A Practical Temporal Constraint Management System for Real-Time Applications

Luke Hunsberger [1]

Abstract. A temporal constraint management system (TCMS) is a temporal network together with algorithms for managing the constraints in that network over time. This paper presents a practical TCMS, called MYSYSTEM, that efficiently handles the propagation of the kinds of temporal constraints commonly found in real-time applications, while providing constant-time access to "all-pairs, shortest-path" information that is extremely useful in many applications. The temporal network in MYSYSTEM includes special time-points for dealing with the passage of time and eliminating the need for certain common forms of constraint propagation. The constraint propagation algorithm in MYSYSTEM maintains a restricted set of entries in the associated all-pairs, shortest-path matrix by incrementally propagating changes to the network either from adding a new constraint or strengthening, weakening or deleting an existing constraint. The paper presents empirical evidence to support the claim that MYSYSTEM is scalable to real-time planning, scheduling and acting applications.

1 Introduction

A *Simple Temporal Network* (STN) is a pair, $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set of time-point variables (or time-points) and $\mathcal{C}$ is a set of temporal constraints, each having the form: $t_j - t_i \leq \delta$, for some $t_i, t_j \in \mathcal{T}$ and some real number δ [3]. In this paper, we let $n = |\mathcal{T}|$ and $m = |\mathcal{C}|$. A *solution* to an STN is a set of real-valued assignments to the variables in $\mathcal{T}$ that satisfy all of the constraints in $\mathcal{C}$. An STN is called *consistent* if it has at least one solution.

Each STN, $(\mathcal{T}, \mathcal{C})$, has a corresponding graph, $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, where the nodes of the graph are the time-points in $\mathcal{T}$, and the edges of the graph correspond one-to-one with the constraints in $\mathcal{C}$. In particular, for each constraint, $t_j - t_i \leq \delta$, in $\mathcal{C}$, there is an edge from t_i to t_j with weight δ in $\mathcal{E}$. In this paper, we let k be the maximum number of edges incident to any node in the graph. An STN is consistent if and only if its corresponding graph has no *negative cycles* (i.e., loops with negative path-length) [3].

Most STNs include a special time-point—called the *zero time-point* (or Z)—whose value is fixed at 0. Temporal constraints involving Z are equivalent to unary constraints. For example, $Z - t_i \leq \delta_1$ is equivalent to the lower-bound constraint, $-\delta_1 \leq t_i$; and $t_j - Z \leq \delta_2$ is equivalent to the upper-bound constraint, $t_j \leq \delta_2$.

The *distance matrix* for an STN is an n-by-n matrix, $\mathcal{D}$, such that $\mathcal{D}(t_i, t_j)$ equals the length of the shortest path from t_i to t_j in the corresponding graph, $\mathcal{G}$. Thus, $\mathcal{D}$ is the *all-pairs, shortest-path* (APSP) matrix for $\mathcal{G}$. If there is no path from t_i to t_j, then $\mathcal{D}(t_i, t_j) = \infty$.

Changing an STN over Time. An STN typically acquires new time-points and constraints over time. Algorithms that incrementally propagate changes to the STN in response to adding a new constraint or strengthening an existing constraint are called *incremental* algorithms. Algorithms that propagate changes to the STN in response to weakening or deleting a constraint already in the network are called *decremental* algorithms. Algorithms that are both incremental and decremental are called *fully dynamic*. Decremental algorithms have higher time complexity than their incremental counterparts [16, 12].

Executing Time-Points. In most applications, the starting and ending times of tasks are represented by time-points in a temporal network. When the task is begun—say, at time K—its starting time-point, t_s, is fixed to the value K, by inserting the constraints, $K \leq t_s \leq K$ (i.e., $Z - t_s \leq -K$ and $t_s - Z \leq K$). We say that t_s has been *executed* at time K. Similarly, when the task is completed—say, at time L—its ending point, t_e, is fixed to the value L.
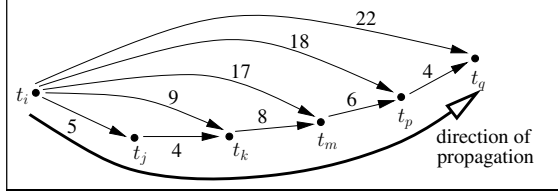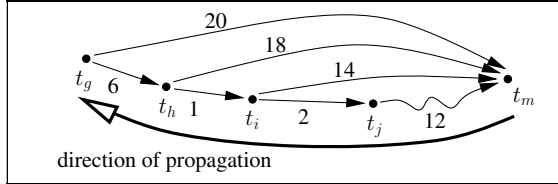
Cesta and Oddi's Algorithm. Cesta and Oddi [2] presented a fully dynamic algorithm for propagating changes to an STN. The algorithm does not maintain the entire distance matrix; instead, it maintains only enough entries to verify the consistency of the network. In particular, for each time-point $t \in \mathcal{T}$, it only maintains entries of the form, $\mathcal{D}(Z, t)$ and $\mathcal{D}(t, Z)$. Thus, the space requirements are $O(n)$. The incremental portion of the algorithm, which is a variation of the Bellman-Ford algorithm, has time complexity $O(nm)$. The decremental portion of the algorithm first determines which entries might be affected by the change to the network and then runs the incremental portion on that part of the network. Since their algorithm does not maintain the full distance matrix, it can only discover negative cycles during the process of constraint propagation. Furthermore, answering distance matrix queries for entries other than those involving Z requires $O(kn)$ time, instead of the constant look-up time that is afforded by having the full distance matrix.

Maintaining the Full Distance Matrix. Maintaining an up-to-date distance matrix requires $O(n^2)$ space and additional constraint propagation; however, it has the following important advantages. First, it provides constant-time lookup for distance-matrix entries, which facilitates the use of multi-agent coordination algorithms (e.g., temporal decoupling algorithms [9]). Second, before adding a new constraint (or strengthening an existing constraint) the consistency of the resulting network can be determined by constant-time lookup— *in advance of any constraint propagation* [11].

Researchers have developed fully dynamic algorithms for maintaining distance matrices [7, 5, 16, 4, 12]. Although these algorithms have attractive time complexities, they restrict the kinds of constraints that can populate a network and, thus, are inappropriate for many applications. Others have presented algorithms making fewer restrictions, but exhibiting poorer performance [13, 6].

The INCR2004 Algorithm. The author recently presented a practical incremental algorithm for maintaining the full distance ma-

---

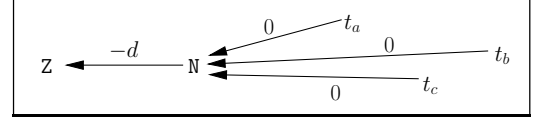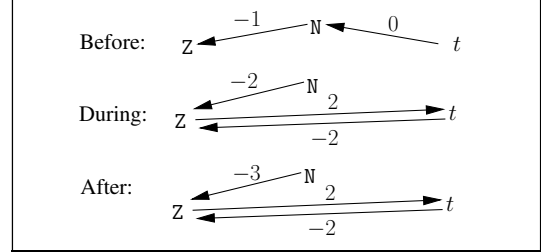[1] Vassar College, Poughkeepsie, NY, USA, hunsberg@cs.vassar.edu

Figure 1.   The *PropFwd* phase of the incremental algorithm

Figure 2.   The *PropBkwd* phase of the incremental algorithm

Figure 3.   The now time-point in an ASTN

Figure 4.   The execution of the time-point t at time 2

trix [8]. For ease of exposition, we shall refer to that algorithm as the INCR2004 algorithm. That algorithm reduces the size of the network by collapsing all *rigid components* down to a single time-point.[2] The INCR2004 algorithm also reduces constraint propagation by propagating only along *undominated* edges.[3] The undominated edges are stored in hash tables. In particular, for each time-point t, $Precs(t)$ is a hash table containing the undominated edges coming in to t; and $Succs(t)$ contains the undominated edges going out from t. The high-level structure of the algorithm, which is based on work by several others [12, 13, 6], has two phases, called *PropFwd* and *PropBkwd*. The algorithm has time complexity $O(k\Delta)$, where Δ is the number of entries of $\mathcal{D}$ that actually need to be changed [12].

The PropFwd *Phase.*   Suppose a new (or stronger) constraint, $t_j - t_i \leq \delta$, is added to the network. Fig. 1 illustrates the *PropFwd* phase, in which changes to distance matrix entries of the form, $\mathcal{D}(t_i, t)$, are propagated by following the successors of t_j. In the figure, decreasing the weight of the edge, $t_i t_j$, from 5 to 2 requires decreasing $\mathcal{D}(t_i, t_k)$ from 9 to 6, and decreasing $\mathcal{D}(t_i, t_m)$ from 17 to 14. Since $\mathcal{D}(t_i, t_p)$ does not need to be changed, forward propagation stops at that point.[4] During the *PropFwd* phase, each time-point, t, for which $\mathcal{D}(t_i, t)$ changed is collected in a hash-table, *AffectedTPs*.

The PropBkwd *Phase.*   Fig. 2 illustrates the *PropBkwd* phase of the INCR2004 algorithm. For each t_m in *AffectedTPs* collected during the *PropFwd* phase, the predecessors of t_i are followed, potentially leading to changes in entries of the form, $\mathcal{D}(t, t_m)$. For example, in the figure, the entry $\mathcal{D}(t_i, t_m)$ had been reduced from 17 to 14 during the first phase. Its new value, requires reducing $\mathcal{D}(t_h, t_m)$ from 18 to 15. However, since $\mathcal{D}(t_g, t_m)$ does not need to be changed, backward propagation stops at that point.

Augmented STNs.   An *Augmented STN* (ASTN) is an STN that has been augmented to include a special time-point, N, which represents the current time (i.e., "now") [11]. Representing the *now* time-point enables the network to *explicitly* handle the passage of time

and the execution of time-points. The passage of time is handled by including a single edge from N to Z, with weight $-d$, representing the lower-bound constraint, $d \leq$ N. This edge, as illustrated in Fig. 3, is the only outgoing edge from the now time-point. As time passes, the value of d increases (i.e., the constraint involving Z and N grows stronger). Since the time-complexity of strengthening a constraint is lower than that of weakening or deleting constraints, this way of dealing with the passage of time is computationally attractive.

In an ASTN, each unexecuted time-point, t, is constrained to occur at or after *now*—represented by an edge from t to N with weight 0. Fig. 3 illustrates these kinds of edges, which are the only incoming edges to the now time-point. When t is executed, the edge from t to N is deleted, and two edges between t and Z are inserted to fix t's value. Fig. 4 provides "before", "during" and "after" snapshots of a network in which t is executed at time 2. In the "before" snapshot, the current time is 1, and t is constrained to occur at or after that time. In the middle snapshot, t has been executed at time 2 (i.e., the edge from t to N has been deleted, and a pair of edges between t and Z have been inserted, fixing the value of t to 2). In the bottom snapshot, the current time has advanced to 3, but that has no effect on t.

For an ASTN, the distance matrix entry, $\mathcal{D}(\text{Z}, \text{N})$, can be interpreted as a kind of deadline [11]. In particular, if some time-point is not executed at or before this deadline, then the network is certain to become inconsistent—because the passage of time (i.e., the increased value of d on the edge from N to Z) will eventually generate a negative cycle. The potential inconsistency can be averted by executing one or more time-points, thereby deleting constraints involving N and increasing the value of $\mathcal{D}(\text{Z}, \text{N})$.

2   Desiderata

The main goal for the work described in this paper is to provide a temporal constraint management system that can serve as the basis for a temporal reasoning module in real-time planning, scheduling and acting applications, including multi-agent systems involving the coordination of temporally dependent, inter-agent activities. This high-level goal consists of the following subsidiary goals:

- To maintain constant-time access to all distance-matrix entries
- To reduce space requirements for the distance matrix (or any other auxiliary data structures)
- To reduce the need for constraint propagation

---

[2]   A rigid component is a set of time-points in which the temporal distance between each pair of time-points is constrained to be some fixed value. Other researchers have described collapsing rigid components [17, 7].

[3]   A constraint is called *undominated* if removing it from the network would necessarily require updating the distance matrix. In contrast, removing a *dominated* constraint from the network would leave the distance matrix unchanged. The algorithm takes advantage of the fact that dominated constraints are easy to detect in networks with no rigid components [10].

[4]   For expositional simplicity, Fig. 1 shows only one branch of the sub-tree rooted at t_j. The *PropFwd* phase normally explores multiple branches of that sub-tree. Similar remarks apply to the *PropBkwd* phase.
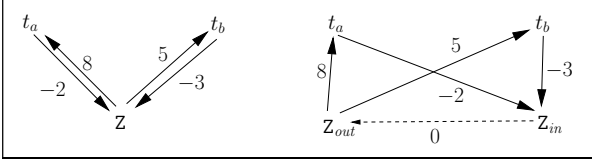
Figure 5. Replacing the zero time-point by a pair of time-points

- To include a fully dynamic constraint propagation algorithm that is scalable to real-time applications

Constant-time access to distance-matrix entries facilitates multi-agent coordination algorithms (e.g., temporal decoupling [9]). Reducing space requirements for the distance matrix implies not explicitly representing every distance-matrix entry, while maintaining constant-time access. Reducing the need for constraint propagation makes the fully dynamic algorithm computationally palatable. "Scalable" means that the resulting TCMS is practical for applications involving hundreds, or even thousands of time-points.

3 Approach

This paper presents a TCMS called MYSYSTEM that meets the desiderata listed above. In MYSYSTEM:

- The *now time-point*, $\mathbb{N}$, is explicitly represented (as in ASTNs).
- The *zero time-point,* $\mathbb{Z}$, is replaced by a pair of time-points, $\mathbb{Z}_{in}$ and $\mathbb{Z}_{out}$, thereby eliminating propagation through $\mathbb{Z}$, and reducing the number of distance matrix entries needing to be computed.
- Since the portion of the distance-matrix that is actually computed is typically quite small, the values are stored in a hash table, instead of a two-dimensional array.
- The incremental algorithm is essentially the same as the INCR2004 algorithm, except that rigid components and dominated constraints are handled differently.
- A new decremental algorithm is provided that manipulates the same data structures as the incremental algorithm. The algorithm, which draws on ideas from other researchers [4, 13], is not the fastest possible, but requires only minor auxiliary data structures.
- Executed time-points are effectively removed from the network.

Replacing the Zero Time-Point by a Pair of Time-Points. In real-world applications, the starting and ending times of tasks are typically subject to a variety of unary constraints—that is, constraints involving the zero time-point, $\mathbb{Z}$. As a result, while the maximum number of edges incident on any other time-point might be, say, ten, the number of edges incident on $\mathbb{Z}$ can be $O(n)$. Thus, a great deal of the constraint propagation needed to fully populate the distance matrix is due to constraints involving $\mathbb{Z}$.

To eliminate constraint propagation through $\mathbb{Z}$, the temporal network in MYSYSTEM replaces $\mathbb{Z}$ by a *pair* of time-points, $\mathbb{Z}_{in}$ and $\mathbb{Z}_{out}$.[5] In particular, as illustrated in Fig. 5, $\mathbb{Z}_{in}$ is the destination for all edges that would normally point to $\mathbb{Z}$, and $\mathbb{Z}_{out}$ is the source of all edges that would normally emanate from $\mathbb{Z}$.

Now, adding an edge from $\mathbb{Z}_{in}$ to $\mathbb{Z}_{out}$ with weight 0 (shown as a dashed arrow in the figure) would make the two networks in Fig. 5 equivalent; however, such an edge is purposely left out of the network in MYSYSTEM. This seemingly minor change eliminates propagation through $\mathbb{Z}$; thus, it dramatically reduces the amount of computation required to maintain the distance matrix. At the same time,

MYSYSTEM retains the property of having constant-time access to all distance-matrix entries. To see this, suppose $\mathcal{A}$ is a standard ASTN and $\mathcal{A}'$ is the same as $\mathcal{A}$, except that the zero time-point has been replaced by $\mathbb{Z}_{in}$ and $\mathbb{Z}_{out}$, as described above. Because the edge from $\mathbb{Z}_{in}$ to $\mathbb{Z}_{out}$ is left out of $\mathcal{A}'$, the distance matrices, $\mathcal{D}$ and $\mathcal{D}'$, are typically quite different. However, the relationship between their corresponding entries is simple. In particular, for any $t_i, t_j \in \mathcal{T} \backslash \{\mathbb{Z}\}$:[6]

- $\mathcal{D}(t_i, \mathbb{Z}) = \mathcal{D}'(t_i, \mathbb{Z}_{in})$
- $\mathcal{D}(\mathbb{Z}, t_j) = \mathcal{D}'(\mathbb{Z}_{out}, t_j)$
- $\mathcal{D}(t_i, t_j) = \min\{\mathcal{D}'(t_i, \mathbb{Z}_{in}) + \mathcal{D}'(\mathbb{Z}_{out}, t_j), \ \mathcal{D}'(t_i, t_j)\}$

The last equality can be glossed as: "The shortest path from t_i to t_j either involves the zero time-point or it doesn't." In this way, although $\mathcal{D}'$ typically contains far fewer finite entries than $\mathcal{D}$, it can be used to fetch the value of any entry in $\mathcal{D}(t_i, t_j)$ in constant time.

The Distance-Matrix Hash Table. Due to the use of $\mathbb{Z}_{in}$ and $\mathbb{Z}_{out}$, the constraint propagation algorithms in MYSYSTEM typically need to compute only a small fraction of the $O(n^2)$ entries in the distance matrix, $\mathcal{D}'$. Thus, to save space, a hash table is used to store only those entries that are actually computed. Any entry, $\mathcal{D}'(t_i, t_j)$, that has not been stored in the hash table is taken to be infinity, representing that there is no path from t_i to t_j. Hash-table keys are integers of the form, $Ni + j$, where N is an upper bound on the number of time-points in the network. [7] For example, if $N = 2^{14} = 16384$, then 28-bit values can be used for hash-table keys—which can be quickly computed using left-shift and addition operations.

A Note about Rigid Components and Undominated Edges. In a purely incremental context, constraints are never weakened or deleted. Thus, rigid components, once created, can never become non-rigid. Thus, it is safe to collapse each rigid component down to a single point as soon as it is created. Insodoing, the network remains free from rigidities, which simplifies the detection of dominated constraints. In contrast, a fully dynamic algorithm must handle the weakening or deleting of constraints and, thus, cannot afford to collapse all rigid components—because undoing such transformations can be too computationally costly. Thus, the fully dynamic algorithm in MYSYSTEM does not typically collapse rigid components. Thus, the network in MYSYSTEM may contain rigidities, thereby complicating the detection of dominated edges. For this reason, the detection of dominated edges in MYSYSTEM is restricted to cases where a *strictly shorter* alternative pathway is found.[8] In addition, the decremental algorithm can sometimes insert dominated edges into the *Precs* and *Succs* hash tables—because avoiding doing so would be too computationally costly. However, when the incremental algorithm detects these dominated edges, they are immediately removed from the *Precs* and *Succs* hash tables. Thus, in this sense, the fully dynamic algorithm in MYSYSTEM can be said to propagate along "mostly" undominated edges.

The Decremental Algorithm in MYSYSTEM. The decremental algorithm is used when an existing constraint, $t_j - t_i \leq \delta$, is either weakened or deleted. The algorithm has the following three phases:

(1) In a hash-table called *Changelings*, collect all pairs, (t_x, t_y), such that $\mathcal{D}'(t_x, t_y)$ *might* need updating.
(2) For each (t_x, t_y) in *Changelings,* check for shorter alternative pathways from t_x to t_y; collect the shortest alternatives in a hash-table called *AltPaths.*
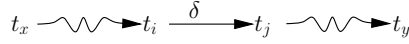
---

[5] This treatment of the zero time-point is somewhat similar to Cesta and Oddi's treatment of the zero time-point as both a source and a sink [2].

[6] $\mathcal{T} \backslash \{\mathbb{Z}\}$ denotes the set of time-points in $\mathcal{A}$ other than $\mathbb{Z}$.

[7] Demetrescu and Italiano [4] encode pairs in this way.

[8] In contrast, the INCR2004 algorithm also detects edges that are dominated by a path whose length is the *same* as that of the edge being dominated.

(3)   Incrementally propagate the constraints in *AltPaths*.

Phase 1. Consider the path from t_x to t_y shown below, where the wavy arrows represent shortest paths and δ is the *original* weight of the edge being weakened/deleted.

$$t_x \rightsquigarrow t_i \xrightarrow{\ \delta\ } t_j \rightsquigarrow t_y$$

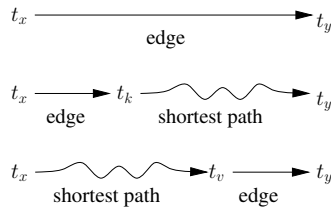The pair, (t_x, t_y), is collected during Phase 1 if and only if:

$$\mathcal{D}'(t_x, t_y) = \mathcal{D}'(t_x, t_i) + \delta + \mathcal{D}'(t_i, t_y)$$

All such pairs are collected using a two-pass algorithm that has the same structure as the *PropFwd* and *PropBkwd* phases of the incremental algorithm. Thus, Phase 1 takes time $O(k\Delta)$, where Δ is the number of pairs in *Changelings*.

After the *Changelings* hash-table has been populated, the corresponding distance-matrix entries are assigned new values, as follows. If the edge, $t_i t_j$, has been deleted, then each $\mathcal{D}'(t_x, t_y)$ is set to ∞, because the deletion of $t_i t_j$ might mean there no longer is any path from t_x to t_y. On the other hand, if $t_i t_j$ was simply weakened—say by an amount α—then each $\mathcal{D}'(t_x, t_y)$ is set to the value $\mathcal{D}'(t_x, t_y) + \alpha + 1$. Using this value, which is necessarily greater than the eventual updated value, forces $\mathcal{D}'(t_x, t_y)$ to be updated during Phase 2 or 3.
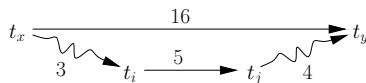
Since MYSYSTEM does not maintain any pointers to first or last steps of shortest paths (e.g., as done by Rohnert [13]), the *Changelings* hash table may end up containing some pairs whose distance-matrix entries do not need to be updated. Instead of maintaining complex auxiliary data structures to avoid this, the decremental algorithm discovers alternative paths during Phase 2 and 3 to ensure that the corresponding distance-matrix entries are restored.

Phase 2. For each (t_x, t_y) in *Changelings,* alternative pathways of the forms given below are collected in a hash-table called *AltPaths*.[9]

$$t_x \xrightarrow{\quad\text{edge}\quad} t_y$$

$$t_x \xrightarrow{\text{edge}} t_k \rightsquigarrow t_y$$
$$\text{shortest path}$$

$$t_x \rightsquigarrow t_v \xrightarrow{\text{edge}} t_y$$
$$\text{shortest path}$$

For some (t_x, t_y) in *Changelings*, it may be that no alternative paths exist. For other pairs, more than one such path may exist; however, only the shortest such paths are kept in *AltPaths*. The hash-key for the *AltPaths* hash table is the pair, (t_x, t_y); the value is the length of the alternative path. (Interior time-points on the path are not needed.)

Notice that the alternative pathways collected during Phase 2 may well have been dominated prior to the weakening (or deleting) of the edge $t_i t_j$, as illustrated below in the case of an alternative edge.

$$t_x \xrightarrow{\ 16\ } t_y$$
$$t_x \overset{3}{\rightsquigarrow} t_i \xrightarrow{\ 5\ } t_j \overset{4}{\rightsquigarrow} t_y$$

Prior to weakening $t_i t_j$ from 5 to 10, the edge, $t_x t_y$ was not a shortest path; however, afterward, it becomes a shorter (and possibly shortest) path. For this reason, the edges considered during Phase 2 are drawn from the set $\mathcal{C}$—which contains *all* of the edges in the network—not just those in the *Precs* and *Succs* hash tables.

Phase 3. During Phase 3, the alternative paths found in Phase 2 are incrementally propagated. There are several options for doing this. Each alternative path could, in turn, be completely propagated using the incremental algorithm. However, this sort of *depth-first* approach might result in a large amount of redundant propagation. Another option, analogous to A^* search, would be to sort the alternative paths according to how close their path-lengths were to the original value of $\mathcal{D}'(t_x, t_y)$ and apply the incremental algorithm to those alternative paths in their sorted order.

The decremental algorithm in MYSYSTEM takes an iterative, *breadth-first* approach. In the first iteration, each path in *AltPaths* is propagated *only one step* along the predecessors of t_x and the successors of t_y. Each one-step propagation generates a new update which is stored in a hash-table called *newAltPaths*. During the second iteration, each update in *newAltPaths* is propagated *only one step*, generating new updates for the third iteration. This iterative process terminates when no more updates are generated. Empirical evidence suggests that this form of incremental propagation is quite practical.

Removing Executed Time-Points. As discussed earlier, the fully dynamic algorithm does not typically collapse rigid components, because undoing such transformations in response to constraint relaxations can be too computationally costly. However, when a time-point, t, is executed, it forms a rigid component with Z_{in} and Z_{out} that is guaranteed to persist. Thus, it is safe to collapse this kind of rigid component. Doing so effectively removes t from the network by reorienting constraints involving t toward Z_{in} and Z_{out}.

4   Empirical Evaluation

The MYSYSTEM TCMS was tested on a set of thirty 25-agent scheduling problems drawn from the Phase 2 Evaluation for the DARPA Coordinators Project [15]. These kinds of problems are represented in the cTAEMS language, the details of which are described elsewhere [1]. The important characteristics of the test problems are shown in the top plot in Fig. 6. Each problem involved between 1507 and 3273 time-points (plotted on the horizontal axis) and between 803 and 1686 activities (ACTS).[10] For each problem, a centralized scheduler [14] was used to generate a set of agent schedules seeking to optimize the cTAEMS *quality* metric. In the process, the scheduler invoked the incremental algorithm of MYSYSTEM between 3461 and 7353 times (INCRS), and the decremental algorithm between 254 and 1185 times (DECRS). The resulting schedules included a total of between 139 and 243 activities (SCHEDS), and resulted in networks with between 3326 and 6797 edges (EDGES).

The middle plot of Fig. 6 shows the CPU time used by MYSYSTEM to do all of the temporal computations for each scheduling problem. The CPU time ranged from 2 seconds to 2 minutes for each problem. In the worst case, the 2 minutes of computation, spread over 8000 invocations of the incremental or decremental algorithms, averaged to about 15 msec per invocation.

The bottom plot of Fig. 6 shows the memory usage by MYSYSTEM. The number of finite distance-matrix cells (i.e., those that were actually stored in a hash table) ranged from about 77,000 to about 850,000 per problem. In contrast, the full distance matrix would have required between 2.2 and 10.7 million cells. Given that typical entries are four bytes, such a matrix could have required over 40 megabytes of memory. In contrast, the *total* memory used by MYSYSTEM during the course of each scheduling problem, most of which was dynamically allocated and freed, ranged from about 8 to 92 megabytes.

---

[9] Demetrescu and Italiano [4] refer to such pathways as *locally shortest*.

[10] Some activities share time-points; hence the number of time-points is somewhat less than double the number of activities.
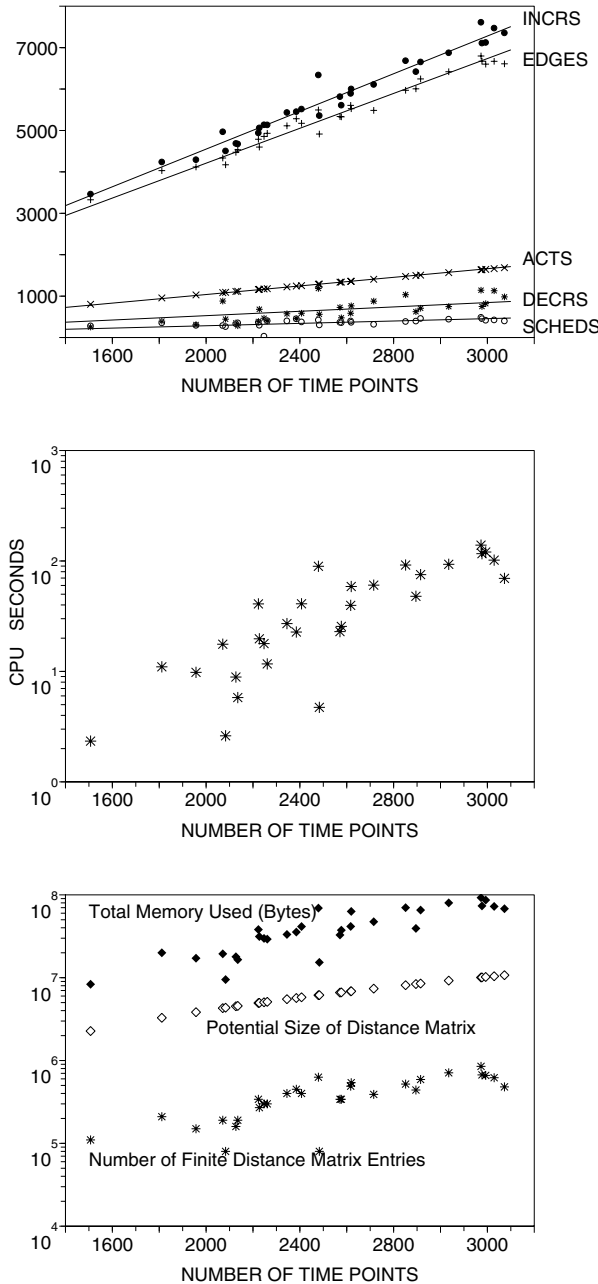
Figure 6. Results of experiments on 25-agent scheduling problems

All experiments were run on an IBM Thinkpad laptop with a 2.4GHz Intel processor using Allegro Common Lisp, version 8.1.

5 Conclusion

This paper presented a new temporal constraint management system, called MYSYSTEM, that combines novel STN representations with a fully dynamic propagation algorithm that is practical for real-world, real-time applications. The temporal network in MYSYSTEM includes special time-points to eliminate a common form of constraint propagation and reduce the number of distance-matrix entries that typically need to be computed. The fully dynamic algorithm extends an earlier incremental algorithm. It limits propagation to "mostly" undominated edges. The paper provided empirical re-

sults on temporal networks derived from a centralized scheduler applied to a variety of 25-agent scheduling problems involving thousands of time-points.

Acknowledgments

REFERENCES

[1] M. Boddy, B. Horling, J. Phelps, R. Goldman, R. Vincent, C. Long, and B. Kohout, 'C_taems language specification, version 1.06 (0)'.
[2] Amedeo Cesta and Angelo Oddi, 'Gaining efficiency and flexibility in the simple temporal problem', in *Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96)*, pp. 45–50. IEEE, (1996).
[3] Rina Dechter, Itay Meiri, and Judea Pearl, 'Temporal constraint networks', *Artificial Intelligence*, **49**, 61–95, (1991).
[4] C. Demetrescu and G. Italiano, 'A new approach to dynamic all pairs shortest paths', in *Proceedings of the 35th STOC*, pp. 159–166, (2003).
[5] Camil Demetrescu and Giuseppe F. Italiano, 'Improved bounds and new trade-offs for dynamic all pairs shortest paths', Technical Report ALCOMFT-TR-02-1, ALCOM, (2002).
[6] Shimon Even and Hillel Gazit, 'Updating distances in dynamic graphs', *Methods of Operations Research*, **49**, 371–387, (1985).
[7] Alfonso Gerevini, Anna Perini, and Francesco Ricci, 'Incremental algorithms for managing temporal constraints', Technical Report IRST-9605-07, IRST.
[8] Luke Hunsberger, 'Quantitative temporal reasoning in planning problems'. AAAI-2004 Tutorial MP-2, slides available at: http://www.cs.vassar.edu/~hunsberg.
[9] Luke Hunsberger, 'Algorithms for a temporal decoupling problem in multi-agent planning', in *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, (2002).
[10] Luke Hunsberger, *Group Decision Making and Temporal Reasoning*, Ph.D. dissertation, Harvard University, 2002. Available as Harvard Technical Report TR-05-02.
[11] Luke Hunsberger, 'Distributing the control of a temporal network among multiple agents', in *Proc. of the 2nd Int'l. Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS-03)*, (2003).
[12] G. Ramalingam and Thomas Reps, 'On the computational complexity of dynamic graph problems', *Theoretical Computer Science*, **158**, 233–277, (1996).
[13] Hans Rohnert, 'A dynamization of the all pairs least cost path problem', in *2nd Symposium of Theoretical Aspects of Computer Science (STACS 85)*, ed., Kurt Mehlhorn, volume 182 of *Lecture Notes in Computer Science*, 279–286, Springer, (1985).
[14] S. Smith, A.T. Gallagher, T.L. Zimmerman, L. Barbulescu, and Z. Rubinstein, 'Distributed management of flexible times schedules', in *Intl. Conf. on Autonomous Agents and Multiagent Systems*, (2007).
[15] Valerie Guralnik Thomas Wagner, John Phelps and Ryan VanRiper, 'COORDINATORS: Coordination managers for first responders', in *Proc. of the 3rd Intl. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2004)*. IEEE Computer Society, (2004).
[16] Mikkel Thorup, 'Worst-case update times for fully-dynamic all-pairs shortest paths', in *Annual ACM Symposium on Theory of Computing*, pp. 112–119, (2005).
[17] Ioannis Tsamardinos, *Reformulating Temporal Plans for Efficient Execution*, Master's thesis, University of Pittsburgh, 2000.