Efficient Message Passing and Propagation of Simple Temporal Constraints

Hung H. Bui, Mabry Tyson, and Neil Yorke-Smith
Artificial Intelligence Center, SRI International
333 Ravenswood Ave., Menlo Park, CA 94025, USA
{bui,tyson,nysmith}@AI.SRI.COM

Abstract

The Simple Temporal Network (STN) is a widely used framework for reasoning about quantitative temporal constraints over variables with continuous or discrete domains. Determining consistency and deriving the minimal network are traditionally achieved by graph algorithms (e.g., Floyd-Warshall, Johnson) or by iteration of narrowing operators (e.g., $\triangle$STP). However, none of these existing methods exploit effectively the tree-decomposition structure of the constraint graph of an STN. Methods based on variable elimination (e.g., adaptive consistency) can exploit this structure, but have not been applied to STNs, in part because it is unclear how to efficiently pass the 'messages' over a set of continuous domains. We first show that for an STN, these messages can be represented compactly as sub-STNs. We then present an efficient message passing scheme for computing the minimal constraints of an STN. Analysis of the new algorithm, *Prop-STP*, brings formal explanation of the performance of the existing STN solvers $\triangle$STP and SR-PC. Preliminary empirical results validate the efficiency of Prop-STP in cases where the constraint network is known to have small tree-width, such as those that arise in Hierarchical Task Network planning problems.

Introduction

Quantitative temporal constraints are essential for many real-life planning and scheduling domains (Smith *et al.* 2000). Many systems adopt a Simple Temporal Network (STN) (Dechter *et al.* 1991) to represent and reason over the temporal aspects of such problems, associating time-points with the start and end of actions, and modeling the temporal relations by binary simple temporal constraints. We present a general, efficient message passing scheme for propagation of such constraints.

The central role of STNs in deployed planning systems (Myers *et al.* 2002; Castillo *et al.* 2006) makes efficient inference with STNs especially important. The two principal inference tasks, determining consistency of an STN and deriving its minimal network, can be achieved by enforcing path consistency (PC) (Dechter *et al.* 1991). The common approach is to run an All-Pair Shortest Path graph algorithm on the *distance graph* of the STN. Algorithms such as *Floyd-Warshall* (denoted F-W or PC-1) (time complexity $\Theta(N^3)$), or *Johnson* (complexity $\Theta(N^2 \log N + NM)$, where M is the number of edges in the constraint graph) can be used (Cormen *et al.* 1990).

To achieve better efficiency, significant efforts have been made to apply more sophisticated constraint propagation techniques to STNs. *Partial Path Consistency* (PPC) (Bliek and Sam-Haroud 1999) can be applied to a triangulated constraint graph rather than a complete graph and is sufficient for backtrack-free reconstruction of all solutions. The state-of-the-art $\triangle$*STP* (Xu and Choueiry 2003) is a specialized solver based on PPC and operates over triangles of the triangulated STN. If only consistency is required, but not the minimal network, *Directional Path Consistency* (DPC) can be used with time complexity $\mathcal{O}(Nw^2)$, where w is the induced tree-width along the node ordering used (Dechter 2003). Empirical comparisons on random STNs (Xu and Choueiry 2003; Shi *et al.* 2004) show that $\triangle$STP outperforms PC-1, Johnson's Algorithm ("Bellman-Ford"), and is comparable to (on dense graphs) or outperforms (on sparse graphs) DPC. The complexity of $\triangle$STP, unstated in (Xu and Choueiry 2003), is not known, but can be bounded by $\mathcal{O}(N^3)$.

Despite the variety of methods for solving STNs, no dedicated STN solver takes full advantage of the tree-decomposition of the STN constraint graph (i.e., the ability to decompose a constraint graph into a 'tree' of variable and constraint clusters (Dechter and Pearl 1989)). One exception is the specialized solver SR-PC (Yorke-Smith 2005) that exploits the structure of STNs associated with plans in the Hierarchical Task Network (HTN) planning paradigm (Erol *et al.* 1994). The HTN planning process gives rise to STNs with the *sibling-restricted* (SR) property. Such an SR-STN can be decomposed into a tree of smaller sub-STNs, mirroring the shape of the hierarchical structure in the plan. SR-PC traverses this tree, invoking an STN solver at each sub-STN. While SR-PC shows strong empirical performance on SR-STN, it does not operate on general STNs.

In contrast to STNs, tree-decomposition methods are often applied to the general Constraint Satisfaction Problem (CSP). These methods, such as *variable elimination (adaptive consistency)* and *cluster-tree elimination* (Dechter 2003), operate by decomposing a triangulated constraint graph into a tree of variable clusters and solving the sub-problem in each.

In this paper we apply the ideas of such tree-decomposition methods to the STN. We note that a direct application of tree-decomposition method to the STN is non-trivial. Since the STN represents a CSP with *continuous* variables, it is not clear how to represent the 'messages', i.e., the sets of additional constraints resulting from eliminating some variables. We first show that, for an STN, these messages can be represented compactly as sub-STNs. We then present an efficient message passing scheme, called *Prop-STP*. Like △STP, our Prop-STP requires the STN to be triangulated. However, unlike △STP, our algorithm operates over the set of maximal cliques of the triangulated constraint graph. The time complexity of Prop-STP is $\mathcal{O}(Kw^3)$ where K is the number of cliques, and w is the induced tree-width (the size of the largest clique minus 1). For STNs with known and bounded tree-width (e.g., SR-STNs), Prop-STP thus achieves linear time complexity, a substantial improvement over the use of All-Pair Shortest Path algorithms. We demonstrated this empirically by showing that on structured SR-STNs, Prop-STP achieves the same level of performance as the specialized solver SR-PC, while greatly outperforming △STP.

For general STNs, triangulation can be carried out efficiently by greedy methods (Kjaerulff 1990). The results of (Shi *et al.* 2004; Xu and Choueiry 2003) demonstrate empirically that with a triangulation step, △STP outperforms current STN solvers. Our analysis of Prop-STP offers insight into how to order the triangles in △STP, and also shows that △STP's complexity can be characterized in terms of the induced tree-width. Our preliminary experiments with randomly generated STNs indicate that Prop-STP often performs better than △STP, thus, showing that it is more efficient to operate on cliques rather than on triangles (as anticipated by (Choueiry and Wilson 2006)).

The next section presents necessary background on STNs and variable elimination. The following sections introduce our message passing scheme for STNs and the resulting *Prop-STP* algorithm. We present a proof of its correctness and analyze its theoretical complexity. Finally, we present preliminary empirical validation of Prop-STP on SR STNs.

Background

We are concerned with relations among a set of variables $\{x_i,\ i \in S\}$, each taking values from the domain $\mathcal{X}_i$. A relation $\mathcal{R}$ over S is simply a subset $\mathcal{R} \subseteq \prod_{i \in S} \mathcal{X}_i$. The index set S is called the *scope* of $\mathcal{R}$. We make use of two standard relational operators, namely, projection and join. The projection of a relation $\mathcal{R}$ onto the index set $V \subseteq S$ is denoted by $\pi_V \mathcal{R}$, and the join of $\mathcal{R}_1$ and $\mathcal{R}_2$ is denoted by $\mathcal{R}_1 \bowtie \mathcal{R}_2$.

Simple Temporal Networks

An STN is formally represented by a set of variables $\{x_i | \, i \in S\}$, representing time-points, with domain $\mathcal{X}_i = \mathbb{R}^1$; a set of interval unary constraints $\mathcal{T}_i \subset \mathbb{R}$ where $\mathcal{T}_i = \{x_i \mid x_i \in [a_i, b_i]\}$; and a set of binary constraints $\mathcal{T}_{ij} \subset \mathbb{R}^2$, $i < j$ where $\mathcal{T}_{ij} = \{(x_i, x_j) \mid x_i - x_j \leq -a_{ij},\ x_j - x_i \leq b_{ij}\}$.[2] Generally, we assume that there are N variables, so $S = \{1, \ldots, N\}$. Given an STN $\mathcal{T}$, its *constraint graph* G is an undirected graph with vertices representing the variables; an edge links x_i and x_j iff the binary constraint $\mathcal{T}_{ij}$ exists.

We differentiate the constraints represented by $\mathcal{T}$ and the relation they imply, denoted by $sol(\mathcal{T})$ and called the *solution set* of the STN. By definition, $sol(\mathcal{T})$ is a relation with scope S that is the join of all the unary and binary constraints $\mathcal{T}_i$ and $\mathcal{T}_{ij}$. Thus every solution in $sol(\mathcal{T})$ is an assignment of values to time-points such that all constraints are satisfied. Relational operators such as join and projection can be applied on the solution set with the usual set semantics. An STN is *consistent* iff its solution set is non-empty. Two STNs are equivalent (denoted $\mathcal{T}_1 \equiv \mathcal{T}_2$) iff their solution sets are the same, while two STNs are equal (denoted $\mathcal{T}_1 = \mathcal{T}_2$) iff they contain exactly the same set of constraints.

We introduce some useful operators that operate directly on $\mathcal{T}$. Let V be a subset of the variables. The subnetwork of $\mathcal{T}$ restricted to V, denoted by $\mathcal{T}_V$, is the STN with scope V and constraints $\mathcal{T}_i, \mathcal{T}_{ij}$ for $i, j \in V$. Any solution of $\mathcal{T}$ of course will satisfy the constraints of $\mathcal{T}_V$, so $\pi_V sol(\mathcal{T}) \subseteq sol(\mathcal{T}_V)$. When $\pi_V sol(\mathcal{T}) = sol(\mathcal{T}_V)$, the STN is said to be *locally minimal* on V. We next consider two STNs with different scopes S_1, S_2 and constraint graphs G_1, G_2. The *join* of $\mathcal{T}_1$ and $\mathcal{T}_2$, denoted by $\mathcal{T} = \mathcal{T}_1 \wedge \mathcal{T}_2$, is the STN with scope $S_1 \cup S_2$ and constraint graph being the superimposition of G_1 and G_2. All constraints in G_1 (but not in G_2) and in G_2 (but not in G_1) are taken from $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively, and all constraints in both G_1 and G_2 are the pairwise intersection between constraints of $\mathcal{T}_1$ and $\mathcal{T}_2$. It is straightforward to show that $sol(\mathcal{T}_1 \wedge \mathcal{T}_2) = sol(\mathcal{T}_1) \bowtie sol(\mathcal{T}_2)$.

An STN $\mathcal{T}$ has an *equivalent* minimal network representation $\mathcal{T}^{min}$ whose constraints satisfy $\mathcal{T}_i^{min} = \pi_{\{i\}} sol(\mathcal{T})$, and $\mathcal{T}_{ij}^{min} = \pi_{\{i,j\}} sol(\mathcal{T})$ for all $i < j$. Hence, $\mathcal{T}^{min}$ is locally minimal on $\{i\}$ for all i, and on $\{i, j\}$ for all pairs (i, j). The constraint graph of $\mathcal{T}^{min}$ is thus a complete graph. Further, it has been shown that STNs are also *binary-decomposable* (Dechter *et al.* 1991), i.e., for every subset of variables V, the projection $\pi_V sol(\mathcal{T})$ is expressible as a binary constraint network. Further still, the minimal network of $\pi_V sol(\mathcal{T})$ is precisely $\mathcal{T}_V^{min}$, the minimal network of $\mathcal{T}$, restricted to V. Thus for every V, $\pi_V sol(\mathcal{T}) = sol(\mathcal{T}_V^{min})$. The *minimalization* operation to compute $\mathcal{T}^{min}$ is the principal inference task for STNs.

A weaker notion, the *partial minimal network*, is denoted by $\mathcal{T}^{pmin}$ and defined by the set of constraints $\mathcal{T}_i^{pmin} = \pi_{\{i\}} sol(\mathcal{T})$, and $\mathcal{T}_{ij}^{pmin} = \pi_{\{i,j\}} sol(\mathcal{T})$ for all $i < j$ and $(i, j) \in edges(G)$. The partial minimal network thus shares the same constraint graph G with the original network, and

[1] While one can consider STNs with discrete domains, we focus on the more difficult case of continuous domains. The theory in this paper can be readily specialized to the discrete case, and the algorithm we present operates effectively for either case.

[2] In some representations, such as the one used by (Xu and Choueiry 2003), unary domain constraints are modeled as binary relations to a distinguished *temporal reference* time-point, denoted TR, which marks the start of time; thus, without loss of generality, all constraints may be taken to have the binary form.

can be obtained from the minimal network $\mathcal{T}^{min}$ by removing all binary constraints on edges that are not present in G. If the constraint graph of $\mathcal{T}$ is triangulated, given $\mathcal{T}^{pmin}$, every solution to $\mathcal{T}$ can be constructed backtrack-free.

Variable and Clustering-Tree Elimination

Complementary to methods that solve a CSP based on the iteration of narrowing operators, such as PC-1 and △STP (which can be seen as AC-3 operating over triangles (Xu and Choueiry 2003)), an alternative method for the general CSP is called *adaptive consistency* or *variable elimination* (Dechter 2003). Given a general (binary) CSP $\mathcal{T}$ with scope S, consider two sets of variables (called *clusters*), W and V that together cover the constraint graph G (this means $W \cup V = S$ and every edge in G belongs entirely in W or in V). If every path between W and V in G passes through $W \cap V$, we say that the two clusters are separated by their *separator* $W \cap V$, denoted by $sep(W, V)$. It is then possible to project onto W as follows:

$$\pi_W sol(\mathcal{T}) = sol(\mathcal{T}_W) \bowtie m(V, W) \tag{1}$$

where $m(V, W) = \pi_{sep(V,W)} sol(\mathcal{T}_V)$ is the *message* from V to W. This operation effectively 'eliminates' all variables in $V - W$. By eliminating the variables in a given order, we can obtain the minimal constraints. The complexity of this method thus hinges on the representation and the calculation of the messages. Because of the difficulty in representing the messages for variables with continuous domain, this idea has not been applied to STNs whose domains are inherently continuous (Dechter 2003, page 357).

Cluster-Tree Elimination is a generalized variable elimination method for computing the partial minimal network $\mathcal{T}^{pmin}$ for a triangulated constraint graph (Dechter 2003). The algorithm works by decomposing the triangulated graph G into a *join-tree* (also known as *junction-tree*) over a set of variables clusters $\{V_1, \ldots, V_K\}$ that cover the original graph G. The vertices $V_1, \ldots, V_K$ of a join-tree have the property that, for every tuple (i, j, k) such that j lies on the path between (i, k), $V_i \cap V_k \subset V_j$. Once a join-tree is constructed, messages can be passed asynchronously among the clusters along the edge of the join-tree. After two messages have been passed, one in each direction, on every edge of the join-tree, the resulting network can be shown to be locally minimal on every cluster V_i. Solving the local networks at each cluster then yields all the minimal constraints needed for the partial minimal network.

STN Propagation

Variable Elimination for STNs

We now focus on the STN. First observe that, since every STN is binary decomposable, its projection onto an arbitrary subset of variables can be computed and represented by the minimal network. Applying this to the calculation of the message in variable elimination yields

$$m(V, W) = \pi_{sep(V,W)} \left(sol(\mathcal{T}_V)\right) = sol\left((\mathcal{T}_V)^{min}_{sep(V,W)}\right) \tag{2}$$

The message thus can be represented compactly and conveniently as the STN $(\mathcal{T}_V)^{min}_{sep(V,W)}$, which is simply the minimal network of $\mathcal{T}_V$, restricted to the separator set. We call this the *message STN*, denoted by $\mu(V, W)$. Using this compact representation of the messages, we immediately obtain an efficient variable elimination procedure:

Theorem 1. *Consider an STN $\mathcal{T}$ and let V, W be two clusters of variables that cover the constraint graph G. Suppose that V and W are separated by $sep(V, W) = V \cap W$. Then the projection of $\mathcal{T}$ onto W, $\pi_W sol(\mathcal{T})$, can be represented by the solution of the STN*

$$(\mathcal{T}_W \wedge \mu(V, W)) \tag{3}$$

Proof. Substitute (2) into (1) and use $sol(\mathcal{T}_1 \wedge \mathcal{T}_2) = sol(\mathcal{T}_1) \bowtie sol(\mathcal{T}_2)$. □

The above expression involves only a minimalization operation on $\mathcal{T}_V$ and a simple restriction to the separator set. Any of the STN solvers described earlier that produces the minimal network can be used for the former operation.[3] The constraint graph of the STN $(\mathcal{T}_V)^{min}_{sep(V,W)}$ is a complete graph over $sep(V, W)$ (because minimalization creates a complete constraint graph). So unless $\mathcal{T}$ is also complete over $sep(V, W)$, the operation in (3) will introduce new edges to the original constraint graph of $\mathcal{T}$.

Message Passing for STNs

Once we have a compact representation of the messages, the cluster-tree elimination algorithm can be applied immediately to compute the partial minimal network $\mathcal{T}^{pmin}$. Let us assume that we are given a join-tree $\mathcal{J}$ over a set of clusters $V_1, \ldots, V_K$ that cover the original constraint graph G. In some cases, such a join-tree can be found from the structure of G, as in the case of sibling-restricted STNs, or it can be found by first triangulating G and then extracting the set of maximal cliques of the resulting triangulated graph.

The standard message passing scheme used in cluster-tree elimination (Dechter 2003) computes a message from a cluster V_i to a neighbouring cluster V_j, using the messages V_i received from its other neighbours. Representing the message from V_i to V_j by the message STN $\mu^{i,j}$, we obtain

$$\mu^{i,j} = \left[\mathcal{T}_{V_i} \wedge \left(\bigwedge_{k \in neighbour(i), k \neq j} \mu^{k,i}\right)\right]^{min}_{sep(V_i, V_j)}$$

To compute each message requires a minimalization operation over one cluster. After computing all such messages, we must go through each cluster and minimalize each one (with the neighbour messages included). Since there are $2(K - 1)$ messages to compute, we need a total of $2(K - 1) + K = 3K - 2$ minimalization operations. At the end, we obtain the minimal domain $\mathcal{T}_i^{min}$ for all i, but only $\mathcal{T}_{ij}^{min}$ for those (i, j) that belong to the same cluster. Since any edge in G must belong to one of the clusters, we obtain the partial minimal network.

[3] PPC-based methods such as △STP can also be used; however, the subgraph in the separator must be complete, and the subgraph in V must be triangulated.

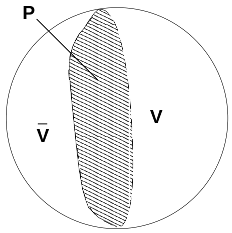

Figure 1: Illustration of the separator set P.

An Improved Propagation Scheme

We can exploit specific properties of STN to make the message passing scheme more efficient. In the above, one minimalization operation is needed for each message. However, binary decomposability of STNs implies that all possible projections of $\mathcal{T}$ onto an arbitrary subset of its variables can be computed in a single minimalization operation. Thus we can rearrange the order of message passing so that one minimalization operation can compute multiple messages. Furthermore, as we show below, it is possible to eliminate the need to store the messages all together.

First, given the join-tree $\mathcal{J}$, a cluster ordering $V_1, \ldots, V_K$ is termed *valid* if whenever V_j lies on the path from V_i to V_K then $j \geq i$. Such a valid ordering can easily be created by choosing an arbitrary cluster to be V_K and treating it as the *root* of $\mathcal{J}$. A valid cluster ordering then lists clusters that are further away from the root first, and root cluster last.

Next, we define a simple operation on STNs, termed *local minimalization*, which takes a subnetwork and replaces it with its minimal network. To be precise, given a subset of variables V, a local minimalization on V returns the STN $lmin(\mathcal{T}, V) = \mathcal{T} \wedge (\mathcal{T}_V)^{min}$. Since $(\mathcal{T}_V)^{min} \equiv \mathcal{T}_V$, it is trivial to see $lmin(\mathcal{T}, V) \equiv \mathcal{T}$ (since, if T_1 is a sub-STP of T_2, then $T_1 \wedge T_2 \equiv T_2$). Note that this operation makes the constraint subgraph in V complete.

Lemma 2. *Let $\mathcal{T}' = lmin(\mathcal{T}, V)$. If $\mathcal{T}$ is locally minimal on some $W \subseteq S$ then so is $\mathcal{T}'$.*

Proof. Observe that $\mathcal{T}'$ constraints are tighter, so $\pi_W sol(\mathcal{T}) = \pi_W sol(\mathcal{T}') \subseteq sol(\mathcal{T}_W^{'}) \subseteq sol(\mathcal{T}_W)$. So if $\pi_W sol(\mathcal{T}) = sol(\mathcal{T}_W)$ then we must have equality, which implies that $\mathcal{T}^{'}$ is also locally minimal on W. $\square$

Lemma 3. *For any $P \subset V$ such that P separates $V - P$ and $\bar{V} = S - V$, $\mathcal{T}'$ is locally minimal on $\bar{V} \cup P$.*

Proof. For an illustration of the sets V, $\bar{V}$ and P, see Figure 1. Let $W = \bar{V} \cup P$, so that $P = sep(V, W)$. Now by Theorem 1, $\pi_W sol(\mathcal{T}) = sol\left((\mathcal{T}_W) \wedge (\mathcal{T}_V)_P^{min}\right)$. In this equality, the LHS is the same as $\pi_W sol(\mathcal{T}')$. In the RHS, $(\mathcal{T}_W) \wedge (\mathcal{T}_V)_P^{min}$ is the same as $\mathcal{T}_W^{'}$. Thus $\pi_W sol(\mathcal{T}') = sol(\mathcal{T}_W^{'})$ so $\mathcal{T}^{'}$ is locally minimal on W. $\square$

Note that as long as we can find a separator P that is a proper subset of V, then $\bar{V} \cup P$ is a proper subset of S. Thus, while the original STN $\mathcal{T}$ is (of course) locally minimal on S, the new STN $\mathcal{T}'$ is locally minimal on a proper subset of S. By repeatedly applying the same kind of operation, we can obtain the minimal constraints.

Algorithm 1 Prop-STP

```
T^0 = T
for i = 1, ..., K-1 do { First Pass }
    T^i = lmin(T^{i-1}, V_i)
end for
for i = K ... 2K-1 do { Second Pass }
    T^i = lmin(T^{i-1}, V_{2K-i})
end for
return T^{2K-1}
```

We call this algorithm *Prop-STP*; pseudo-code is shown in Algorithm 1. We now show that this algorithm returns all the minimal constraints of the original network.

Theorem 4. *At the end of Prop-STP, $\mathcal{T}^{2K-1}$ is locally minimal on every cluster V_k; furthermore, the subnetwork $\mathcal{T}_{V_k}^{2K-1}$ is a minimal network. As a result, $\mathcal{T}^{2K-1}$ is locally minimal on every variable i, and on every pair of variables $\{i, j\}$ for all (i, j) belonging to the same cluster.*

Proof. We proceed by induction on the number of clusters K in the join-tree. The base case is trivial, so assume the theorem holds for any join-tree with $K - 1$ clusters.

Let $S_1 = \bigcup_{i \geq 2} V_i$. Since V_1 must be a leaf node in $\mathcal{J}$, if we let $P = V_1 \cap S_1$ then P separates $V_1 - P$ from $\overline{V}_1$. Thus by Lemma 3, $\mathcal{T}^1$ is locally minimal on $\overline{V}_1 \cup P = S_1$.

Let $\mathcal{T}' = \mathcal{T}_{S_1}^1$ be the subnetwork of $\mathcal{T}^1$ over S_1 and $\mathcal{J}'$ be the join-tree $\mathcal{J}$ minus the cluster V_1. Observe that $\mathcal{J}'$ has $K - 1$ clusters and the ordering $V_2, \ldots, V_K$ is a valid one for $\mathcal{J}'$. Running Prop-STP on $\mathcal{J}'$ would produce the STN $\mathcal{T}_{S_1}^{2K-2}$. Applying the inductive hypothesis, we obtain that $\mathcal{T}_{S_1}^{2K-2}$ is locally minimal on all V_i, $i \geq 2$, as well as on every variable and pair of variables contained within these clusters. Observe that $\mathcal{T}^{2K-2}$ is locally minimal on S_1 (since $\mathcal{T}^1$ is, and using Lemma 2), so the projection of $\mathcal{T}^{2K-2}$ onto V_i $(i \geq 2)$ is the same as the projection of $\mathcal{T}_{S_1}^{2K-2}$. It follows that $\mathcal{T}^{2K-2}$ itself is locally minimal on all V_i, $i \geq 2$, and on all variables and pairs of variables within these clusters, and so is $\mathcal{T}^{2K-1}$ (Lemma 2).

It remains to show that $\mathcal{T}^{2K-1}$ is also locally minimal on V_1 and the subnetwork $\mathcal{T}_{V_1}^{2K-1}$ is a minimal network. By Theorem 1, $\pi_{V_1} sol(\mathcal{T}^{2K-2})$ is the solution of the STN $\mathcal{T}_{V_1}^{2K-2} \wedge (\mathcal{T}_{S_1}^{2K-2})_P^{min}$. Examining the separator P, we see that it must be a subset of some V_i, $i \geq 2$, so the reasoning in the previous paragraph implies that the subnetwork $\mathcal{T}_P^{2K-2}$ is already a minimal one, that is $\mathcal{T}_P^{2K-2} = (\mathcal{T}_{S_1}^{2K-2})_P^{min}$. Thus, $\pi_{V_1} sol(\mathcal{T}^{2K-2})$ is the solution of the STN $\mathcal{T}_{V_1}^{2K-2} \wedge \mathcal{T}_P^{2K-2} = \mathcal{T}_{V_1}^{2K-2}$. Therefore, $\mathcal{T}^{2K-2}$ is locally minimal on V_1, and so is $\mathcal{T}^{2K-1}$. Finally, the last $lmin$ operation is invoked on V_1, so after that $\mathcal{T}^{2K-1}$ is locally minimal on every variable and pair of variables in V_1. $\square$

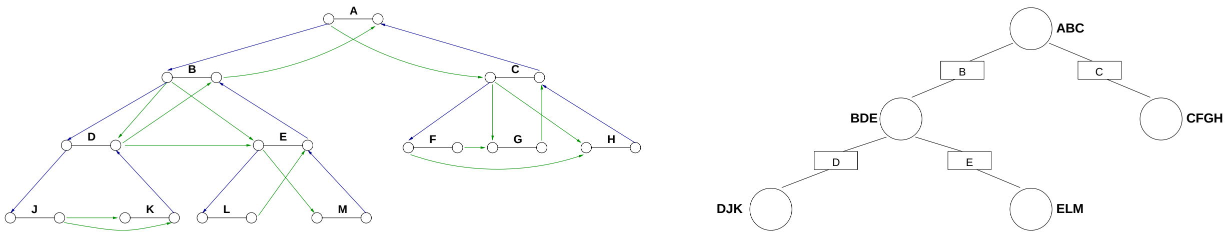

Figure 2: Left: STN; right: join-tree of clusters

Example Figure 2 depicts an STN arising from a hierarchical planning problem. The edges labeled with letters refer to tasks; each such edge models the duration between the start and end time-points of that task. For example, letter A refers to two time-points (variables) A_{start} and A_{end}. The other edges model precedence constraints between parent and child tasks, together with a representative sample of other temporal constraints. The STN has the sibling-restricted property, which provides a simple clustering of the variables to yield the join-tree $\mathcal{J}$ shown on the right. The separators of each cluster are depicted in the rectangular boxes on the edges. On this STN, the first pass of Algorithm 1 processes the clusters in an order such as DJK, ELM, BDE, $CFGH$, ABC. Hence the cluster ABC is the root of the join-tree. The second pass processes $CFGH$, BDE, ELM, DJK. A single $lmin$ operation on BDE in the second pass effectively computes the two messages to DJK and ELM at once. At the end, we obtain the minimal subnetwork for each cluster, and thus the partial minimal network for the whole STN.

Analysis of the New Algorithm

Different variants of Prop-STP can be implemented using different STN subsolvers to perform the $lmin$ operation. For example, using PC-1 (Floyd-Warshall) leads to *Prop-PC1-STP*. Since there are precisely $2K - 1$ $lmin$ operations,[4] each with complexity $\mathcal{O}(w^3)$, the overall complexity of Prop-PC1-STP is $\mathcal{O}(Kw^3)$.

Since Prop-STP effectively completes the constraint graph within each cluster V_i, the resulting global constraint graph is triangulated. In practice, one can triangulate G in the initialization phase, and if so, the constraint graph is already complete within each cluster. In this case, Prop-STP returns the partial minimal network of the triangulated G. Like other algorithms that work on triangulated graph, such as PPC and $\triangle$STP, our Prop-STP does not return the full minimal network. However, this is sufficient for a triangulated graph since every solution can be constructed backtrack-free from its partial minimal network.

It is interesting to examine the behavior of our algorithm when we use $\triangle$STP as the subsolver at each cluster (although this is not likely to result in a performance improvement since the subgraph at each cluster is already complete). In this case, *Prop-$\triangle$-STP* will process triangles in G, one by one, but following the order imposed by our propagation scheme: all triangles within each cluster are processed until stabilized before moving on to triangles in another cluster. As noted by (Xu and Choueiry 2003), the order in which triangles are processed has a crucial effect on the performance of $\triangle$STP. The improved order of triangle processing in our algorithm also agrees with the intuition of the authors of the $\triangle$STP algorithm and others (Choueiry and Wilson 2006).

[4]Improvement can be made by keeping track of which clusters have not been changed after the $lmin$ operations in the first pass, so that in the second pass we need not perform $lmin$ on them again.

Experimental Results

We investigate the performance of Prop-STP on two benchmarks: structured STNs arising from Hierarchical Task Network (HTN) plans, and random unstructured STNs.

Sibling-Restricted STNs

HTN planning assumes a hierarchical flow, with high-level tasks being decomposed progressively into collections of lower-level tasks through the application of matching methods with satisfied preconditions. In a *sibling-restricted* STN, constraints may occur only between parent tasks and their children, and between sibling tasks. This restriction on what STN constraints may exist between plan elements is inherent to HTN planning models; in particular, there is no way in standard HTN representations to specify temporal constraints between tasks in different task networks (Erol *et al.* 1994). The specialized STN solver SR-PC (Yorke-Smith 2005) transverses a tree of sub-STNs that correspond to the decompositions in the HTN plan. Because the STNs thus considered are small, compared to the *global* STN corresponding to the whole plan, the overall amount of work to enforce PC is much less.[5]

Despite its success on benchmark problems from the PASSAT plan authoring system (Myers *et al.* 2002), SR-PC has two drawbacks. First, standard HTN representations have been extended to support coordination between different task networks via *landmark variables* (Castillo *et al.* 2006) that allow synchronization of key events in the plan. SR-PC can accommodate a limited number of such landmark variables and their corresponding constraints, but only

[5]Observe that not only does SR-PC impose no additional limitations on the expressiveness of HTNs, but also that the SR condition guarantees we can propagate on the tree of sub-STNs and lose no information compared to propagating with the whole global STN: hence the algorithm SR-PC is sound and complete.

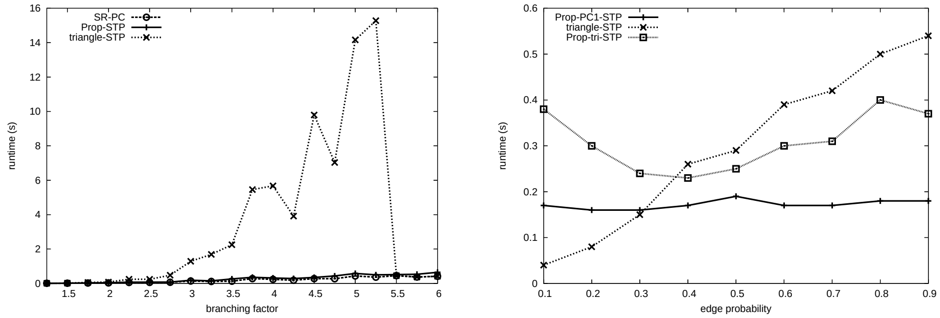

Figure 3: Random SR STNs (left); Random general STNs (right)

awkwardly. Second, although not exhibited in practice, SR-PC has poor and weakly characterized worst-case theoretical complexity.[6]

The small tree-width w of an SR network means that Prop-STP will be particularly efficient for this class of STNs, provided we can find an optimal or near-optimal decomposition of the constraint graph G of the global STN into a join-tree, and a cluster ordering over this tree. Fortunately, there is a natural decomposition based on each parent task $\mathbf{T}$ and its children $\mathbf{T_i}$. Namely, we form into a cluster the start and end time-point variables of task $\mathbf{T}$ and all its $\mathbf{T_i}$, and all temporal relations between them (including those between the children). Figure 2 illustrates this clustering. Since task networks typically comprise a handful of tasks, the size of each cluster is small. Therefore, performing minimalization on each cluster, as done in Prop-STP, is much more efficient than computing the minimal network for the global STN. In general, if we consider a task network represented by a balanced tree with depth d and branching factor f (hence the number of nodes is $\mathcal{O}(f^d)$), the complexity of Prop-STP is $\mathcal{O}(f^2 f^d)$ (the join-tree has $\mathcal{O}(f^{d-1})$ clusters, each with size $\mathcal{O}(f)$). We also note that an SR STN has no articulation points.[7] Because each task in the HTN corresponds to two variables, as can be seen in Figure 2, the constraint graph is biconnected and decomposes only via separator sets of cardinality at least two. This hampers algorithms that seek articulation points, whether explicitly such as *F-W+AP*, or implicitly, such as $\triangle$STP.

Prop-STP also allows us to explain the strong performance of SR-PC in practice, compared to its poor worst-case theoretical complexity. Implicitly, SR-PC works over the natural join-tree for SR STNs. Its recursion through the tree of sub-STNs corresponds to a certain, albeit non-optimal, set of minimalization operations on the clusters. We infer that the number of iterations $\lambda(\Pi)$ of the loop in Algorithm 1 of (Yorke-Smith 2005) is bounded by 1, not 2, explaining the empirical observation that SR-PC does not reconsider sibling sub-STNs once all other siblings have been processed.

Prop-STP on SR STNs

To validate the concept of Prop-STP, we implemented the algorithm within the PASSAT HTN plan authoring system. Figure 3 (left) compares Prop-STP with PC-1 as the subsolver, SR-PC with PC-1, and $\triangle$STP (with triangles queued at end of the queue) on SR STNs.[8] The STNs were extracted from random plans with a uniform tree of tasks, as described in (Yorke-Smith 2005). The figure shows runtime (in seconds) as the mean branching factor of the HTN plan increases (with depth fixed to five), representing random problems with increasing tree-width. It indicates that Prop-STP, which is not restricted to this specialized class of STNs, is as efficient as SR-PC on this specialized class. As expected, $\triangle$STP exhibits a poorer performance, since, with the triangle orderings of (Xu and Choueiry 2003), it is unable to exploit the SR structure to decompose the constraint graph, nor the triangle ordering Prop-STP infers from the join-tree. At the highest branching factors, the STNs are largely over-constrained and thus inconsistent; all three algorithms detect this situation easily.

Prop-STP on Random STNs

We next report preliminary experimental results in comparing the performance of Prop-STP and $\triangle$STP on random STNs. We experimented with two variants of Prop-STP, one using PC-1 and the other using $\triangle$STP as the STN subsolver. The randomly generated STNs are produced by the generator of (Xu and Choueiry 2003).[9] Figure 3 (right) compares the algorithms on STNs with 30 time-points as the number of constraints varies from a sparse to a complete graph (and so the problems from under-constrained, through the critical region, to over-constrained). The results confirm those in the literature that $\triangle$STP is most effective for sparse networks (Shi *et al.* 2004). Prop-PC1-STP is relatively insensitive to the constrainedness, while the performance of Prop-$\triangle$-STP is a blend of the two solvers from which it is composed.

[6] For uniform tree-shaped random SR STNs with a depth of d, a mean branching factor of f, the expected time complexity of SR-PC, using PC-1 as the subsolver, is $\Theta(f^4 f^d)$ (Yorke-Smith 2005).

[7] This is true even when the STN is represented without unary constraints, i.e., there is no temporal reference TR that connects to every time-point. In fact, planning systems such as PASSAT use unary constraints in the STN representation, which precludes any possibility of finding articulation points in the STN.

[8] The experiments were conducted on a Sun Blade 1500 with 2 GB RAM, using Allegro Lisp 6.2; the results average 100 runs.

[9] Both the generator and the $\triangle$STP source code to were kindly made available to us by their authors.

Overall, we observe that PC-1 is somewhat more effective as a subsolver than $\triangle$STP within the Prop-STP framework which may be attributed to the subnetworks (clusters) being complete.

Our current Prop-STP implementation is written in Lisp to allow a fair comparison with the existing Lisp-based implementations of $\triangle$STP and SR-PC, and to allow integration with the PASSAT planning system. Although our reported CPU times agree qualitatively with previous experiments reported in (Xu and Choueiry 2003), on the absolute scale, our CPU runtimes are generally higher, especially for networks with a large number of edges or triangles. We attribute this artifact to the simplistic memory handling of our Lisp environment. We are currently working on the reimplementation of the algorithms in Java to facilitate a direct and meaningful comparison with PC-1 and other STN solvers. Even with the current implementation, however, our *relative* comparison of Prop-STP, $\triangle$STP, and SR-PC is valid.

Conclusion

We have presented a new method, *Prop-STP*, for solving Simple Temporal Networks. In contrast to methods based on graph algorithms or on iteration of narrowing operators, our algorithm is based on an efficient message passing scheme over the join-tree of the network. The complexity of Prop-STP depends on the minimalization operator, i.e. the STN solver used to enforce path consistency on subproblems. Thus consistency and the minimal constraints of an STN (from which solutions can be derived backtrack-free) can be determined with complexity $\mathcal{O}(Kw^3)$ or better, where K is the number of cliques and w is the induced tree-width. For STNs with known and bounded tree-width, Prop-STP thus achieves linear time complexity. The new propagation scheme provides formal explanation of the performance of the existing STN solvers $\triangle$STP and SR-PC. For $\triangle$STP, the new algorithm also provides an efficient triangle ordering based on the join-tree clusters.

Our motivation comes from the sibling-restricted STNs that arise in HTN planning problems. Prop-STP is well-suited to such STNs because these problems (1) have a small tree-width w, and (2) the SR structure leads to an easy way to decompose the network into a join-tree. Prop-STP generalizes the best-known solver, SR-PC, for this class of problems. It avoids the poor worst-case complexity of SR-PC, and it can accommodate landmark variables in SR STNs. At the same time, empirical results validate that Prop-STP retains the efficiency of SR-PC on problems which the latter can solve. For general STNs, our preliminary empirical results on a benchmark of randomly generated networks indicate that Prop-STP outperforms $\triangle$STP, except for the sparest networks. Prop-STP with PC-1 as the subsolver is empirically more effective overall than with $\triangle$STP as the subsolver as the problem size increases.

In our future work, we plan to perform a more thorough empirical evaluation of Prop-STP and other solvers on general STNs, as well as on STNs that are "almost" sibling-restricted. We also plan to explore the practical use of Prop-STP in an HTN planning system with support for landmark variables. Another direction for future work is to employ Prop-STP in incremental STN solving, where time-points and constraints are added or removed incrementally.

Acknowledgment. We gratefully thank Berthe Choueiry, Karen Myers and Bart Peintner for discussions and insights.

References

C. Bliek and D. Sam-Haroud. Path consistency on triangulated constraint graphs. In *Proc. of IJCAI'99*, 1999.

L. Castillo, J. Fdez-Olivares, and F. Palao O. García-Pérez. Efficiently handling temporal knowledge in an HTN planner. In *Proc. of ICAPS'06*, 2006.

B. Y. Choueiry and N. Wilson. Personal communication, February 2006.

T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

R. Dechter and J. Pearl. Tree clustering schemes for constraint-processing. *Artificial Intelligence*, 38(3):353–366, 1989.

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3), 1991.

R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

K. Erol, J. Hendler, and D. Nau. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, University of Maryland, 1994.

U. Kjaerulff. Triangulation of graphs - Algorithms giving small total state space. Technical Report R90-09, Department of Mathematics and Computer Science, Aalborg University, Denmark, 1990.

K. L. Myers, M. W. Tyson, M. J. Wolverton, P. A. Jarvis, T. J. Lee, and M. desJardins. PASSAT: A user-centric planning framework. In *Proc. of the Third Intl. NASA Workshop on Planning and Scheduling for Space*, 2002.

Y. Shi, A. Lal, and B. Y. Choueiry. Evaluating consistency algorithms for temporal metric constraints. In *Proc. of AAAI-04*, 2004.

D. E. Smith, J. Frank, and A. K. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Eng. Review*, 15(1), 2000.

L. Xu and B. Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. In *Proc. of TIME'03*, 2003.

N. Yorke-Smith. Exploiting the structure of hierarchical plans in temporal constraint propagation. In *Proc. of AAAI-05*, 2005.