Fast Transformation of Temporal Plans for Efficient Execution

Ioannis Tsamardinos
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260
tsamard@cs.pitt.edu

Nicola Muscettola
Recom Technologies.
NASA Ames Research Center
Moffett Field, CA 94035
mus@ptolemy.arc.nasa.gov

Paul Morris
Caelum Research.
NASA Ames Research Center
Moffett Field, CA 94035
pmorris@ptolemy.arc.nasa.gov

Abstract

Temporal plans permit significant flexibility in specifying the occurrence time of events. Plan execution can make good use of that flexibility. However, the advantage of execution flexibility is counterbalanced by the cost during execution of propagating the time of occurrence of events throughout the flexible plan. To minimize execution latency, this propagation needs to be very efficient. Previous work showed that every temporal plan can be reformulated as a dispatchable plan, i.e., one for which propagation to immediate neighbors is sufficient. A simple algorithm was given that finds a dispatchable plan with a minimum number of edges in cubic time and quadratic space. In this paper, we focus on the efficiency of the reformulation process, and improve on that result. A new algorithm is presented that uses linear space and has time complexity equivalent to Johnson's algorithm for all-pairs shortest-path problems. Experimental evidence confirms the practical effectiveness of the new algorithm. For example, on a large commercial application, the performance is improved by at least two orders of magnitude. We further show that the dispatchable plan, already minimal in the total number of edges, can also be made minimal in the maximum number of edges incoming or outgoing at any node.

Introduction

In a control system that distinguishes a deliberative layer (*planner*) and a reactive layer (*executive*) (Pell *et al.* 1997; Bonasso *et al.* 1997; Wilkins *et al.* 1995; Drabble, Tate, & Dalton 1996; Simmons 1990; Musliner, Durfee, & Shin 1993; Bresina *et al.* 1993), the function of a plan is to provide robust and effective directives to the executive on how to control a system toward desired behaviors. To be robust against uncertainty in the execution environment a plan must be *flexible*, i.e., must specify a set of possible acceptable behaviors. The executive should be able to choose among such behaviors on the basis of the actual execution conditions. To be effective a plan must be *localized*, i.e., it must be possible for the executive to locally process the constraints in the plan and quickly decide which action to execute next.

To obtain flexibility one can explicitly represent in the plan the relationship between a set of plan parameters as a network of constraints. When receiving the constraint network, the executive will iteratively pick one variable and decide which value to assign to it. To make this decision the executive needs to propagate to the current variable the consequence of the value assignments that have already been made.

Relying on explicit constraint propagation during execution can be costly. In fact, the greater the time needed to propagate through the constraint network, the higher will be the total time needed by the executive to decide when and how to execute a task. It can be shown that this decision time determines the intrinsic uncertainty on the exact time of occurrence of any event in the plan (Muscettola *et al.* 1998). The more precise we want the execution of a plan to be, the less propagation an execution algorithm should perform. This is particularly important when plans are used in mission critical applications (Pell *et al.* 1997; Carpenter, Driscoll, & Hoyme 1994) for which the executive must guarantee to operate within a specific time bound.

Fortunately for certain classes of constraints one can rely on the special nature of the execution constraint propagation process in order to significantly speed it up. In the rest of the paper we will focus on flexible plans that represent temporal information as a Simple Temporal Network (STN) (Dechter, Meiri, & Pearl 1991). In previous work (Muscettola, Morris, & Tsamardinos 1998) we described a simple dispatcher, i.e., an execution algorithm that maximally localizes execution propagation in STNs. We showed that any STN can be transformed into an equivalent one that is both *dispatchable* and *minimum*. An STN is dispatchable if a dispatcher can generate all assignments of time to time variables that are consistent with the constraints in the STN. A dispatchable STN is minimum if it contains the minimum number of constraints among all dispatchable networks.

The main focus in the previous paper was to establish the existence of the transformation to a minimum dispatchable network. A simple algorithm was given that performs the transformation in $O(N^2)$ space and $O(N^3)$ time, where N is the number of variables in the original STN. While this is fine for problems of moderate size (hundreds of nodes), it becomes unworkable for large graphs (tens of thousands of nodes) that may occur in some applications.

In this paper we give a new transformation algorithm that, when applied to an input STN with N nodes and E edges, uses space linear in the size of the input and output STNs and $O(NE + N^2 \ln N)$ time. For problems in which E is roughly proportional to N the new algorithm can yield very big improvements over our previous one. In particular, as discussed later in the paper, our current implementation can solve a large problem from a commercial domain in minutes using tens of megabytes of memory while best estimates for the older algorithm yield processing times of several days and memory usage of tens of gigabytes.

We also revisit the concept of minimality for a dispatchable network. It may be argued that to achieve the best execution performance with a dispatcher, it is not sufficient for a dispatchable STN to have a minimum number of edges. Among all such networks it is possible to select a *fastest dispatchable* network as the one that also minimizes the maximum in/out degree, i.e., the number of temporal constraints that enter/exit a variable in the STN. To this end we present an additional $O(N^2 \ln N)$ time transformation step that when applied to the output of our new transformation algorithm, yields a fastest dispatchable network.

The rest of the paper is organized as follows. The first section summarizes the results of our previous work and provides the background for the rest of the paper. The next section formally describes the new algorithm, while the succeeding sections justify the algorithm and discuss experimental results for random and natural problems. A further section describes the additional transformation to yield the fastest dispatchable network, and the final section concludes the paper.

Temporal Network Dispatchability

In this section we summarize the main results of our previous work. See (Muscettola, Morris, & Tsamardinos 1998) for details and proofs.

Recall that Simple Temporal Networks (Dechter, Meiri, & Pearl 1991) are directed graphs where each node is an *event* or *time point* (e.g., time points A and B) and each edge AB is marked with a bound delay $[d, D]$. The interpretation of each edge is that if T_A and T_B are the times of occurrence of A and B respectively, then in a consistent execution $d \leq T_B - T_A \leq D$. It has been shown (Dechter, Meiri, & Pearl 1991) that finding the ranges of execution times for each event's *time bounds*, is equivalent to solving two single-source shortest-path problems (Cormen, Leiserson, & Rivest 1990) on a simple transformation of the STN graph. Figure 1 shows such a distance graph obtained from a simple plan with two tasks BD and CD of fixed durations, respectively 1 and 2 time units, that synchronize at the end (event D) and must start within 10 time units of a time origin (event A). From now on, any time we refer to an STN, we will mean the transformed distance graph.

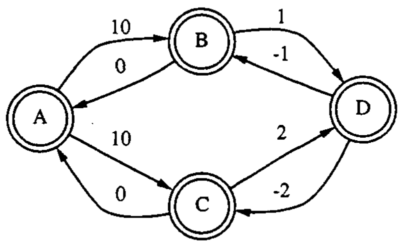

Figure 1: Distance Graph.

The simplest algorithm that can select occurrence times for events at execution is a *dispatcher* (figure 2). Unfortunately STNs may not always be correctly executed by a dispatcher. For example, since the network in figure 1 does not contain an explicit edge of length 1 to synchronize C and B, it is possible for the dispatcher to select B before C, yielding an inconsistent execution.

The STNs that can always be correctly executed by a dispatcher are called *dispatchable.* It is always possible to transform any STN into an equivalent dispatchable STN. Trivially, it can be shown that the all-pairs shortest-path graph ($\mathcal{APSP}$) (Cormen, Leiserson, & Rivest 1990) derived from the original STN is dispatchable. However, this would be the largest dispatchable network for the problem. We are interested in finding small dispatchable networks, in fact, networks that contain the minimum total number of edges, or *minimal dispatchable* STNs.

In order to find the minimal number of edges in a dispatchable graph, we look for edges that are *dominated* in $\mathcal{APSP}$, i.e., whose propagations are subsumed by those of some other edge in all possible executions. It has been shown in our previous paper that, for execution purposes, upper and lower bounds can be independently propagated. In fact, it is sufficient to propagate upper bounds through non-negative edges and lower bounds through negative edges (in the reverse direction (Dechter, Meiri, & Pearl 1991)). So, dominance relations between lower-bound links (*lower-domination*) can be checked separately from those between upper-bound links (*upper-domination*). It was also shown that removal of a dominated edge does not affect the dominance relation between other pairs of edges.

The following fundamental filtering theorem applies.

Theorem 1 (Triangle Rule) *Consider an $\mathcal{APSP}$ derived from a consistent STN.*

```
TIME DISPATCHING ALGORITHM:
  1. Let
      A = {start_time_point}
          current_time = 0
      S = {}
  2. Arbitrarily pick an event TP in A such
     that current_time is in TP's time bound;
  3. Set TP's execution time to current_time
     and add TP to S;
  4. Propagate the time of execution
     to its IMMEDIATE NEIGHBORS in the distance
     graph;
  5. Put in A all events TPx such that all
     negative edges starting from TPx have a
     destination that is already in S;
  6. Wait until current_time has advanced to
     some time between
        min{lower_bound(TP) : TP in A}
     and
        min{upper_bound(TP) : TP in A}
  7. Go to 2 until every event is in S.
```

Figure 2: The Dispatching Execution Controller.

(1) A non-negative edge AC is upper-dominated by another non-negative edge BC if and only if $|AB| + |BC| = |AC|$.

(2) A negative edge AC is lower-dominated by another negative edge AB if and only if $|AB| + |BC| = |AC|$.

The transformation algorithm that we proposed in our previous work is quite simple: it applies the triangle rule in all possible ways to the $\mathcal{APSP}$ obtained from the original STN. Since this algorithm relies on all shortest paths being known in advance, it requires $O(N^2)$ space. Moreover, the time needed by the algorithm is dominated by applying the filtering rule (constant time) over all possible triples of nodes, yielding a time complexity of $O(N^3)$.

Fast Filtering Algorithm

The algorithm sketched in the previous section precomputed $\mathcal{APSP}$ and then applied the dispatchability filtering step. However we can do better if we interleave filtering with the process of computing the shortest-paths. The algorithm we propose is a modification of the Johnson's all-pairs shortest-path algorithm and its overall structure is described in figure 3. The references to lines of JOHNSON in the figure refer to the line numbering in the description of Johnson's algorithm at page 569 of (Cormen, Leiserson, & Rivest 1990).

In the rest of the paper we will discuss the formal details of how steps 2, 3.b, 3.c and 3.d work and why. Here we want to give some general observations that will help frame the rest of the discussion.

The first observation is that it is possible to exploit to our advantage any amount of "rigidity" present in the temporal network. Step 2 examines the graph G and identifies all the sets of time points that are rigidly

```
FAST-DISPATCHABILITY-MINIMIZATION (G)

  1.  Run Bellman-Ford pre-processing step
      of Johnson's algorithm
      [ lines 1-7 of JOHNSON]
  2. [RIGID COMPONENTS PROCESSING]
      Identify all rigid components in G.
      For each rigid component RC;
        a) find a single node representative for
           MIN(RC) and contract G so that all
           nodes in RC are represented by MIN(RC).
        b) output the minimum dispatchable graph for RC.
      Call the contracted graph obtained after this
      step CONTR_G.
  3. [DAG DISPATCHABILITY MINIMIZATION]
      For each node A in CONTR_G;
        a.  Run Dijkstra's algorithm on CONTR_G
            with A as the source [ lines 9-11 of JOHNSON]
        b.  Do a preliminary depth-first search
            of the predecessor graph computed at step a.
            to collect the nodes into reverse-postorder.
        c.  [UPPER-DOMINATES MINIMIZATION]
            Find and output all non UPPER-BOUND dominated
            edges with source in A;
        d.  [LOWER-BOUND DOMINATES MINIMIZATION]
            Find and output all non LOWER-BOUND dominated
            edges in G with source in A;
```

Figure 3: Fast Dispatch Minimization Algorithm.

connected, i.e., such that once the execution time of one of them is fixed, we know exactly when all the others in the set must execute. Given this we will show that the entire set can be represented in the minimization process by a single node, without loss of information. The complexity of this step (beyond the propagation already required by Johnson's Algorithm) is $O(N+E)$, where N and E are the numbers of nodes and edges, respectively, in the input graph. This is well within the Johnson bounds.

Once the graph has been contracted, step 3 uses single-source distance information to scan the graph for dominated edges in a systematic way. The key data structure in this process, and in the rigidity analysis in step 2, is the *predecessor graph*, which is a subgraph of the distance graph that retains only the edges that participate in shortest paths from the current source. We will show that one important consequence of step 2 is that all predecessor graphs of CONTR_G are DAGs. This means the non-dominated edges can be found through a fixed number of depth-first searches in CONTR_G, making the overall cost of filtering $O(N^2 + NE)$ where N and E are the numbers of nodes and edges in CONTR_G. This is also within the bounds of Johnson's algorithm.

Finally, we observe that the algorithm only requires space needed to store the input graph G, and the output minimal dispatchable graph. Moreover, steps 2.b, 3.c and 3.d can output the edges incrementally, so they do not actually need to be stored in main memory. This is a significant space improvement with respect to the previous approach, which required $O(N^2)$ space to store the intermediate all-airs shortest-path graph obtained by Johnson's Algorithm.

The result is an algorithm that, as we shall see, makes it practical to apply the filtering process to enormous networks where it would be infeasible to use the cubic filtering algorithm.

Notation

We use upper case italics to denote nodes in a temporal network or distance graph. Edges and paths are denoted by lower italics. The shortest-path distance from node X to node Y is denoted by $|XY|$. In the context of a single-source distance computation from an origin node S, the shortest-path distance from S to any node X is denoted by $d(X)$. The All-Pairs Shortest-Path graph is denoted by $\mathcal{APSP}$. We will use XY to denote an edge from X to Y in $\mathcal{APSP}$.

Predecessor Graph

The central data structure needed for steps 2 and 3 is the *predecessor graph.* The predecessor graph is a generalization of the *predecessor tree* generated by a single-source shortest-path algorithm (Cormen, Leiserson, & Rivest 1990). The predecessor tree is constructed while finding *some* shortest-path from a source node to every other node. The predecessor graph, on the other hand, concisely represents information on *all* shortest-paths from the source node.

Definition 1 *Given a consistent distance graph, the* predecessor graph *with respect to an origin node S, denoted by $\mathcal{P}(S)$, is the subgraph defined by the set of all edges on shortest paths from S.*

The next result gives a characterization that allows edges to be checked locally for membership in the predecessor graph.

Theorem 2 *An edge from node X to node Y is in $\mathcal{P}(S)$ if and only if*

$$d(Y) = d(X) + b(X,Y)$$

where $b(X,Y)$ is the length of the edge, and $d(X)$ and $d(Y)$ are the shortest-path distances from S to X and Y, respectively.

Proof Let e be the edge from X to Y.

Suppose the distance equation $d(Y) = d(X)+b(X,Y)$ holds. Then the path from S to Y that passes through X and along e has length equal to the shortest-path distance to Y. Thus, e is on a shortest path.

Conversely, suppose e is on a shortest path p from S. Without loss of generality, we may assume e is the last edge in p. Then $d(Y) = \text{length}(p)$. Let p' be the part of p that does not include the final edge e. Since any subpath of a shortest path is itself a shortest path, we have $d(X) = \text{length}(p')$. It follows that $d(Y) = d(X) + b(X,Y)$. □

Corollary 2.1 *Every path in $\mathcal{P}(S)$ is a shortest path. Moreover, the length of such a path from X to Y in $\mathcal{P}(S)$ is given by $d(Y) - d(X)$.*

Proof Consider a path p from a node X to a node Y. If we sum the distance equations from Theorem 2 for each edge in the path, we obtain $d(Y) = d(X) + l(p)$, where $l(p)$ is the length of p. It follows that p is a shortest path, since otherwise we could derive a smaller value for $d(Y)$ by choosing a shorter alternative to p. It also follows that $l(p) = d(Y) - d(X)$. □

Dominance

As noted before, we need to systematically apply the Triangle Rule to all triangles in $\mathcal{APSP}$ for which the triangle equality among distances applies. The following theorem allows us to precisely pinpoint these triangles among the $O(N^3)$ possible triangles in $\mathcal{APSP}$. This observation is a key element in the efficiency of the proposed algorithm.

Theorem 3 *Let A, B, and C be nodes in a consistent distance graph. Then the equation $|AC| = |AB| + |BC|$ holds if and only if there is a path from B to C in $\mathcal{P}(A)$.*

Proof

Suppose the equation $|AC| = |AB| + |BC|$ holds. Then there is a shortest path from A to C that passes through B. By definition, all the edges on this path are in $\mathcal{P}(A)$. In particular, the subpath from B to C is in $\mathcal{P}(A)$.

Conversely, suppose $\mathcal{P}(A)$ includes a path from B to C. By Corollary 2.1, this is a shortest path and its length is $d(C) - d(B)$. Thus, $|BC| = d(C) - d(B) = |AC| - |AB|$. The result follows. □

We now construct specific tests for upper and lower dominance. The edges considered for elimination or retention in the following theorems are edges in the implicit $\mathcal{APSP}$. The algorithm does not build the full $\mathcal{APSP}$ but only outputs an edge in $\mathcal{APSP}$ according to the values of the upper and lower dominance tests. The implicit edge AC of $\mathcal{APSP}$ is considered by examining the properties of the node C with respect to $\mathcal{P}(A)$.

First we derive a test for lower-dominance of negative edges.

Theorem 4 *A negative edge AC is lower-dominated by a negative edge AB if and only if there is a path from B to C in $\mathcal{P}(A)$.*

Proof Immediate by the Triangle Rule (Theorem 1) and Theorem 3. □

Note that in the case where $\mathcal{P}(A)$ is a DAG, this means that the edge AC associated with a negative-distance node C may be eliminated if and only if there is another negative-distance node B that precedes C. This suggests an algorithm that traverses the DAG, collecting minimal negative nodes for retention. (Although predecessor graphs are not acyclic in general, step 2 of the algorithm reduces them to DAGs for the benefit of step 3.)

The next theorem gives a condition for determining whether a non-negative edge is upper-dominated.

Theorem 5 *A non-negative edge AC is upper-dominated if and only if there is a node B, distinct from A and C, such that $|AB| \leq |AC|$ and there is a path from B to C in $\mathcal{P}(A)$.*

Proof

Suppose $|AB| \leq |AC|$ and there is a path from B to C in $\mathcal{P}(A)$. By Theorem 3, $|AC| = |AB| + |BC|$. It follows that $|BC| \geq 0$. By the Triangle Rule, AC is then upper-dominated by $|BC|$.

The argument in the converse is just the reverse. (Note that if AC is upper-dominated by BC, then $|BC|$ is non-negative by definition.) □

The above conditions can be used to decide in time $O(N+E)$ which edges emanating from a node A should be retained in the output graph. While traversing $\mathcal{P}(A)$, the algorithm propagates two pieces of information. The first datum indicates whether a negative-distance node has been encountered at a predecessor node. The second datum keeps track of the minimum distance value for all the predecessor nodes other than A itself. These data are used to determine whether the implicit edge AC corresponding to a node C is dominated. If $|AC|$ is negative, this depends on the first datum in the manner dictated by Theorem 4. If $|AC|$ is non-negative, then Theorem 5 shows that the edge is dominated if and only if the minimum distance value for the predecessors does not exceed the distance value for C (i.e., $|AC|$).

The above algorithm correctly identifies the edges to include in the output, provided the nodes are visited in an order that ensures the propagated values are calculated correctly. In the case where the predecessor graph is acyclic, a reverse-postorder traversal is guaranteed to visit all ancestors before a given node. In the case where the predecessor graph is not initially a DAG, we will see that the strongly-connected components can be effectively contracted to single points, resulting in a DAG. This is considered in the next section.

Identifying and Using Rigid Components

An important concept for our analysis concerns a situation where two nodes have a connection with no slack. More formally, two points X and Y are ***rigidly-related*** if in the distance graph we have $|XY| + |YX| = 0$.

It is easy to verify that, given a consistent distance graph, the property of being rigidly-related determines an equivalence relation. We call each equivalence class a *rigid component* ($\mathcal{RC}$). We will see that constructing a dispatchable graph can be simplified if these can be identified; in that case the problem can be reduced to one where each $\mathcal{RC}$ is contracted to a single point.

Identifying Rigid Components

Before considering the contraction process in detail, we address the issue of how to identify each $\mathcal{RC}$. For this, we offer the following result.

Theorem 6 *Given a consistent distance graph, and a single-source propagation from an arbitrary node S that reaches every node in the graph, each $\mathcal{RC}$ of the distance graph coincides with a strongly-connected component (Cormen, Leiserson, & Rivest 1990) of the predecessor graph $\mathcal{P}(S)$ (and vice versa).*

Proof: Suppose X and Y are rigidly-related. Consider a shortest path from the source to X. This can be extended by a shortest path to Y and then back again to X. Since $|XY| + |YX| = 0$, this is also a shortest path to X. Thus, the predecessor graph includes a path from X to Y and vice versa, so X and Y are in the same strongly-connected component.

Conversely, suppose X and Y are in the same strongly-connected component of the predecessor graph. Then there is a path from X to Y in $\mathcal{P}(S)$. By Corollary 2.1, $|XY| = d(Y) - d(X)$. Similarly, $|YX| = d(X) - d(Y)$. It follows that X and Y are rigidly-related. □

The theorem states that we can find all $\mathcal{RC}$ subgraphs by doing a single-source propagation from a suitable starting point in the distance graph. Since Johnson's Algorithm requires an initial run of Bellman-Ford to set up a "potential-function" value at every node, it is convenient to use this to determine the strongly-connected components and hence $\mathcal{RC}$ subgraphs. There is a well-known algorithm (Cormen, Leiserson, & Rivest 1990) for computing strongly-connected components (SCCs) that runs in time linear in the number of edges. This has two parts, an initial depth-first search to collect the nodes in reverse-postorder, and a secondary traversal to trace out each SCC. For our purposes, it is necessary to do some further processing on every SCC. It is convenient to piggy-back this on the part that traces out the SCC.

Rigid Component Contraction

In order to contract a $\mathcal{RC}$ to a single-point for further processing, it is necessary to choose some point in the $\mathcal{RC}$ as a representative or *leader*. The algorithm selects a *minimum point* for this purpose, that is, a node X such that $d(X)$ is minimum over the $\mathcal{RC}$, where $d(X)$ is the distance from the origin node of the single-source propagation.

Once a leader is selected, some further issues arise. To prepare the $\mathcal{RC}$ for contraction, we need to modify the input graph so that all incoming and outgoing edges of the $\mathcal{RC}$ are replaced by equivalent edges to/from the leader. This is accomplished by appropriately modifying the edge lengths. Second, in order to justify the contraction, we need to show that the $\mathcal{RC}$ can be represented by the leader as far as output edges are concerned. We do this by demonstrating that potential output edges to/from the interior of the $\mathcal{RC}$ are dominated by those to/from the leader. There will also be edges in the output that correspond to internal edges of the $\mathcal{RC}$. These are identified and collected prior to the contraction. This step can be accomplished by considering the $\mathcal{RC}$ in isolation, and simply consists of arranging the $\mathcal{RC}$ nodes in a doubly-linked chain.

Rigid Component Edge Rearrangement

We now consider the preparation step that rearranges the input graph by redirecting the outgoing and incoming edges of each $\mathcal{RC}$ to the leader node.

Theorem 7 *Suppose X and Y are rigidly-related with $|XY| = b$. Then (1) an edge YZ of length u is equivalent to an edge XZ of length $u+b$, and (2) an edge ZY of length v is equivalent to an edge ZX of length $v-b$.*

Proof: The given rigid relation corresponds to the equation $T_Y = T_X + b$. In its presence, the inequality $T_Z - T_Y \leq u$ is equivalent to $T_Z - T_X \leq u + b$. A similar argument works for (2). □

Notice that if two nodes in the $\mathcal{RC}$ are connected to the same node Z outside the $\mathcal{RC}$, then the theorem provides two replacement inequalities of the form $T_Z - T_X \leq u1$ and $T_Z - T_X \leq u2$. In this case, one of the inequalities is subsumed, and we need only retain the edge corresponding to $T_Z - T_X \leq \min(u1, u2)$. Thus, the replacement process allows us to recognize and remove some logically redundant edges in the distance graph.

After the edge replacement, the only connection the $\mathcal{RC}$ has to the rest of the graph is through the leader node.

Rigid Component Edge Elimination

In this section we prove dominance properties for edges entering or exiting nodes in a $\mathcal{RC}$. In particular we see that all edges that start or end with an "interior" node are dominated. An interior node is a node in $\mathcal{RC}$ other than the leader (minimum mode).

Lemma 1 *Suppose L and A belong to the same $\mathcal{RC}$, and B is any other node. Then $|AB| = |AL| + |LB|$ and $|BA| = |BL| + |LA|$.*

Proof:

From the properties of shortest-path graphs, the triangle inequalities $|AB| \leq |AL| + |LB|$ and $|LB| \leq |LA| + |AB|$ must hold. The second inequality can be rewritten as $|AB| \geq |LB| - |LA|$. Since $|LA| = -|AL|$, we have $|AB| \geq |LB| + |AL|$. Combining this with the first inequality gives $|AB| = |AL| + |LB|$

The proof of the second condition is similar. □

The following result permits the elimination from the output of edges to/from interior nodes of an $\mathcal{RC}$. The proof requires an assumption that there are no *zero-related* pairs of nodes in the distance graph. (Two nodes X and Y are zero-related if $|XY| = |YX| = 0$.) This is actually not a significant restriction because zero-related nodes must be executed simultaneously, and so they may be collapsed to a single node. The system described in this paper detects zero-related nodes during the $\mathcal{RC}$ identification phase and automatically collapses them.

Theorem 8 *Assume a consistent distance graph with no zero-related pairs. Suppose L and A are distinct nodes in a rigid component, where L is the leader of the $\mathcal{RC}$, and suppose B is a node not in the $\mathcal{RC}$. Then the edges AB and BA are always dominated.*

Proof:

We will consider only AB. The dominance proof for BA is analogous. By lemma 1, we have $|AB| = |AL| + |LB|$. We distinguish two cases, depending on whether $|AB|$ is negative or non-negative.

Suppose first that $|AB|$ is negative. Note that $|AL|$ is also negative (assuming there are no zero-related pairs) since L is the minimum node of the $\mathcal{RC}$. Since $|AB| = |AL| + |LB|$, the Triangle Rule (Theorem 1) allows us to conclude $|AB|$ is lower-dominated by $|AL|$.

Now suppose $|AB|$ is non-negative. Since $|AB| = |AL| + |LB|$ and $|AL|$ is negative, it follows that $|LB|$ is non-negative. Then $|LB|$ upper-dominates $|AB|$ by the Triangle Rule. □

We have shown that output edges to and from non-leader nodes of the $\mathcal{RC}$ to the rest of the graph are dominated, and so may be eliminated from the final output. Edges in the output graph that are entirely within the $\mathcal{RC}$ can be generated independently of the rest of the graph, and may be dumped immediately. (One valid arrangement consists of edges that connect the nodes of the $\mathcal{RC}$ in a doubly-linked chain.) Thus, the non-leader nodes play no essential role in further processing of the input graph, and so may be deleted. The effect is the same as contracting the $\mathcal{RC}$ to a single node. (Note, however, that this may entail an arbitrary choice of which edges to eliminate in cases of mutual dominance. We will see in a later section that there may be reason to redistribute some of the unfiltered edges to the $\mathcal{RC}$ interior as a postprocessing step.)

Consequences of Contraction

An obvious benefit of the $\mathcal{RC}$ contraction process is that it may reduce the size of the network, but this is not its primary purpose. Because of the equivalence of rigid components and strongly-connected components, removal of the former will also eliminate the latter. Thus, subsequently determined predecessor graphs are acyclic (DAGs) This facilitates dominance identification. For example, reverse-postorder traversals can be used to ensure that parents are visited before children in descents through the predecessor graph. These traversals require only linear time (in the number of edges).

Another consequence is that the dominance relations are simplified by eliminating mutual-dominating edges. To see this we need the following result.

Theorem 9 *(1) Suppose AC and BC are two non-negative edges in $\mathcal{APSP}$. Then AC and BC mutually dominate each other if and only if A and B are rigidly-related.*

(2) Suppose AC and AB are two negative edges in the $\mathcal{APSP}$. Then AC and AB mutually dominate each other if and only if B and C are rigidly-related.

	Input Graph		Output Graph		
Nodes	Edges	Degree	Edges	Degree	Time
Grid-SSquare family data					
257	768	16	744.8	12	1.18
1025	3072	32	2997.4	22.8	29.66
4097	12288	64	12010	45.4	878.64
Grid-SWide family data					
257	768	32	745.6	21.8	1.13
1025	3072	128	2982.5	90.8	18.57
4097	12288	512	11905.4	377.4	302.13
Grid-SLong family data					
257	768	8	746.2	5.2	1.08
1025	3072	8	3002.2	5.8	17.17
4097	12288	8	12028.4	5.8	297.22
Grid-NHard family data					
257	2166	12	570	4	3.33
1025	9944	12	2311.8	4.2	165.06
4097	40904	12	9235.8	5	12272.7

Table 1: Data from random generated networks.

	Input Graph			Output Graph		
Nodes	Edges	Deg.	#RC	Edges	Deg.	Time
61	133	24	37	106	18	0.02
63	135	26	35	104	17	0.02
42	84	11	28	64	11	0.01
85	194	48	46	153	27	0.03
59487	192790	1151	7111	190733	4104	2230

Table 2: Data from natural plans.

The proof is omitted for brevity, but is an easy consequence of combining the triangle conditions associated with the mutual dominance relations. The theorem shows that mutually dominating edges imply nontrivial rigid components. Thus, in a graph where the $\mathcal{RC}$ subgraphs are contracted, the dominance relation becomes asymmetric. This removes any danger of inadvertently eliminating both edges in a mutually-dominating pair.

Experimental Results

The algorithm was implemented in Lisp and the experiments were run on an Ultra-2 Sparc. We experimented with five natural temporal plans as well as 60 randomly generated ones. Four out of the five natural plans were generated by the planner/scheduler of the Remote Agent control architecture (Pell *et al.* 1997) and were relatively small, averaging about 60 nodes. The fifth plan was taken from an an avionics processor schedule for a commercial aircraft provided by Honeywell (Carpenter, Driscoll, & Hoyme 1994) and was much larger, having about 60,000 nodes.

For the generation of the random networks we used the same code as in (Cherkassky, Goldberg, & Radzik 1996), where a variety of shortest paths algorithms are evaluated on a number of different families of randomly generated networks. We chose the four families of networks that most approximate STNs found in natural plans: Grid-SSquare, Grid-SWide, Grid-SLong, and Grid-NHard. For every family and every different size of the initial network we generated 5 network instances. The averages for a number of different statistics are reported in table 1. The initials in tables 1 and 2 stand for: **Nodes**, the number of nodes in the input and output graphs; **Edges**, the number of edges; **Degree**, the maximum out-degree; **Time**, the time in seconds for the filtering algorithm to run; and **#RC**, the number of rigid components in the input graph. All statistics are within 15% of the reported average.

From the results of tables 1 and 2 we observe that the number of edges in the output graph is much smaller than the worst case corresponding to the $\mathcal{APSP}$, which has $N(N-1)$ directed edges. In fact, in our experiments, the output graph was slightly smaller than the input graph—an indication of some redundancy in the latter.

A second observation is that the performance of the algorithm is greatly improved relatively to the old cubic algorithm. A 60,000 nodes network, such as the one displayed in table 2, would take at least an estimated 48.7 days to be filtered (counting only memory accesses, assuming an extremely fast memory cycle of 5ns) and would use about 14.14 GBytes of memory (assuming 4 bytes per edge). The new algorithm filtered the network in about 37 minutes using 25.3 MBytes of memory. Out of these only 204.7 KBytes are used for data structures other than the input graph.

Minimizing the Outdegree

The foregoing sections of this paper have addressed the issue of producing a dispatchable network with a minimum number of edges. However, the time needed for propagation by the dispatching controller (Step 4 in Figure 2) depends on the *indegree* and *outdegree* of the node, i.e., the number of edges to or from from the node. Thus, to optimize the Real-Time execution guarantee we need to minimize the *maximum* indegree and outdegree in the network. In this section, we briefly sketch how this can be done for the outdegree within the framework developed in this paper. A similar analysis can be used to minimize the maximum indegree.

Out of all the edge-minimal dispatchable networks, we seek one that minimizes the maximum outdegree. (Notice that we need only consider the edge-minimal dispatchable networks. If a dispatchable network has a small maximum outdegree but does not have a minimum number of edges, we can eliminate edges from it until it is edge-minimal.) The different edge-minimal networks correspond to different choices of which edge to eliminate in cases of mutual dominance. Theorem 9 shows that mutual dominance is associated with the rigid components. Within the framework of the Fast-Filtering algorithm, we can ensure minimality in terms of maximum outdegree by judiciously choosing which edges to keep among those outgoing from the nodes in each $\mathcal{RC}$.

Consider an $\mathcal{RC}$ with leader L. The algorithm as presented resolves mutual dominance in a way that assigns to L all outgoing edges from the $\mathcal{RC}$ (in the final output graph). To assure minimality of maximum outdegree, we need instead to redistribute those out-edges as evenly as possible among the nodes of the $\mathcal{RC}$. The redistribution actually involves choosing alternate members of mutual dominating pairs, but we visualize it as "moving" the out-edges from the minimum node to the other nodes of the $\mathcal{RC}$. An examination of the mutual dominance conditions associated with an $\mathcal{RC}$ shows that only non-negative out-edges may be moved, and they may only be moved over the range in which they remain non-negative. During the move algorithm a number of internal nodes of $\mathcal{RC}$ will be "poorest," i.e., will have a minimum number of out-edges. It is easy to see that a greedy algorithm that moves edges with the shortest ranges first, and moves them to one of the currently "poorest" nodes will provide an optimal distribution. Since the range depends on the edge length, this requires sorting the list of out-edges according to length. Finding a "poorest" node involves searching the $\mathcal{RC}$ within the allowable range. Adopting conservative upper bounds for these operations, the complexity for the redistribution is $\sum_i (E_i \log E_i + E_i * K_i)$ where E_i is the number of out-edges from the leader, and K_i is the number of nodes, of the i-th $\mathcal{RC}$ in an enumeration of all $\mathcal{RC}$ subgraphs. Note that $E_i < N$ and $\sum_i K_i = N$, where N is the total number of nodes. Thus, an upper bound on the complexity is given by $O(N^2 \log N)$, which fits within the bound of Johnson's Algorithm.

Conclusion

We have presented a sophisticated algorithm for reformulating temporal plans so that they may be executed with local propagation. With linear space and time complexity equivalent to Johnson's Algorithm, this is a substantial improvement over the previous simpler quadratic space and cubic time algorithm.

Acknowledgments. We thank Marc Ringer and Mark Boddy of the Honeywell Technology Center who provided us with the avionics processor schedule that motivated much of this research. Participation by the first author was supported by the National Science Foundation under grants IRI-9258392 and IRI-9619579.

References

Bonasso, R. P.; Kortenkamp, D.; Miller, D.; and Slack, M. 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence* 9(1).

Bresina, J.; Drummond, M.; ; and Kedar, S. 1993. Reactive, integrated systems pose new problems for machine learning. In Minton, S., ed., *Machine Learning Methods for Planning.* San Mateo, California: Morgan Kaufmann.

Carpenter, T.; Driscoll, K.; and Hoyme, K. C. J. 1994. Arinc 659 scheduling: Problem definition. In *Proceedings of 1994 IEEE Real Time System Symposium.* IEEE.

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

Cormen, T.; Leiserson, C.; and Rivest, R. 1990. *Introduction to Algorithms.* Cambridge, MA: MIT press.

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

Drabble, B.; Tate, A.; and Dalton, J. 1996. O-plan project evaluation experiments and results. Oplan Technical Report ARPA-RL/O-Plan/TR/23 Version 1, AIAI.

Muscettola, N.; Morris, P.; Pell, B.; and Smith, B. 1998. Issues in temporal reasoning for autonomous control systems. In Wooldridge, M., ed., *Proceedings of the Second Int'l Conference on Autonomous Agents.* ACM Press.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proc. of Sixth Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98).*

Musliner, D.; Durfee, E.; and Shin, K. 1993. Circa: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6).

Pell, B.; Bernard, D. E.; Chien, S.; Gat, E.; Muscettola, N.; Nayak, P. P.; Wagner, M.; and Williams, B. 1997. An autonomous spacecraft agent prototype. *Autonomous Robotics.*

Simmons, R. 1990. An architecture for coordinating planning, sensing, and action. In *Procs. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 292–297. San Mateo, CA: DARPA.

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical Artificial Intelligence* 7(1):197–227.