

Incremental Algorithms for Managing Temporal Constraints*

Alfonso Gerevini
DEA – University of Brescia
via Branze 38
25123 Brescia, Italy
gerevini@bsing.ing.unibs.it

Anna Perini
IRST
38050 Povo
Trento, Italy
perini@irst.itc.it

Francesco Ricci
IRST
38050 Povo
Trento, Italy
ricci@irst.itc.it

Abstract

This paper addresses the problem of efficiently updating a network of temporal constraints when constraints are removed from or added to an existing network. Such processing tasks are important in many AI applications requiring a temporal reasoning module. First we analyze the relationship between shortest-paths algorithms for directed graphs and arc-consistency techniques. Then we focus on a subclass of STP for which we propose new fast incremental algorithms for consistency checking and for maintaining the feasible times of the temporal variables.

1 Introduction

The efficient management of temporal constraints is an important issue in several areas of Artificial Intelligence (AI), including knowledge representation and planning & scheduling. In these areas determining the consistency of a given set of constraints, and computing for each involved temporal variable a “window” of feasible values are essential reasoning tasks. In many AI applications these are “on-line” tasks prompted by the addition (or deletion) of constraints to (from) an existing data base of constraints which is managed by a specialized reasoning module [2, 3].

The work of this paper is based on the STP framework for metric temporal reasoning that was introduced in [6] as a class of binary constraints involving real valued time-point variables. We first investigate the relationship between known shortest-paths algorithms and arc-consistency algorithms for processing the constraints of STP (STP-constraints); then we introduce a subclass of the STP-constraints, that we

termed STP^- , for which we propose new algorithms for incremental consistency checking and for the incremental update of the feasible times of the temporal variables. In an extended version of this paper the interested reader can find a complete description of the algorithms presented, together with some experimental results illustrating the efficiency of our approach [8].

2 Arc-consistency and Graph Algorithms for STP

An STP network $T = (V, C)$ represents a set of constraints C of the form $y - x \in I$, where x and y are point-variables in a set V , and I is an interval in the time domain [6]. STP networks are decomposable [6], and both consistency checking and computing the “minimal network” representation can be performed in $O(n^3)$ time [6, 12]. In [6] it is shown that checking the consistency of a STP network leads to dealing with a shortest-paths problem for a directed graph.

The *distance graph* $G = (V, E)$ associated with a STP network T is a directed weighted graph having the same vertices as T , and a pair of edges (x, y) and (y, x) labeled $-a$ and b respectively, for each constraint $y - x \in [a, b]$ in C [6]. It can be proved that if s is a temporal variable constrained to be the time origin (i.e., $s = 0$), then for each variable x the earliest time and the latest time are $-d^t(x)$ and $d(x)$ respectively, where $d(x)$ and $d^t(x)$ are the shortest-path distances from s to x in G and in the transpose graph $G^t = (V, E^t)$ of G respectively ($E^t = \{(y, x) \in V \times V \mid (x, y) \in E\}$ and the weight of an edge (y, x) in G^t is the same as the weight of (x, y) in G).

Therefore, consistency checking and computing the feasible times can be achieved by using a *single-source* shortest-paths algorithm such as the algorithm by Bellman and Ford (BF) [8]. Furthermore, if the graph is acyclic then a more efficient shortest-paths algorithms specialized for DAGs can be used [4].

*This work has been partially supported by the EspritIII project #6095 CHARADE.

The BF algorithm can be considered a particular arc-consistency algorithm that does not suffer the termination problem, which may affect known algorithms for enforcing arc-consistency in continuous domains CSP. This problem consists in the possibility of generating an arbitrary long succession of edge revisions [5, 10]. In particular, we can state the following theorem (AC-BF(T) is shown in Figure 1).

Theorem 1 *Let $T = (V, C)$ be a simple temporal constraint network, AC-BF(T) enforces arc-consistency on T in $O(|V||C|)$ time. Moreover, T is consistent iff AC-BF(T) returns TRUE, and if T is consistent then AC-BF(T) computes the earliest time and the latest time for each variable in V .*

The procedure “Refine(u, v)” in AC-BF(T), used by traditional arc-consistency algorithms [5, 10], eliminates from the current domain of u those values that are not compatible with the current domain of v according to the constraints between u and v of T .

Remark. Theorem 1 shows that enforcing arc-consistency is an efficient procedure for deciding consistency of an STP, extending to continuous-domain variables a result by Van Hentenryck *et al.* [9] given for a similar set of constraints on discrete-domain variables. Moreover, Theorem 1 extends the application of Faltings’ arc-consistency algorithms for tree-structured constraint networks with continuous labels [7].

3 Incremental Algorithms for STP⁻

We now introduce a subclass of STP, called STP⁻, consisting of the STP-constraints of the form $x_j - x_i \leq a$, where $a \leq 0$. STP⁻ has an appealing feature: the distance graph of a set of STP⁻-constraints can be incrementally managed through efficient algorithms. Concerning the expressiveness of STP⁻, it can be easily shown that this class can express the following constraints:

1. simple unary constraints of the form $x_i \in I$;
2. constraints stating the minimal duration of a temporal interval;
3. deadline constraints of the form $x_i < d$, where d is a real number expressing a certain “date”;
4. assertions of qualitative relations in the Convex Simple Interval Algebra [13];
5. assertions of basic qualitative relations in Allen’s Interval Algebra extended with positive lag times [1].

Before addressing the dynamic management of STP⁻-constraints we need to give some basic background and to introduce a few useful concepts. For a distance graph $G = (V, E)$ with no negative cycles

```

AC-BF( $T$ ):
1  for  $i \leftarrow 1$  to  $|V| - 1$ 
2      do for each edge  $(u, v) \in E \cup E^t$ 
3          do Refine( $u, v$ )
4  for each edge  $(u, v) \in E \cup E^t$ 
5      do if Refine( $u, v$ )
6          then return FALSE
7  return TRUE

```

Figure 1. The algorithm AC-BF

we can compute a shortest-paths tree (s-p tree) \mathcal{T} of G , rooted at a source vertex s [4]. We shall indicate with $\pi(x)$ and with $d(x)$ the predecessor of x in \mathcal{T} and the s-p distance from s to x in G respectively.

The *distance metagraph* of a distance graph is a directed acyclic graph in which sets of equivalent vertices are “collapsed” into a single vertex forming a *metavertex*. If G does not contain cycles with negative length then two vertices x, y are said to be *equivalent* (written as $x \sim y$) if and only if there exists in G a cycle connecting x and y whose length is zero. The distance metagraph of G is indicated with $G^* = (V^*, E^*)$, where E^* is defined as

$$E^* = \{(X, Y) \in V^* \times V^* \mid \exists(x, y) \in E, x \in X, y \in Y\}$$

and each *metavertex* X in V^* represents a set \hat{X} of equivalent vertices of G . Note that the vertices of \hat{X} forms a “strongly connected component” (SCC) of G [4]. The *extensions* $e(X, Y)$ of an edge (X, Y) is defined as

$$e(X, Y) = \{(x, y) \in E \mid x \in X, y \in Y\}$$

The weight $W(X, Y)$ of an edge (X, Y) is the minimum among the weights of the edges in $e(X, Y)$. Finally, we indicate with $[x]$ the vertex X of V^* such that $x \in \hat{X}$ and $x \in V$. In the rest of the paper we shall also use the notation $Rep(X)$ to indicate a vertex $x_r \in \hat{X}$ such that $\pi(X) = [\pi(x_r)]$. This vertex is called the *representative vertex* of X .

The following Lemma has a straightforward proof.

Lemma 1 *Let G be the distance graph of a STP network T . G has no negative cycles iff G^* has no negative cycles. Moreover if G^* has no negative cycles, then for all $X \in V^*$ there exists a vertex x in \hat{X} such that $d(x) = d(X)$ and $\pi(X) = [\pi(x)]$.*

Remark. If G is the distance graph of a consistent STP⁻ network, then Lemma 1 has a very useful consequence. In fact, since G^* is acyclic, its shortest paths can be computed by using a shortest-paths algorithm for DAGs such as DAG-Shortest-Paths given in [4, page 536]. DAG-Shortest-Paths takes $O(|E|)$ time

and space, and thus it is more efficient than the BF algorithm requiring $O(|V||E|)$ time. Moreover, we shall see how Dag-Shortest-Paths can be modified to efficiently support incremental assertions and retraction of temporal constraints.

In the following we shall assume that $G = (V, E)$ is the distance graph of an initial set of STP⁻-constraints which does not contain negative cycles, and that $G^* = (V^*, E^*)$ is the corresponding metagraph. Since G^* is a DAG, it can be topologically ordered. We assume that $\langle X_0, X_1, \dots, X_n \rangle$ is a topological order of the vertices of G^* .

Two main operations can be performed on G : deleting an edge from G and adding an edge to it. When these operations are performed, maintaining the feasible times of a network variable x requires to update $d(x)$ and $d^t(x)$.

3.1 Deleting an Edge

Figure 2 gives an algorithm, called *Delete*, for removing an edge from G updating G^* appropriately. The following Theorem states the correctness of *Delete*.

Theorem 2 *When an edge (x, y) is removed from G $Delete((x, y), G, G^*)$ correctly updates the shortest-path tree and the shortest-path distances of G and G^* .*

Sketch of the proof. Let $[x] = X$ and $[y] = Y$, there are two possible cases: (a) $X = Y$, i.e. in G there is a cycle connecting x and y ; (b) $X \neq Y$. Let us first suppose that $X \neq Y$ (step 1 of *Delete*). The topological order of G^* does not need to be updated. We have two cases to examine: either $\pi(Y) \neq X$ or $\pi(Y) = X$. In the first case the s-p tree is not modified by the deletion of the edge.

Let us suppose that $\pi(Y) = X$. If $w(x, y) > W(X, Y)$, or if there is another edge (x', y') in $e(X, Y)$ such that $w(x', y') = w(x, y)$, then the deletion of (x, y) does not produce any significant change (see steps 2-6 of *Delete*). If $w(x, y) = W(X, Y)$ and (x, y) was the only edge in $e(X, Y)$ with weight $w(x, y)$, then we update the s-p tree and the s-p distances of G^* by running *Increase*, which applies the *Relax* procedure of DAG-Shortest-Paths [4, page 520] to a minimal set of edges.¹

Let us now suppose that $Y = X$, that is we have to remove from G an edge connecting two vertices “collapsed” into the same metavertex. In this case we have to check whether the subgraph $G_X = (\hat{X}, E_X)$ of G such that $E_X = \{(t, z) \in E \mid t, z \in \hat{X}, t \neq x, z \neq y\}$ still forms a single SCC, and if this is not the case we have to compute the possible new SCCs in G_X that

¹For lack of space we omit the specification of this procedure. The interested reader is referred to [8] for a detailed description.

```

Delete( $(x, y), G, G^*$ )
0  Remove the edge  $(x, y)$  from  $G$ 
1  if  $[x] \neq [y]$  then
2    if  $\pi([y]) = [x]$  and  $y = Rep([y])$  then
3      for each  $(u, v)$  in  $e([x], [y])$ 
4        if  $w(u, v) = W([x], [y])$ 
          and  $(u, v) \neq (x, y)$  then
5           $Rep([y]) \leftarrow v$ 
6        return
7       $\pi([y]) \leftarrow NIL$ 
8       $Increase(\{[y]\}, G^*)$ 
9  else  $\langle [x_1], \dots, [x_m] \rangle = L \leftarrow Recompute-Cycles([x])$ 
10 if  $|L| > 1$  then
11   Replace  $[x]$  with  $L$ 
12   shift forward  $m - 1$  positions the vertices
      after  $[x]$  in the previous topological order
      and update their predecessor function  $\pi$ 
13    $j \leftarrow$  index s.t.  $Rep([x]) = x_j$ 
14    $\pi([x_j]) \leftarrow \pi([x])$ 
15    $d([x_j]) \leftarrow d([x])$ 
16    $Rep([x_j]) \leftarrow Rep([x])$ 
17   for  $i \leftarrow j + 1$  to  $m$ 
18      $\pi([x_i]) \leftarrow [p(x_i)]$ 
19      $d([x_i]) \leftarrow d([x])$ 
20      $Rep([x_i]) \leftarrow x_i$ 
21   for  $i \leftarrow 1$  to  $j - 1$ 
22      $\pi([x_i]) \leftarrow NIL$ 
23      $Rep([x_i]) \leftarrow NIL$ 
24    $Increase(\{[x_1], \dots, [x_{j-1}]\}, G^*)$ 

```

Figure 2. The algorithm *Delete*

will yield a set of new metavertices in G^* . This is accomplished by the procedure called *Recompute-Cycles* which is based on two depth-first searches on G_X and G_X^t [4, page 489].¹ The rest of *Delete* updates the topological order of the vertices of G^* and the s-p tree. \square

3.2 Adding an Edge

Figure 3 gives an algorithm, called *Add*, for the incremental addition of new edges to G . The following theorem states the correctness of *Add*.

Theorem 3 *When a new edge (x, y) whose weight is w is added to G , the procedure $Add((x, y), w, G, G^*)$ checks the consistency of the resulting graph, and correctly updates G^* , the corresponding functions π and d , and the topological order of the vertices in G^* .*

Sketch of the proof. If $[x] = [y]$ and (x, y) has negative weight, then the addition of the new edge gives rise to

```

Add-Oriented( $(x, y, w, G^*)$ ):
1  if Relax( $([x], [y])$ ) then
2     $L \leftarrow \{[y]\}$ 
3    for each  $X$  after  $[y]$  in the t.o. of  $G^*$ 
4      insert  $X$  at the end of  $L$ 
5       $m(X) \leftarrow FALSE$ 
6     $m([y]) = TRUE$ 
7    for each  $X$  in  $L$ 
8      if  $m(X)$  then
9        for all  $Z \in Adj(X)$ 
10          $m(Z) \leftarrow Relax(X, Z)$ 

Add( $(x, y, w, G, G^*)$ ):
1  Add the edge  $(x, y)$  to  $G$ 
2  if  $[x] = [y]$  then
3    if  $w < 0$  then
4      return  $FALSE$ 
5    else return  $TRUE$ 
6  else if  $ord([x]) > ord([y])$  then
7    if not Update-TO( $([x], [y]), w, G^*$ )
8      return  $FALSE$ 
9    Add-Oriented( $(x, y, w, G^*)$ )
10 return  $TRUE$ 

```

Figure 3. The algorithm Add

an inconsistency, whereas if it has weight zero G^* then π , d and the topological order (t.o.) do not need to be updated (steps 1-5 of Add).

Let us suppose that $[x] \neq [y]$, we have $[x] = X_i$ and $[y] = X_j$ for some $0 \leq i, j \leq n$. If X_i precedes X_j in the current topological order then the algorithm *Add-Oriented*((x, y, w, G^*)) correctly updates the functions π and d (step 9 of Add). *Add-Oriented*((x, y, w, G^*)) is an incremental shortest-paths algorithm for DAGs, where only the edges leaving from a metavertex X , such that $d(X)$ has been updated, need to be relaxed, and where the t.o. of the vertices is exploited to enhance efficiency [8].

Conversely, if in the current t.o. X_i is after X_j , then the new edge violates it. Thus, the new metagraph either becomes inconsistent (a negative cycle has been introduced), or it is still consistent and a new t.o. can be found (possibly with some metavertrices merged to form a new metavertex). This is attained by the algorithm *Update-TO*($(X_i, X_j), w, G^*$) [8] which generalizes an incremental algorithm proposed in [11] for DAGs where edges have no weights.¹ The last part of Add runs *Add-Oriented*($([x], [y]), w, G^*$) to update π and d . \square

4 Conclusion

We have proposed a collection of techniques aimed to serve as a base for building efficient temporal reasoning tools. The first part of the paper analyzes the relationship between known shortest-paths algorithms for directed weighted graphs and arc-consistency techniques for STP networks. The second part investigates a subclass of STP called STP⁻ whose constraints are of the form $y - x \leq a$ ($a \leq 0$). For this class of constraints we have proposed new fast incremental algorithms for consistency checking and for maintaining the feasible times of the temporal variables. Our method is based on shortest-paths techniques for directed graphs and on a distance metagraph whose vertices are incrementally maintained topologically ordered. An experimental comparison of the proposed algorithms with the (non-incremental) BF algorithm showed that drastic CPU-time reductions can be obtained [8].²

References

- [1] J. F. Allen. Maintaining knowledge about temporal intervals. *Comm. of ACM*, 26(11):832–843, 1983.
- [2] C. E. Bell and A. Tate. Use and justification of algorithms for managing temporal knowledge in o-plan. Tec. Rep. AIAI-TR-6, Edinburgh, U.K., 1985.
- [3] R. Cervoni, A. Cesta, and A. Oddi. Managing dynamic temporal constraint networks. In K. Hammond, editor, *Proc. of the 2nd Int. Conf. on Artificial Intelligence Planning Systems*, 196–201, 1994.
- [4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. The MIT Press, 1992.
- [5] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.
- [6] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49, 1991.
- [7] B. Faltings. Arc-consistency for continuous variables. *Artificial Intelligence*, 65:363–376, 1994.
- [8] A. Gerevini, A. Perini, and F. Ricci. Incremental algorithms for managing temporal constraints. Tec. Rep. IRST-9605-07, IRST, 1996.
- [9] P. V. Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57, 1992.
- [10] A. K. Mackworth. Consistency in network of relations. *Artificial Intelligence*, 8:99–118, 1977.
- [11] A. Marchetti-Spaccamela, U. Nanni, and H. Rohert. On-line graph algorithms for incremental compilation. *Lecture Notes in Comp. Sci.*, 740:113–151, 1993.
- [12] U. Montanari. Networks of constraints: fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.
- [13] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58, 1992.

²For lack of space we omit these experimental results which are reported in [8].