Optimization of Bounds in Temporal Flexible Planning with Dynamic Controllability [*]

Benjamin W. Wah and *Dong Xin*
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana
Urbana, IL 61801, USA
E-mail: {wah, dongxin}@manip.crhc.uiuc.edu

Abstract

*A temporal flexible planning problem can be formulated as a simple temporal network with uncertainty (*STNU*), whose links are classified as contingent and requirement links. The problem of constraint satisfaction for* STNU *has been characterized as controllability, where dynamic controllability is the most interesting and useful controllability property. In this paper, we study the assignment of bounds allowed on the requirement links in order for the resulting* STNU *to be dynamically controllable and the total cost over the allowed ranges of the requirement links to be minimized. Since the problem with a linear cost function is NP-hard, we formulate the dynamic controllability of an* STNU *with a general cost function as constraints in a nonlinear optimization problem. Our approach is flexible because it can incorporate additional constraints, such as resource constraints, in the formulation. Finally, we present methods to reduce the number of constraints in order to make the problem tractable.*

1. Introduction

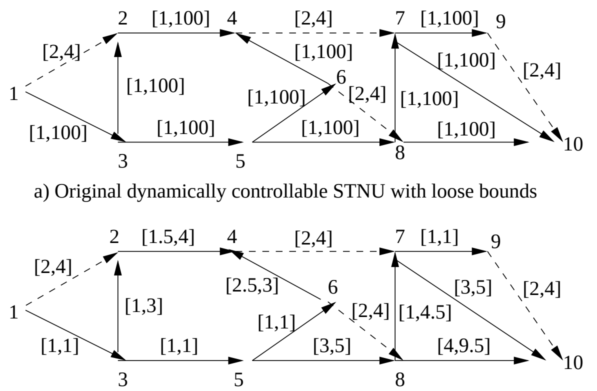
In temporally flexible planning, event that govern specific actions of a planner may be non-deterministic. To cope with uncertainties in the occurrence of events, the planner must respond to these contingent events and develop plans in such a way that all the problem requirements are satisfied eventually. As an example, the camera and the computer on a Mars rover may need to be turned on or off, depending on whether the rover encounters an obstacle in its path and whether the obstacle can be circumvented.

A temporal flexible planning problem can be formulated formally in a simple temporal network with uncertainty (*STNU*) [7], whose links are classified as *contingent* and *requirement links*. Contingent links may be thought of as representing causal processes of uncertain duration, whose final time points are controlled by nature, whereas requirement links are controlled by a planner. Formally, an *STNU* is described by a 5-tuple $\Gamma = \langle V, E, L, U, C \rangle$, where V is a set of nodes; E is a set of links; L: $E \rightarrow R \cup \{-\infty\}$ and U: $E \rightarrow R \cup \{+\infty\}$ are functions that map a link into the lower and upper bounds of the interval of possible durations; C is a subset of links that are contingent links; and $E \backslash C$ is the subset of requirement links.

The problem of constraint satisfaction for *STNU* has been characterized as *controllability* [6]. A network is controllable if there is a strategy for executing the actions by a planner that satisfies all the requirements in all situations involving contingent events. 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 future times of contingent events when all the past contingent events are known. It is easy to see from the definitions that strong controllability implies dynamic controllability; that is, if an *STNU* is strongly controllable, then it must be dynamically controllable. Dynamic controllability in turn implies weak controllability. 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 study the following optimization problem. Define a dynamically controllable *STNU* $\Gamma = \langle V, E, L, U, C \rangle$, where $[L(e), U(e)]$ are the loose bounds of

[*] Research supported by National Science Foundation Grant IIS 03-12084
Proc. IEEE Int'l Conference on Tools with Artificial Intelligence, 2004



a) Original dynamically controllable STNU with loose bounds

b) Dynamically controllable STNU with optimized bounds

Figure 1. Example of dynamically controllable *STNU* with optimized bounds. Solid arrows represent requirement links, and dashed arrows represent contingent links.

$e \in E$; and $[l(e), u(e)]$ are the desired bounds of e. Assuming a cost function f: $l(e_1), u(e_1), \ldots, l(e_n), u(e_n) \rightarrow R$, where $n = |E|$, the optimal *STNU* with respect to f, $\Gamma' = \langle V, E, l, u, C\rangle$, is the solution over $l(e_i)$, $u(e_i)$:

$$\begin{array}{rll} (P_{stnu}): \ \min & f(l(e_1), u(e_1), \ldots, l(e_n), u(e_n)) & e_i \in E \\ \text{subject to} & L(e) \le l(e) \le u(e) \le U(e) & e \in E \backslash C \\ & l(e) = L(e), u(e) = U(e) & e \in C \\ \text{and} & \Gamma' \text{ is dynamically controllable.} & \end{array}$$

P_{stnu} is interesting in practice. A larger interval on a requirement link is always desirable because the resulting *STNU* may be more likely to be dynamically controllable. However, such flexibility may incur new costs. For instance, the cost of allowing a camera onboard a satellite be turned on at any time in [10,20] may be higher than that if the camera were allowed to be turned on in [18,20]. The larger interval in the first case may incur an additional cost because other resources must be made available earlier.

Figure 1a illustrates P_{stnu}, whose dashed (*resp.* solid) lines are contingent (*resp.* requirement) links. It is easy to verify that the network is dynamically controllable. If we set the cost function as $\sum_{e \in E \backslash C}(u(e) - l(e))$, then the *STNU* has objective value 1089. By reducing the bounds on each requirement link, Figure 1b shows the modified P_{stnu} with objective value 18 that is also dynamically controllable.

P_{stnu} with a linear cost function can be proved to be NP-hard by reducing it from the 3-coloring problem. This is done by constructing a mapping from the 3-coloring problem to an instance of P_{stnu} in such a way that a solution to the 3-coloring problem maps to a solution of P_{stnu} constructed. Due to space limitation, we do not show the proof.

Previous studies on *STNU*s have focused on algorithms for checking controllability [4] and for executing an *STNU* efficiently and successfully [3]. Section 2 reviews existing algorithms for checking dynamic controllability. Although such algorithms have polynomial complexity [4], P_{stnu} is NP-hard even when a linear cost function is involved. In this paper, we formulate P_{stnu} with a general cost function 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 [5], in the formulation. Since existing methods for checking dynamic controllability are procedural [4], we first define in Section 3 the constraints that specify the conditions for dynamic controllability in our constrained formulation. We show a naive formulation that leads to an intractable problem with $\mathcal{O}(N^2)$ variables and $\mathcal{O}(N^3)$ constraints for an N-node *STNU*. In order to reduce the complexity, we propose in Section 4 methods to eliminate unnecessary variables and implied constraints. Finally, Section 5 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 [4]. We further assume that the lower bounds of contingent links are positive because influences of contingent events should propagate only forward in time. The dynamic controllability of an *STNU* is classified into local and global dynamic controllability [4].

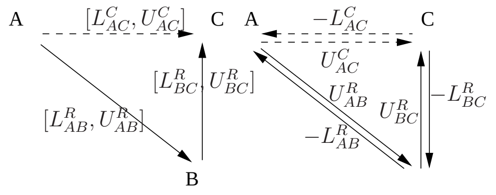
2.1. Local Dynamic Controllability

By treating an N-node *STNU* as C_3^N triangles, local dynamic controllability examines each triangle in two steps.

The first step treats each contingent link as a requirement link and examines each triangle in the network and the associated *distance graph* [2]. Each directed link in the triangle corresponds to two directed edges in the distance graph, whose weights are derived from the upper and the negative lower bounds of the corresponding links (Figure 2). Moreover, the bounds of each link in the triangle are the shortest paths in the distance graph. For instance, it follows that $[L^R_{BC}, U^R_{BC}] \subseteq [L^C_{AC} - U^R_{AB}, U^C_{AC} - L^R_{AB}]$ in Figure 2, 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 that has at least one contingent link is classified into one of the following three categories. (If a triangle has two contingent links, it will be considered twice, with each contingent link in turn plays the role of a requirement link.) Refer to Figure 2a in the following discussion.

a) If $U^R_{BC} < 0$, the triangle is in the *follow case*, and B always follows C in its occurrence. Here, contingent link AC acts like a requirement link because C has already oc-



a) Triangle in *STNU* b) Corresponding distance graph

Figure 2. Triangle network and its distance graph

curred at the time when B is scheduled to occur. The condition for dynamic controllability is always satisfied.

b) If $L^R_{BC} \geq 0$ and $U^R_{BC} > 0$, the triangle is in the *precede case*, and B must occur before or simultaneously with C. Here, the information about the occurrence of C is not available to B when B is scheduled. Hence, 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 the triangle is in the *unordered case*, and B can occur before or after C. Here, if C has not occurred, 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:

$$w_{ABC} = U^C_{AC} - U^R_{BC} \quad \text{(triangular wait on AB for AC).} \qquad (1)$$

The triangular wait on AB 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. Clearly, 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 because the wait on the contingent link must equal the lower bound defined. For simplicity, we refer to the lower and upper bounds on the wait defined in (1) as the *wait-bound constraints*.

2.2. Global Dynamic Controllability

In global dynamic controllability, wait information is propagated throughout a network by *regression* [4]. Consider regressing $\langle C, w_{ABC} \rangle$ for a wait on AB to AD, where AC is the contingent link that causes the wait.

a) If there is any link DB with upper bound U^R_{DB}, then the wait regressed to AD is $\langle C, w_{ABC} - U^R_{DB} \rangle$.

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

To distinguish from *triangular waits*, we call the wait regressed to AD a *regression wait* in this paper. The actual wait $\langle C, w_{ABD} \rangle$ on AD will be the maximum of its regression and triangular waits.

By applying the steps for checking dynamic controllability, the bounds of links may be tightened. 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 any contingent link c [4].

3. Naive Formulation

Given that P_{stnu} is NP-hard for a linear cost function, we propose to formulate the dynamic-controllability conditions in P_{stnu} with a general cost function as constraints and solve the problem as a nonlinear constrained optimization problem. These conditions can be formulated as constraints by following the properties in Section 2. Appendix A presents the complete list of constraints that are grouped into *shortest-path*, *precede*, and *wait* constraints. The latter include constraints on triangular-wait, regression-wait and wait-bound.

In our formulation, we represent the desired bounds on a link as variables $[l, u]$, and the wait value on AB for contingent link AC as variable w_{ABC} or just w for simplicity.

Given *STNU* S_c, there are two steps in developing the constrained formulation. First, we generate precede and wait constraints according to each contingent link in S_c and add 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 constraints to ensure every bound to be the shortest path in the corresponding distance graph.

In contrast to the algorithm for checking controllability in which the bounds are known and the types of triangles are determined a priori, the bounds in P_{stnu} are variables. Hence, the conditions for the different cases must be incorporated as constraints in the formulation. The only exception is the conditions for the *unordered case* in which we prove that the associated constraints are implied. This is stated formally as follows:

Lemma 1 *If the shortest-path and precede constraints are satisfied, then removing the conditions for the unordered case will not change the solution region of P_{stnu}.*

The proof is not shown due to space limitations.

Because constraints can be translated directly from the properties in Section 2, we only illustrate the construction of the constraints for regression wait through a contingent link. Assuming wait w_2 obtained by regressing wait w_1 through a contingent link with bounds $[l^C, u^C]$, then:

$$w_2 \geq \begin{cases} w_1 - l^C & \text{if } w_1 \geq 0 \\ w_1 - u^C & \text{otherwise.} \end{cases} \qquad (2)$$

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

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

To combine the two cases into a single constraint, we generate a linear constraint that is true for both cases:

$$w_2 \geq w_1 - u^C. \tag{3}$$

We then introduce three constraints using an auxiliary variable α:

$$\alpha \geq 0; \quad \alpha \geq w_1; \tag{4}$$
$$\alpha \times (w_2 - w_1 + l^C) \geq 0.$$

In addition, we add $\alpha(\alpha - w_1)$ to the objective function in order to ensure that the α chosen is either 0 or w_1. Note that $\alpha \geq w_1$ if $w_1 \geq 0$ and $\alpha \geq 0$ if $w_1 < 0$.

The two constraints defined in (2) with respect to the sign of w_1 are not always needed because we can infer the sign of w_1 in some cases. Since the wait on a link must be within its initial loose bounds $[L_1, U_1]$ specified for the link, we know that $w_1 \geq 0$ if $L_1 \geq 0$, and that $w_1 < 0$ if $U_1 < 0$.

Table 1 shows that the naive formulation results in $\mathcal{O}(N^2)$ variables and $\mathcal{O}(N^3)$ constraints for an N-node *STNU*. Such a problem is usually too large to be solved. In the next section, we present methods to reduce the number of redundant variables and implied constraints.

4. Reduced Formulation

The naive formulation treats an *STNU* S_c as a clique and enumerates all possible triangles that lead to the numerous linear constraints, resulting in many redundant variables and implied constraints. Although pre-solving techniques and linear reductions in linear programming can help eliminate such redundancies, they are of limited use because they cannot find complex problem-specific redundancies.

In this section, we present methods to eliminate redundancies in our naive formulation. Instead of looking for redundancies directly, we examine the temporal order among nodes and links in S_c and analyze the relationship among the constraints in order to avoid generating implied constraints. Note that since every shortest path is embodied by existing links in S_c, it is not necessary to formulate shortest-path constraints on links that do not exist in S_c. However, contingency information has to be propagated in S_c before the shortest-path constraints can be derived when each contingent link is treated as a requirement link. This step is done by adding new links to S_c and by introducing wait and precede constraints on these links. These links are necessary because their wait and precede constraints give tighter bounds than the corresponding shortest-path constraints. The shortest-path constraints are then formulated in the updated network. The amount of reductions are, of course, problem dependent. In the worst case, if S_c is a clique, then all the shortest-path constraints are necessary and cannot be reduced.

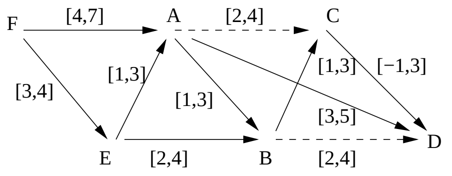


Figure 3. Reductions on wait constraints

In the following subsections, we introduce methods to reduce wait, precede, and shortest-path constraints.

4.1. Reductions on Wait Constraints

Before the reduction, we first check S_c for dynamic controllability and derive better bounds for all possible links, including those that do not exist in S_c.

Given contingent link AC, we derive the wait (triangular-wait, regression-wait, and wait-bound) constraints on every node in the naive formulation. To find redundant constraints, we consider temporal information based on the loose bounds and classify every node B of S_c into:

- *Pre-Set*: $\{B$ such that $U_{AB} \leq L^C_{AC}\}$,
- *Post-Set*: $\{B$ such that $L_{AB} \geq L^C_{AC}\}$,
- *Wait-Set*: otherwise,

where AB can be a requirement or a contingent link.

As an illustration, if AC in Figure 3 is a contingent link, then {E,F} is the Pre-Set, {D} is the Post-Set, and {B} is the Wait-Set. Basically, nodes in the Pre-Set (*resp.* Post-Set) are guaranteed to occur before (*resp.* after) some time point. Knowing that the wait on a link is always between its lower and upper bounds, we conclude that the wait on the link between A and any node in the Pre-Set (*resp.* Post-Set) is no larger (*resp.* no less) than L^C_{AC}.

As shown in Lemma 1, triangular waits in the precede and follow cases always satisfy the wait-bound constraints. However, Section 2.1 shows that nodes in the Pre-Set (*resp.* Post-Set) are not guaranteed to form triangles with contingent link AC in the precede (*resp.* follow) cases. Using a similar argument as in Lemma 1, triangular-wait constraints for nodes in the Post-Set are found to be redundant. For the Pre-Set, we find the necessary nodes to formulate the triangular-waits and migrate them to the Wait-Set, leaving other nodes in the Pre-Set to be ignored in the triangular-wait formulation. The migration procedure is presented as Guard-Migrations below.

Regression-wait constraints are more complex for two reasons. First, if waits in the Pre-Set and the Post-Set are found by regression, then the bounds on which the waits apply can be tightened. Second, if waits in the Pre-Set and the Post-Set are regressed to other links, then the bounds on the target links can also be tightened. In both case, the corresponding regression-wait constraints are no longer non-redundant. We have found that if a wait satisfies the wait-bound constraints, then its regression through a requirement link satisfies the wait-bound constraints. (If a wait is negative, then its regression through a contingent link is the same as through a requirement link.) As a result, we are only interested in waits that are positive and that can be regressed through a contingent link (that is, the end point of the link where the wait applies is the end point of a contingent link). The procedures are presented as Post-Migration (Pre-Migration) for nodes in the Post-Set (Pre-Set).

a) *Post-Migrations* are used to find waits in the Post-Set that may affect the bounds in the Wait-Set or the Pre-Set. For each F in the Post-Set, let F be an end point of contingent link EF, and P be either node C or a node in the Wait-Set that is the starting point of any contingent link. Both E and F can be migrated to the Wait-Set if $L^R_{PF} < 0$. Hence, the regression of wait w_{APC} to link AF and the regression of w_{AFC} through EF will be included in the formulation. This step is repeated until the Wait-Set does not change. As an illustration, node D in Figure 3 will be migrated to the Wait-Set, since D is the end point of contingent link BD and $L^R_{CD} < 0$.

b) *Pre-Migrations* are used to find waits in the Pre-Set that may affect other nodes in the Pre-Set. For each node F in the Pre-Set, if $0 \leq U^R_{AF} \leq L^C_{AC}$ (hence the value of wait w_{AFC} can be positive) and F is the end point of contingent link EF, migrate both E and F to the Wait-Set.

c) *Guard-Migrations* are used to find waits in the Pre-Set that will eliminate triangular-wait constraints on the remaining nodes in the Pre-Set. This step is achieved by finding *guard* nodes that are in the Pre-Set and that appear first in each path starting from C. Hence, for each node P that is not directly connected to C, there are guard nodes on every possible path between P and C. The upper bound of non-existent link PC must be equal to the upper bound of one of paths from P to C, where the upper bound of a path is the sum of the upper bounds of all the links in 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 equivalent 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-Migration. This regression will be possible if constraints formulated on w_{ADC} are satisfied. Consequently, the constraints formulated on the guard nodes ensure that the rest of the nodes in the Pre-Set can be ignored.

By treating the network as an undirected graph, we use a modified Depth First Search (DFS) to find the guard nodes. Take C as the root and A as visited. Consider any node B visited during the traversal:

1. If B is in the Post-Set or the Wait-Set, mark B as visited and continue.
2. Otherwise, B is a guard node; migrate B to the Wait-Set; and mark B as visited. At this point, DFS does not continue from the current node but return to the parent of the current node before continuing.

For example, DFS will classify E in Figure 3 as a guard node and put it into the Wait-Set, and leave F in the Pre-Set.

After the migrations and updating the Wait-Set, Pre-Set, and Post-Set, the following lemma states that only nodes in the Wait-Set are needed in the formulation involving wait constraints. We skip the proof due to space limitation.

Lemma 2 *If the shortest-path, precede, and wait constraints (triangle-wait, regression-wait, and bound constraints) on the Wait-Set are satisfied, then excluding the updated Post-Set and Pre-Set in the formulation involving wait constraints will not change the solution region of P_{stnu}.*

4.2. Reductions on Precede Constraints

Similar to reductions on wait constraints, we consider temporal information according to the loose bounds and classify every node B of S_c for contingent link AC into:

- Post-Set: $\{B$ such that $L_{CB} \geq 0\}$,
- Pre-Set: $\{B$ such that $U_{CB} < 0\}$,
- Unordered-Set: otherwise,

where CB can be a requirement or a contingent link. For any B in the Post-Set, triangle ABC must be in the follow case, and no precede constraints will be generated. For any B in the Unordered-Set, the type of triangle ABC is undetermined, and nonlinear precede constraints will be generated. Reductions can be applied on the Pre-Set, where, similar to the reductions on wait constraints, the set of guard nodes will be found by the modified DFS search and migrated to the Unordered-Set. Likewise, we state the following lemma without proof.

Lemma 3 *If the shortest-path constraints are satisfied, then excluding the Post-Set and the Pre-Set in the formulation involving precede constraints will not change the solution region of P_{stnu}.*

As stated earlier, the formulation of wait and precede constraints may add new links to S_c. The updated network

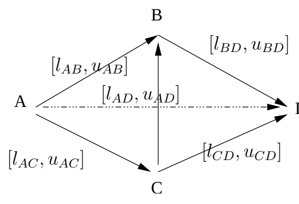


Figure 4. The order of node removals affect the number of shortest-path constraints. (AD is a link added to the network.)

will lead to different guard nodes found by DFS in both wait and precede reductions. In our algorithm, we redo both wait and precede reductions until no new guards are found.

4.3. Reductions on Shortest-Path Constraints

In a naive formulation, there are a large number of shortest-path constraints because all possible triangles in the network are considered. To reduce the number of 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 number of shortest-path constraints works by finding recursively new candidate nodes in the distance graph of S_c, formulate the corresponding shortest-path constraints, and remove the candidate nodes from the distance graph until no nodes are left. For any node A in the distance graph, we define 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 always be connected by existing links, or new links will be created in case they do not exist. After generating the constraints, we remove A from the distance graph and repeat the procedure on the remaining graph. One can easily verify that, if nodes are removed one by one this way, then the shortest-path constraints between any two nodes in the original distance graph will be satisfied.

Shortest-path constraints are formulated in triangles where the number of such constraints is proportional to the number of triangles in S_c. 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 in the distance graph.

For example, in Figure 4, if B is removed first, then BAC, BAD, and BCD will be considered in the shortest-path formulation after adding AD to the network, and the shortest-path constraints will be formulated in ACD. Hence, four triangles will be considered. On the other hand, if A is removed first, only the shortest-path constraints for two triangles (ABC followed by BCD) will be considered, and those of ABD and ACD are redundant. To illustrate that the latter is true, consider as an example one shortest-path constraint in ABD: $u_{AB} \leq u_{AD} - l_{BD}$. Since there is no direct link between AD, we assume the upper bound of AD to be $u_{AD} = u_{AC} + u_{CD}$. From the shortest-path constraints on ABC and BCD, we know that $u_{AB} \leq u_{AC} + u_{CB}$ and $u_{CB} \leq u_{CD} - l_{BD}$. These constraints imply that $u_{AB} \leq u_{AC} + u_{CB} = u_{AC} + u_{CD} - u_{CD} + u_{CB} \leq u_{AD} - l_{BD}$, which show that the constraint is redundant.

We have developed a heuristic algorithm to identify the order of 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 of the algorithm, the algorithm finds the node with the maximum v for removal.

As an illustration, using the proposed methods, the naive formulation in Figure 1 with 116 variables and 1173 constraints can be reduced to 53 variables and 263 constraints.

5. Experimental Results

In our experiments, we generated our *STNU*s randomly, using the same code as in [1]. We chose the GRID family that closely approximated *STNU*s found in natural plans, where $L \times H + 1$ is the number of nodes, L is the number of horizontal layers, H is the number of vertical heights, and one accounts for the source node. The links are either horizontal or vertical, where a contingent link is chosen randomly among horizontal links, and there is at most one contingent link in each layer. We restricted the lower bounds to be negative and the upper bounds to be positive in each vertical link. The bounds on requirement links were so loose that the generated *STNU*s were guaranteed to be dynamically controllable. In each network, we varied the topologies by changing its height to 2, 4, and 5, respectively. We also varied the random seed in order to get a different distribution of contingent links. Finally, we averaged our results over ten different random seeds.

We chose our cost function as $\sum_{e \in E \setminus C}(u(e) - l(e))$ in order to minimize the sum of durations of all requirement links. We formulated the optimization of bounds for both the naive and the reduced formulations as nonlinear programming problems and solved them by SNOPT at the NEOS server (*http://www-neos.mcs.anl.gov*).

Table 2 shows the formulation and computation results. Using the naive formulations, SNOPT was not able to find any solution within 6000s. Our results show that reductions are important because they allow large problems to be solvable within a reasonable amount of time, and that they per-

STNU	Topology				Naive Formulation				Reduced Formulation			
Nodes	Layers	Height	Links	Ctg.	Var.	Const.	NL	Time	Var.	Const.	NL	Time
41	20	2	60	12	2099	85630	51	-	276	1438	51	1.194
41	10	4	70	8	1971	79642	102	-	502	3972	102	16.066
41	8	5	72	6	1923	77416	114	-	574	5224	114	19.993
81	40	2	120	25	8436	681622	106	-	609	3418	106	3.110
81	20	4	140	17	7865	631993	215	-	1081	8810	215	36.237
81	16	5	144	14	7672	615368	243	-	1265	12088	243	128.255
121	60	2	180	36	18826	2274089	156	-	881	4755	156	8.212
121	30	4	210	26	17727	2133019	334	-	1698	14317	334	100.734
121	24	5	216	22	17219	2069761	369	-	1977	19266	369	251.766
161	80	2	240	52	33992	5465487	224	-	1216	6643	224	13.276
161	40	4	280	33	31121	4989835	417	-	2219	18608	417	154.296
161	32	5	288	28	30380	4867119	476	-	2632	25812	476	486.370
201	100	2	300	63	52772	10596461	273	-	1508	8223	273	19.260
201	50	4	350	42	48783	9772733	537	-	2814	23622	537	221.225
201	40	5	360	37	47660	9540734	617	-	3319	32478	617	837.102

Table 2. Complexity of test networks generated and their solution times in seconds on the NEOS Server. Each network is represented by the number of nodes (*Nodes*), the number of horizontal layers of the GRID network (*Layers*), the number of vertical layers of the GRID network (*Height*), the total number of links (*Links*), and the number of contingent links (*Ctg.*). Each formulation is represented by the number of variables (*Var.*), the number of linear constaints (*Const.*), the number of nonlinear constraints (*NL*), and the solution time in seconds of SNOPT on the NEOS server (*Time*).

form better when the height of the graph is small. The latter is true because most nodes in the same layer with a given starting point (*resp.* end point) will be in the Wait-Set (*resp.* Unordered-Set) when we restrict every vertical link to have negative lower and positive upper bounds. When the height of the network increases, reductions become less effective due to increases in the size of the Wait-Set in wait reductions and that of the Unordered-Set in precede reductions.

6. Conclusions

In this paper, we have presented a constrained formulation of finding the bounds allowed on the requirement links of a simple temporal network with uncertainties (STNU) in order for the resulting STNU to be dynamically controllable and the total cost over the allowed ranges of the requirement links to be minimized. We have first shown a naive formulation that treats an STNU as a clique and that enumerates all possible triangles. Such a formulation leads to many redundant variables and implied constraints, rendering the problem unsolvable even for a small network. To address this issue, we present methods to eliminate such redundancies by studying the temporal order among nodes and links and by analyzing the relationship among the constraints in order to avoid generating implied constraints. Our experimental results illustrate that our reduction methods lead to tractable solutions for reasonable large networks.

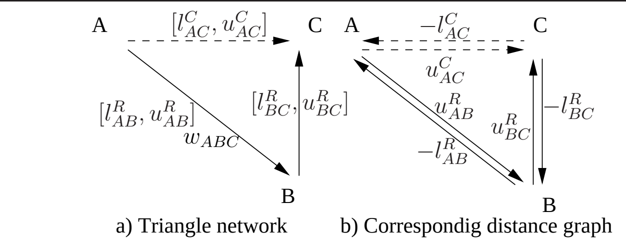


Figure 5. Illustration for constraints discussed in Appendix A. l and u denote the desired bounds.

APPENDIX A: COMPLETE CONSTRAINT LIST The constraints are discussed with respect to Figure 5.

Shortest-Path Constraints.

$$\begin{cases} l_{AC}^C \le u_{AB}^R + l_{BC}^R \le u_{AC}^C \\ l_{AC}^C \le u_{BC}^R + l_{AB}^R \le u_{AC}^C \\ u_{AB}^R + u_{BC}^R \ge u_{AC}^C \\ l_{AB}^R + l_{BC}^R \le l_{AC}^C. \end{cases} \tag{5}$$

Precede Constraints.

1. If $L_{BC}^R < 0$ and $U_{BC}^R \ge 0$,

$$\begin{cases} l_{BC}^R \times (l_{AB}^R + u_{BC}^R - u_{AC}^C) \ge 0 \\ l_{BC}^R \times (u_{AB}^R + l_{BC}^R - l_{AC}^C) \le 0. \end{cases} \tag{6}$$

2. If $L^R_{BC} \geq 0$,

$$\begin{cases} l^R_{AB} &= u^C_{AC} - u^R_{BC} \\ u^R_{AB} &= l^C_{AC} - l^R_{BC}. \end{cases} \tag{7}$$

Triangular Wait.

$$w_{ABC} \geq u^C_{AC} - u^R_{BC}, \text{ where } w_{ABC} \text{ is the wait on AB.} \tag{8}$$

Regression Wait ($w_1 \rightarrow w_2$).

1. Through a requirement link with upper bound u^R,

$$w_2 \geq w_1 - u^R. \tag{9}$$

2. Through a contingent link with bounds $[l^C, u^C]$, assuming $[L, U]$ to be the bounds on the link where w_1 applies:

$$w_2 \geq \begin{cases} w_1 - l^C & \text{if } L \geq 0 \\ w_1 - u^C & \text{if } U < 0. \end{cases} \tag{10}$$

Otherwise,

$$\begin{cases} \alpha & \geq 0 \\ \alpha & \geq w_1 \\ \alpha \times (w_2 - w_1 + l^C) & \geq 0 \\ w_2 & \geq w_1 - u^C, \end{cases} \tag{11}$$

where α is an auxiliary variable. An additional nonlinear part $\alpha \times (\alpha - w_1)$ is added to the objective function to ensure that α is either 0 or w_1.

Wait-Bound Constraints.

1. Wait-bound on contingent link, given l^C to be the lower bound of contingent link where the wait applies:

$$w = l^C. \tag{12}$$

2. Upper bound of wait on a requirement link where the wait applies:

$$w \leq u^R. \tag{13}$$

3. Lower bound of wait on a requirement link where the wait applies. $[L^R, U^R]$ are the loose bounds of the requirement link, and L^C is the lower bound of the contingent link causing the wait:

$$w = l^R \quad \text{if } U^R \leq L^C. \tag{14}$$

Otherwise,

$$\begin{cases} (l^C - w)(l^R - w) & \geq 0 \\ \beta & \geq 0 \\ \beta & \geq w - l^C \\ \beta(l^R - l^C) & \geq 0, \end{cases} \tag{15}$$

where β is an auxiliary variables. A nonlinear part $\beta(\beta - (w - l^C))$ is added to the objective function to ensure that β is either 0 or $w - l^C$.

APPENDIX B: PROOF OF LEMMA 1 A triangle must be in the follow or the precede case when it is not in the unordered case. Removing the condition for the unordered case will cause additional triangular-wait constraints in the follow and the precede cases to be formulated. Assuming that the shortest-path and precede constraints are satisfied, we show that those additional triangular-wait constraints in the follow and the precede cases are redundant. The proof is constituted by the following two propositions.

a) The triangular waits are redundant in the follow and precede cases with respect to local dynamic controllability. Referring to Figure 5, consider triangular wait w_{ABC} defined in (1). It is sufficient to show that this triangular wait does not change the solution region defined by wait-bound constraints. We first prove that the statement is true in the follow case, and the proof for the precede case is much simpler. In each case, we need to consider two different situations: AB is a requirement link and AB is a contingent link.

1. The triangle is in the follow case (*i.e.*, $u^R_{BC} < 0$). First, assume AB is a requirement link. From the shortest-path constraints, we have:

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

This proves that the upper bound of the wait-bound constraint in (13) is not affected by the additional triangular-wait constraint. For the lower bound of the wait-bound constraint, 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 (15) instead of (14). Because $u^R_{BC} < 0$, we have:

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

Combining the shortest-path constraints:

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

The above three inequalities prove that the lower bound of the wait-bound constraint in (15) is not affected by the additional triangular-wait constraint.

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

$$w_{ABC} = u^C_{AC} - u^R_{BC} = l^R_{AB}.$$

Hence, the wait-bound constraint in (12) for AB is not affected by the additional triangular-wait constraint.

2. The triangle is in the precede case (i.e. $l^R_{BC} \geq 0$). From the constraints in the precede case, we have $w_{ABC} = u^C_{AC} - u^R_{BC} = l^R_{AB}$. Regardless of whether AB is a contingent or a requirement link, all the wait-bound constraints are satisfied.

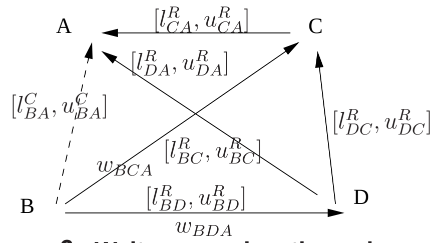


Figure 6. Wait regression through requirement link

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

Consider the waits in Figure 6. Here, w_{BCA}, w_{BDA} are triangular waits in BCA and BDA, respectively, and BA is the contingent link causing the wait:

$$w_{BCA} = u_{BA}^C - u_{CA}^R, \quad w_{BDA} = u_{BA}^C - u_{DA}^R.$$

Let w_{BCA} be the source triangular wait in the follow or precede case, 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 in which DC is a requirement link and DC is a contingent link.

1. Assume DC to be a requirement link. The value of the regression wait of w_{BCA} is:

$$w'_{BDA} = w_{BCA} - u_{DC}^R = u_{BA}^C - u_{CA}^R - u_{DC}^R.$$

Using the shortest-path constraints, $u_{DA}^R \leq u_{CA}^R + u_{DC}^R$:

$$w_{BDA} \geq u_{BA}^C - u_{CA}^R - u_{DC}^R = w'_{BDA}.$$

2. Assume DC to be a contingent link with C as the end point. We assume $w_{BCA} \geq 0$ (regression a negative wait through a contingent link is the same as through a requirement link). Then,

$$w'_{BDA} = w_{BCA} - l_{DC}^R = u_{BA}^C - u_{CA}^R - l_{DC}^R.$$

First, assume that BCA is in the follow case ($u_{CA}^R < 0$), which implies that DAC (with DC as contingent link) is in the precede case. Using the precede constraints, $u_{DA}^R = l_{DC}^R - (-u_{CA}^R)$, we have:

$$w_{BDA} = u_{BA}^C - u_{DA}^R = w'_{BDA}.$$

Second, consider BCA to be in the precede case ($l_{CA}^R > 0$). From the shortest-path constraint, we have $l_{DA}^R \geq l_{CA}^R + l_{DC}^R > 0$. (Here we use the assumption that the lower bound l_{DC}^R of contingent link DC is positive.) Hence, both BCA and BDA are in the precede case. Combining the precede constraints lead to:

$$w_{BCA} = u_{BA}^C - u_{CA}^R = l_{BC}^R, \quad w_{BDA} = u_{BA}^C - u_{DA}^R = l_{BD}^R.$$

Using the shortest-path constraint, $l_{BD}^R \geq l_{BC}^R - l_{DC}^R$, the regression of w_{BCA} is:

$$w_{BDA} = l_{BD}^R \geq l_{BC}^R - l_{DC}^R = w_{BCA} - l_{DC}^R = w'_{BDA}.$$

References

[1] B. Cherkassky, A. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.

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

[3] P. Morris and N. Muscettola. Execution of Temporal Plans with Uncertainty. *Proc. National Conf. on Artificial Intelligence*, 2000.

[4] P. Morris, N. Muscettola, and T. Vidal. Dynamic Control of Plans with Temporal Uncertainty. *Proc. Int'l Joint Conf. on Artificial Intelligence*, pages 494–499, 2001.

[5] N. Muscettola. Computing the Envelop for Stepwise-Constant Resource Allocations. *Proc. of 8th Int'l Conf. on Principles and Practice of Constraint Programming*, pages 139–154, 2002.

[6] T. Vidal. Controllability characterization and checking in Contingent Temporal Constraint Networks. *Proc. 7th Int'l Conf. on Principles of Knowledge Representation and Reasoning*, 2000.

[7] T. Vidal and M. Ghallab. Dealing with Uncertain Durations in Temporal Constraint Networks dedicated to Planning. *Proc. European Conf. on Artificial Intelligence*, 1996.