{AEC}. Since the wait on AE is chosen to be the maximum of all waits, we conclude that the regression of wait w_{AFC} would not increase the wait on AE.
(2) Let EF be a contingent link (with F as the end point). We know from post migrations that $\ell^R_{CF} \geq 0$ and AFC (with AC a contingent link) is in the follow case. We have shown in (25) that the regression of triangular waits through a contingent link in the follow case does not increase other waits.

Third, we prove that, if waits on nodes in the unordered-wait set satisfy the wait-bound constraints, then their regression to the post-wait set will not violate the wait-bound constraints. Since we have already shown in the second observation in Section 5.1 that a wait on nodes in the post-wait set satisfies the lower-bound wait constraints, we only need to examine the upper-bound wait constraints.

Referring to Figure 10, assume D to be in the unordered-wait set, and E in the post-wait set. Based on the third observation in Section 5.1, we conclude that only the upper-bound wait constraints on AE, where E in the post-wait set is a starting point of contingent link EF, have to be examined. From post migrations, we know that, for any contingent link, if its end point is in the unordered-wait set, then the corresponding starting point is also in the unordered-wait set. Thus all regressions from the unordered-wait set to the post-wait set are through requirement links. To see the regression effect on AE, the only case to be considered is first regressing the wait on AD (where D is in the unordered-wait set) to AF through requirement link FD, then regressing the wait on AF to AE through contingent link EF. Assume the wait on AD to be w_{ADC}, whose regression wait to AF is w_{AFC}. Let w_{AEC} be the wait regressed from w_{AFC}, and w'_{AEC} be the wait directly regressed from w_{ADC} through requirement link ED.

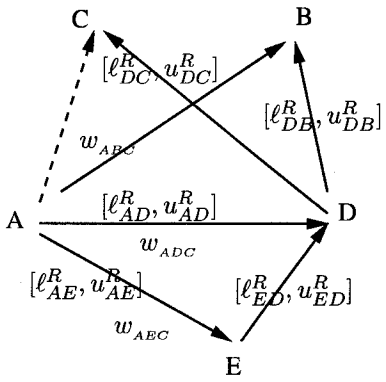


Fig. 11. Proof on excluding nodes in the pre-wait set from the formulation. Here, E is in the pre-wait set, and D is the guard node along the path from E to C and from E to B.

From post migrations, we have $\ell^R_{DF} \geq 0$. Hence, EDF is in the precede case, and we have:

$$u^R_{ED} = \ell^C_{EF} + u^R_{FD}. \tag{28}$$

Regressing the wait through FD and EF, we have:

$$w_{AEC} = w_{AFC} - \ell^C_{EF} = w_{ADC} - u^R_{FD} - \ell^C_{EF}. \tag{29}$$

Similarly, regressing the wait through ED, we have:

$$w'_{AEC} = w_{AEC} - u^R_{ED} = w_{ADC} - u^R_{FD} - \ell^C_{EF} = w_{AEC}. \tag{30}$$

That is, the effect of regressing w_{ADC} to any node in the post-wait set can be considered as the regression through a requirement link. Based on the first and third observations in Section 5.1, all these regressions are redundant if the upper-bound wait constraints are formulated in the unordered-wait set.

Last, we need to prove that the regression of waits in the pre-wait set will not increase the waits of nodes in the post-wait set. This conclusion is proved in the second part in (b) below.

b) There are three parts in proving that the pre-wait set can be excluded from the formulation.

First, referring to Figure 11, for any node E in the pre-wait set, we prove that triangular wait $w_{AEC} = \ell^R_{AE}$; that is, all triangular waits on the link between A and a node in the pre-wait set already satisfy the wait-bound constraints. According to guard migrations, there is a guard node D on the shortest path from E to C. Hence, we have $u_{EC} = u^R_{ED} + u^R_{DC}$. Given the definition of pre-wait set that $u^R_{AE} \leq \ell^C_{AC}$ (*i.e.*, E is guaranteed to occur before C), AEC is in precede case. As a result,

$$\ell^R_{AE} = u^C_{AC} - u_{EC} = u^C_{AC} - u^R_{ED} - u^R_{DC}. \tag{31}$$

The triangular wait on AE satisfies the following equation:

$$w_{AEC} = u^C_{AC} - u_{EC} = u^C_{AC} - u^R_{ED} - u^R_{DC} = \ell^R_{AE}. \tag{32}$$

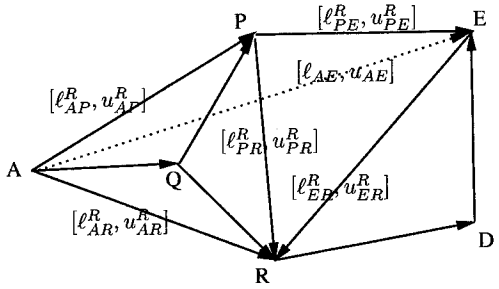


Fig. 12. Reductions on shortest-path constraints. AE is a non-existent link in the network.

Second, we prove that the regression of a triangular wait between A and a node E in the pre-wait set will not increase other waits. We examine the regression effect of w_{AEC}. If E in the pre-wait set is the end point of a contingent link from A, then from pre-migration, we know that $u^C_{AE} < 0$. Hence, $w_{AEC} = \ell^C_{AE} \leq u^C_{AE} < 0$, and all regressions are equivalent to regressions through requirement links. Combining with the shortest-path constraints, we know that the regression of a triangular wait through a requirement link will not raise the target triangular wait.

Last, we prove that, if waits on nodes outside the pre-wait set already satisfy the wait-bound constraints, then their regressions to nodes in the pre-wait set will also satisfy the constraints. That is, if the wait on AE in Figure 11 is raised to a value larger than the lower bound of AE due to the regression of wait w_{ABC} on AB outside the pre-wait set, then the wait on guard node D along the path from E to B must be larger than the lower bound of AD already. In Figure 11, assume that the regression of a wait on AB, w_{ABC}, causes the wait on AE to be raised to a value larger than the lower bound of AE. Combining with the shortest-path constraints:

$$\begin{aligned} & w_{AEC} = w_{ABC} - u^R_{DB} - u^R_{ED} > \ell^R_{AE} \\ \Rightarrow\ & w_{ABC} - u^R_{DB} > u^R_{ED} + \ell^R_{AE} \\ \Rightarrow\ & w_{ABC} - u^R_{DB} > \ell^R_{AD} \\ \Rightarrow\ & w_{ADC} > \ell^R_{AD}. \end{aligned}$$

Since all guard nodes are previously in the pre-wait set, the existing constraints would imply $w_{ADC} = \ell^R_{AD}$, which ensures that all waits on nodes in the pre-wait set will not violate the wait-bound constraints even if they are found by regressions from waits outside the pre-wait set.

Appendix D. Proof of Lemma 5.3

Suppose A in Figure 12 is the candidate node to be eliminated, and P, Q, and R form the adjacent set A. Assume that the shortest-path constraints on APQ, APR, AQR, and the rest of the network that excludes A have been formulated.

We first prove that any node in the rest of the network will not lead to violations of the shortest-path constraints related to AP, AQ and AR. Consider AR as an

example. Given an arbitrary node E in the rest of the network, we know from the fourth observation in Section 5.1 that the upper bound of AE, which is a link that does not exist in the network, is the minimum of the upper bounds of all possible paths from A to E. Assuming $u_{AE} = u^R_{AP} + u^R_{PE}$, we have:

$$u^R_{AR} \leq u^R_{AP} + u^R_{PR} = u_{AE} - u^R_{PE} + u^R_{PR}. \tag{33}$$

Since the rest of the network satisfies the shortest-path constraints, we have:

$$u^R_{PR} \leq u^R_{PE} + u^R_{ER} \implies u^R_{AR} \leq u_{AE} + u^R_{ER}. \tag{34}$$

Similarly, we can prove that the shortest-path constraints with respect to the lower bound of AR are satisfied.

Last, we need to prove that A will not cause the violation of the shortest-path constraint of any link in the rest of the network. The proof is very similar to that above and will not be shown.

References

1. E.D. Anderson and K. D. Anderson. Presolving in linear programming. *Mathematical Programming*, 71:221–245, 1995.
2. B. Cherkassky, A. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd edition.* MIT Press / McGraw-Hill, 2001.
4. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
5. P. Morris and N. Muscettola. Managing Temporal Uncertainty Through Waypoint Controllability. *Prof. Int'l Joint Conf. on Artificial Intelligence*, pages 1253–1258, 1999.
6. P. Morris and N. Muscettola. Execution of Temporal Plans with Uncertainty. *Proc. National Conf. on Artificial Intelligence*, pages 491–496, 2000.
7. P. Morris, N. Muscettola, and T. Vidal. Dynamic Control of Plans with Temporal Uncertainty. *Proc. Int'l Joint Conf. on AI*, pages 494–499, 2001.
8. N. Muscettola. Computing the Envelop for Stepwise-Constant Resource Allocations. *Proc. Principles and Practice of Constraint Programming*, pages 139–154, 2002.
9. N. Muscettola, P. Morris, and I. Tsammardinos. Reformulating Temporal Plans For Efficient Execution. *Proc. Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 444–452, 1998.
10. T. Vidal. Controllability characterization and checking in Contingent Temporal Constraint Networks. *Proc. Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 559–570, 2000.
11. T. Vidal and H. Fargier. Contingent Duration in Temporal CSPs: from Consistency to Controllabilities. *Proc. IEEE Symposium on Temporal Representation and Reasoning*, pages 78–85, 1997.
12. T. Vidal and M. Ghallab. Dealing with Uncertain Durations in Temporal Constraint Networks dedicated to Planning. *Proc. European Conf. on Artificial Intelligence*, pages 48–54, 1996.