A Representation for Efficient Temporal Reasoning

James P. Delgrande and **Arvind Gupta**
School of Computing Science,
Simon Fraser University,
Burnaby, BC, V5A 1S6
Canada
email: {jim, arvind}@cs.sfu.ca

Abstract

It has been observed that the temporal reasoning component in a knowledge-based system is frequently a bottleneck. We investigate here a class of graphs appropriate for an interesting class of temporal domains and for which very efficient algorithms for reasoning are obtained, that of series-parallel graphs. These graphs can be used for example to model process execution, as well as various planning or scheduling activities. Events are represented by nodes of a graph and relationships are represented by edges labeled by $\leq$ or $<$. Graphs are composed using a sequence of *series* and *parallel* steps (recursively) on series-parallel graphs. We show that there is an $O(n)$ time preprocessing algorithm that allows us to answer queries about the events in $O(1)$ time. Our results make use of a novel embedding of the graphs on the plane that is of independent interest. Finally we argue that these results may be incorporated in general graphs representing temporal events by extending the approach of Gerevini and Schubert.

Introduction

It has been observed that in knowledge-based systems, planning systems, and the like, that the temporal reasoning component is frequently a severe bottleneck. The difficulty is that *scalability* is a problem: even if an algorithm requires, say, $O(n^2)$ time and/or space, such a bound is unacceptable for large databases, particularly if frequent use is to be made of such an algorithm. Consequently it is of interest to investigate not just efficient algorithms for temporal reasoning, but *very* efficient algorithms for such reasoning. An important and interesting problem, then, is concerned with identifying classes of problems for which very efficient algorithms exist.

In this paper, we investigate a class of graphs appropriate for an interesting class of temporal domains, and for which we have obtained very efficient algorithms for reasoning. In these graphs, events are represented by nodes and temporal relationships between events are represented by labeled edges. These graphs, which we call $(<, \leq)$-*series-parallel* graphs, are composed using a sequence of *series* and *parallel* steps (described later), where edges are directed and labeled by $<$ or $\leq$. We show that following a linear (in the size of the graph) time preprocessing step that we can answer queries concerning the temporal relationship between two nodes in constant time.

So there are two steps in dealing with these graphs: first there is a preprocessing step; second, queries are answered with respect to this processed information. Clearly, for the representation of temporal precedence in arbitrary graphs, one can store precedence information in an adjacency matrix – the difficulty is that the $O(n^2)$ space and preprocessing requirements do not permit large-scale applications. Consequently we require linear (or near linear) preprocessing time and storage requirements, with constant (or near constant) time query answering. We show that for the classes of graphs we consider, there is an $O(n)$ time preprocessing algorithm that allows us to answer queries about the order of events in $O(1)$ time, with the addition of constant space overhead. Therefore, query answering effectively becomes *information retrieval*, and so the approach can be looked on as compiling temporal information into a highly efficient representation.

In more detail we have the following. Our results centre on what we call $(<, \leq)$-*series-parallel* graphs. A series-parallel graph (VTL82) is perhaps best envisaged as comprising a qualitative temporal trace of process execution, where a process can overlay itself with another, or spawn subprocesses but must wait for all spawned processes to terminate before it can terminate. Slightly more formally (a formal definition is given in the Preliminaries to follow), a series parallel graph consists of a directed edge between two nodes,[1] or (recursively) some number of such graphs connected in series or in parallel (i.e. with common source and sink in the last case). A $(<, \leq)$-series-parallel graph is a series-parallel graph where edges are labeled either $<$ or $\leq$.

We show that every $(<, \leq)$-series-parallel graph can be embedded in the plane such that for nodes u and

[1] In practice a single node should also constitute a series parallel graph; however the development is simplified by omitting this trivial case.

v with coordinates (x_u, y_u) and (x_v, y_v), $u \leq v$ (i.e. there is a path from u to v) if and only if $x_u < x_v$ and $y_u < y_v$. We show that this embedding can be carried out in $O(n)$ time, where n is the number of nodes in the graph. Clearly, given the embedding of a graph, determining $\leq$ relations can be carried out in constant time. It is a bit more complex determining $<$ relations between time points. In this paper we show how the original embedding can be "perturbed" such that for points u and v where $x_u < x_v$ and $y_u < y_v$, if their co-ordinates are unperturbed then the relation is $\leq$; otherwise it is $<$.

These graphs are of independent interest: for example they model process execution, wherein if a process spawns others, it must wait for the spawned processes to terminate before it can terminate. In reasoning about the state of a database for example, temporal precedence of processes is required to determine which process might have last accessed a portion of memory. Similarly for planning systems and various scheduling problems: If a task or action must precede another, and that task is composed of some number of subtasks, then each of these subtasks must precede the second task. For example, if Pat wishes to go to the airport, and going there consists of getting ready, travelling to the airport, and checking in, then any subtasks involved in getting ready must necessarily finish before any involved with travelling or checking in can begin. That queries concerning strict and non-strict temporal precedence can be answered in constant time clearly would be useful for systems employing such structures.

While we argue that these results are of independent interest, we also indicate how they may be extended to arbitrary graphs, representing assertions in the point algebra (VK86), and from there to stronger systems. We present a brief synopsis suggesting how the approach may be used to extend that of (GS95). While this constitutes preliminary work, nonetheless it is easy to show that the approach of Gerevini and Schubert is ill suited to deal with series parallel graphs; moreover, even a naive incorporation of the present approach into theirs would represent a useful extension.

The next section briefly reviews related work. We then introduce notation, and give precise definitions of the problems considered in the paper. Following this is a linear time algorithm for preprocessing series-parallel graphs such that temporal precedence can be determined in constant time. We then present an outline of how this algorithm can be extended to deal with general graphs. Finally, we finish the paper with brief conclusions and open problems. Complete proofs are contained in the full paper.

Related Work

In temporal reasoning there is a fundamental choice between whether time points or time intervals are the primitive objects. In Artificial Intelligence, (All83) has proposed the *interval algebra* (IA) framework of temporal relations. Reasoning with this algebra (that is, reasoning about implied interval relations or determining the consistency of a set of assertions) however has been shown to be NP-complete (VK86). The *point algebra* (PA) is introduced in (VK86; VKvB90), based on the notion of a time point in place of an interval. The basic relations of the PA that can hold between two points are $<$, $=$, and $>$. Allowing the relation between two points to be a disjunction of the basic relations gives the set $\{<, \leq, >, \geq, =, \neq, \emptyset, ?\}$. The subset of the IA that can be translated into the PA is called the *pointisable interval algebra* (SIA). Finding a consistent scenario (i.e. interpretation) for a set of assertions in the PA and SIA takes $O(n^2)$ time for n points while computing the closure takes $O(n^4)$ time (vBC90; vB92). Again, these bounds are too big to permit large scale applications. Finally, (GS93) consider complexity characteristics of various restrictions of the IA.

Other approaches have attempted to provide good expected performance, rather than providing guaranteed bounds. (GM89) uses a spanning tree underlying a lattice of time points for achieving efficient indexing. Performance for retrieving and updating temporal relations is argued experimentally to be linear. (Dor92) develops the notion of a *sequence graph*, based on the observation that frequently in applications, processes, for example, will execute or recur sequentially. In sequence graphs, only "immediate" relations are stored. No information is lost, and the reduction in complexity is claimed to be significant. In the work of Schubert and collaborators (MS90; GS95) temporal reasoning is centred on *chains* of events. Assuming that temporal event histories are dominated by such chains, along with assertions between them, Gerevini and Schubert obtain efficient algorithms for reasoning. Reasoning within a chain takes constant time; reasoning between chains is less efficient, but is determined by a graph significantly smaller[2] than the original.

The work reported here can be considered as belonging to both groups of approaches described above. On the one hand we identify an interesting and useful class of graphs for which, using these graphs to represent temporal information, we can guarantee very efficient query-answering. On the other hand, we argue that this work can be used to extend existing general systems for (more) general temporal reasoning, with guaranteed improved performance.

Preliminaries

Graphs: Our results will rely substantially on graph theoretic concepts; we refer the reader to (BM76) for terms not defined here. The graphs that we use are simple, finite and directed. For a graph G, we denote the vertex and edge sets of G by $V(G)$ and $E(G)$ respectively. An edge $e \in E(G)$ from u to v is denoted

[2] if the original graph is dominated by chains.

by the tuple (u, v). Edges will generally be *labeled* with labels drawn from a finite set.

A *directed acyclic graph* (dag) is a directed graph with no directed cycles. For G a dag, G^{-1} is the graph obtained from G by reversing the direction of all edges. That is, $V(G^{-1}) = V(G)$ and $E(G^{-1}) = \{(u, v) \mid (v, u) \in E(G)\}$. For G a dag, every vertex v of G can be assigned a rank, $rank(v)$, as follows: The rank of the sources of G is 0. For any other vertex v with parents $w_1, \ldots, w_k$, $rank(v) = \max\{1 + rank(w_i) : 1 \leq i \leq k\}$. Notice that the rank of all vertices of a dag can be computed in $O(|E|)$ time using breadth-first search. Our algorithms will rely on computing variants of rank.

A *series-parallel graph* is a dag with source s and sink t, defined inductively as follows. A single edge $e = (s, t)$ is a series-parallel graph with source s and sink t called the *base graph*. Let G_1 and G_2 be series-parallel graphs with source and sink s_1, t_1 and s_2, t_2 respectively such that $V(G_1) \cap V(G_2) = \emptyset$. Then,

- the graph G constructed by taking the disjoint union of G_1 and G_2 and identifying s_2 with t_1 is a series-parallel graph with source s_1 and sink t_2 constructed using a *series step.*
- the graph G constructed by taking the disjoint union of G_1 and G_2 and identifying s_1 with s_2 (call this vertex s) and t_1 with t_2 (call this vertex t) is a series-parallel graph with source s and sink t constructed using a *parallel step.*
- no graph other than those constructed using the operations above is a series-parallel graph.

Fact 1 *For G a series-parallel graph,*

1. *G is acyclic with a single source and sink.*
2. *$|E(G)| \leq 2|V(G)|$, that is, the number of edges is linear in the number of vertices;*

We will rely on being able to decompose a series-parallel graph G into series and parallel steps in linear time. This decomposition is in terms of a tree, the *SP-decomposition tree* of G. Internal nodes of such a tree are labeled by either "*series*" or "*parallel*" and leaves are labeled by edges of G. With each node α of a SP-decomposition tree T, we will associate a subgraph of G; we call this the subgraph *induced by* α. A node α of T labeled by "*series*" has two children and the subgraph induced by α is formed by taking the subgraphs G_1 and G_2 induced by the children of α and combining them using a series step. Similarly, a node α labeled by "*parallel*", has two children where the the subgraph induced by α is formed by taking the subgraphs induced by the children of α and joining them in using a parallel step. The next result follows from results of (VTL82).

Lemma 1 *For G a series-parallel graph with two distinguished nodes s and t, a SP-decomposition tree of G can be constructed in linear time.*

Notice that for T a SP-decomposition tree of a series-parallel graph G, each edge of G appears at exactly one leaf of T. Furthermore, $|V(G)|$ is easily computed from T: For s the the number of nodes labeled "*series*", $|V(G)| = s + 2$. We will denote the quantity $s + 2$ by $\mathcal{N}(T)$. Furthermore, for α a node of T, we will denote the subtree of T rooted at α by T_α. Figure 1 illustrates a series-parallel graph with its corresponding SP-decomposition.

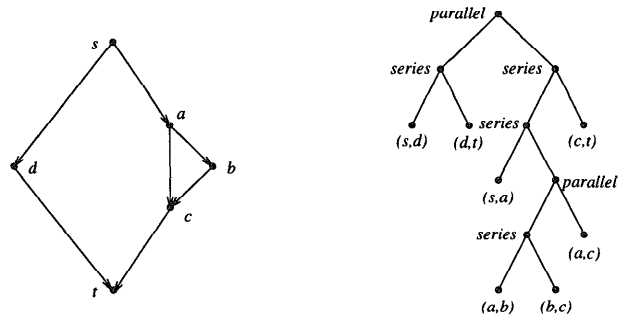

Figure 1: A series-parallel graph and its SP-decomposition tree.

Temporal Reasoning in Series-Parallel Graphs: To finish this section, we formally present the central problem under consideration in this paper. Let G be a dag with each edge labeled by one of $<$ or $\leq$; vertices of G represent a set of events with edge labels giving information about their relative order of occurrence. Let $L = [a, b]$ be a closed interval on the integer line; L is a domain of interpretation of G, which we call a *time window of G*. Here L represents the legal (integer) times at which the events in G could occur. We are interested in labelings of the vertices of G by elements of L such that the vertex labels are consistent with the inequalities induced by the edge labels, that is, labelings that satisfy the edge constraints. Formally, a *temporal labeling* of G over L is an integer-valued function $\ell : V(G) \rightarrow L$ such that for every edge $e = (u, v)$, if e is labeled by $<$ ($\leq$ respectively) then $\ell(u) < \ell(v)$ ($\ell(u) \leq \ell(v)$ respectively). For a time window $L = [a, b]$, we can assume, without loss of generality, that $a = 0$ since every labeling can be translated by $-a$.

Notice that the use of time windows results in algorithms that are more general than those obtained by other authors. In particular, time windows result in events being labeled by explicit time points; if we only want to allow implicit time points we only need make the time window sufficiently large.

Formally, the problem we are interested in is the following:

Name: Temporal reasoning.

Instance: G a dag with edges labeled by one of $\leq$ or $<$ and $L = [0, b]$ a closed interval on the integer line.

Problem: Preprocess G such that given any two

vertices u and v and a relation $R \in \{\leq, <\}$, there is a constant time procedure to determine whether $\ell(u) R \ell(v)$ for every temporal labeling ℓ of G over L.

There are two points to notice about our formal problem statement. First, in Allen's approach (All83), the relation R is not given but rather output (that is, given two points the relation R that holds between the two points is generated). We feel our approach results in a cleaner presentation since we can give a different algorithm for each relation. Furthermore, it is easy to use our algorithms to also solve the problem for the $=$ relation. However, our algorithms do not solve the problem for the $\neq$ relation.

A Note on the Model of Computation: We assume throughout that basic operations on small integers (of size $\log n$) can be performed in constant time and that such numbers take unit space for storage. This is a standard complexity-theoretic assumption – for example, in sorting algorithms, it is assumed that a comparison of two numbers is performed in constant time. This approach also consistent with other work in temporal reasoning. If a log-cost RAM model of computation is used, the complexity of our algorithms is increased by a factor of $\log \log n$.

An algorithm for series-parallel graphs

In this section we consider the temporal reasoning problem for series-parallel graphs. Our main result for this section is the following:

Theorem 1 *There is an $O(n)$ time preprocessing algorithm for the temporal reasoning problem on series-parallel graphs.*

To obtain the algorithm, we consider three subproblems:

1. For each vertex v of G determine

$$L(v) = \{\ell(v) : \ell \text{ is a temporal labeling of } G\}$$

in $O(n)$ time. We call $L(v)$ the *vertex-time window of v*.

2. Given a series-parallel graph G, generate a representation of G in $O(n)$ time so that in $O(1)$ time it can be determined whether there is a directed path from input vertex u to input vertex v.

3. Further process the representation from 2 so that arbitrary queries $Q(u, v)$ can be answered in $O(1)$ time.

Throughout the remainder of this section, we will assume that G is a series-parallel graph with one source s and one sink t and that $L = [0, b]$ is a time window.

We defer the proof of Theorem 1 to end of this section. We begin with the solutions to each of the three subproblems listed above.

Vertex Time Windows

Let G be a dag with every edge labeled by one of $<$ or $\leq$. Then, the *strict rank* of a vertex v, $srank_G(v)$, is the length of the shortest path from s to v where only edges labeled by $<$ are counted. Clearly, the strict rank of all vertices of G can be computed in $O(|E|)$ time using breadth-first search. We will write $srank(v)$ instead of $srank_G(v)$ when G is clear from context. We will also be interested in the strict rank of G^{-1}; we write $srank^{-1}(v)$ for $srank_{(G^{-1})}(v)$ when G is clear.

Since all edges of our graph are labeled by one of $<$ and $\leq$, it is clear that in any temporal labeling ℓ of G, $\ell(v)$ is a non-decreasing function along any directed path of G. For every $v \in V(G)$, let $a_v = \min_l\{l(v)\}$ and $b_v = \max_l\{l(v)\}$. Then $L(v)$ is the closed interval $[a_v, b_v] \subseteq L$. Also, $a_v = srank(v)$ and $b_v = b - srank^{-1}(v)$; we show that $\{L(v) : v \in V(G)\}$ can be computed in $O(|E|)$ time.

Theorem 2 *For G a series-parallel dag and L a time window, there is a linear time algorithm for determining whether there are any temporal labelings of G and if so, determining $L(v)$ for every vertex v of G.*

Proof. For $n = |V(G)|$ notice that $|E(G)| \in O(n)$. We can use breadth-first search to compute $srank(v)$ and $srank^{-1}(v)$ for every vertex v of G. If $srank(v) \leq b$ for every vertex v then at least one temporal labeling of G exists and $L(v) = [srank(v), srank^{-1}(v)]$. ∎

Determining paths quickly

In this section we give an $O(n)$ time representation of a series-parallel graph so that we can determine the existence of paths between arbitrary vertices in $O(1)$ time. Our techniques are partially inspired by work of Valdes et al (VTL82) who also use a geometric representation for another class of graphs.

Given a series-parallel dag G, we will assign to each vertex v of G a coordinate (x_v, y_v) on the integer plane such that for any other vertex w, there is a path from v to w if and only if $x_v < x_w$ and $y_v < y_w$.

For integers a_1, b_1, a_2, b_2 , $a_1 < a_2$ and $b_1 < b_2$, we call the set of (integer) points $\{(x, y) : a_1 \leq x \leq a_2, b_1 \leq y \leq b_2\}$ a $(a_1, b_1) \times (a_2, b_2)$-*box*. Then, given (a_1, b_1) and (a_2, b_2), our general strategy is to inductively (on the structure of G) solve the following problem: Assign coordinates inside the $(a_1, b_1) \times (a_2, b_2)$-box to all vertices of G such that the source s has coordinates (a_1, b_1), the sink t has coordinates (a_2, b_2), and a vertex u is an ancestor of a vertex v if and only if $x_u < x_v$ and $y_u < y_v$. We call this the $(a_1, b_1) \times (a_2, b_2)$-*embedding problem of G*. In general, for a graph G on n nodes, we will require a box of size $n \times n$, that is, $a_2 - a_1 + 1 = b_2 - b_1 + 1 = n$.

Algorithm: $(a_1, b_1) \times (a_2, b_2)$-embedding problem.
Inputs: $((a_1, b_1) \times (a_2, b_2), T)$ where $(a_1, b_1) \times (a_2, b_2)$ is a box with $a_2 - a_1 = b_2 - b_1 = \mathcal{N}(T) - 1$ and T is an SP-decomposition tree.

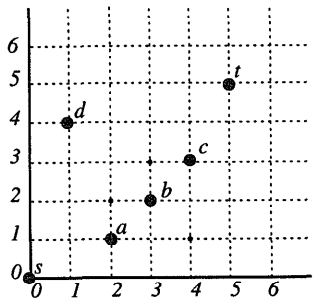

Figure 2: The planar embedding of the series-parallel graph given in Figure 1

1. let α be the root of T.
2. if $|V(T)| = 1$ then
2.1. let $e = (s, t)$ be the label on T.
2.2. assign (a_1, b_1) to s and (a_2, b_2) to t.
3. if α is labeled by "*series*" then
3.1 let β_1 and β_2 be the children of α.
3.2 let $a' = a_1 + \mathcal{N}(T_{\beta_1}) - 1$.
3.3 let $b' = b_1 + \mathcal{N}(T_{\beta_1}) - 1$.
3.4 solve $((a_1, b_1) \times (a', b'), T_{\beta_1})$.
3.5 solve $((a', b') \times (a_2, b_2), T_{\beta_2})$.
4. if α is labeled by "*parallel*" then
4.1 let β_1, β_2 be the children of α.
4.2 let $a' = a_1 + \mathcal{N}(T_{\beta_1}) - 1$.
4.3 let $b' = b_2 - \mathcal{N}(T_{\beta_1}) + 1$.
4.4 solve $((a_1, b') \times (a', b_2), T_{\beta_1})$.
4.5 let $a'' = a_2 - \mathcal{N}(T_{\beta_2}) + 1$.
4.6 let $b'' = b_1 + \mathcal{N}(T_{\beta_2}) - 1$.
4.7 solve $((a'', b_1) \times (a_2, b''), T_{\beta_2})$.
4.8 assign s the coordinates (a_1, b_1).
4.9 assign t the coordinates (a_2, b_2).

We initially call the algorithm to solve the $(0,0) \times (n-1, n-1)$-embedding problem on a SP-decomposition tree T of G, $n = |V(G)|$. Figure 2 illustrates the coordinates assigned by the algorithm to the graph given in Figure 1.

We must show that the coordinates assigned to each vertex are well-defined. This is straight-forward by induction except for the parallel step. Here, the subgraphs T_{β_1} and T_{β_2} are embedded so that their sources and sinks do not have the same coordinates yet the two sources (respectively sinks) are actually the same. This is handled by steps 4.8 and 4.9 of the algorithm where we assign new coordinates to these nodes.

Lemma 2 *If the above algorithm is initially called to solve the $(0,0) \times (n-1, n-1)$-embedding problem on T then for any subsequent call to the algorithm to solve the $(a_1, b_1) \times (a_2, b_2)$-embedding problem on a subtree T' of T, $a_2 - a_1 + 1 = b_2 - b_1 + 1 = \mathcal{N}(T')$.*

Lemma 3 *For G a series-parallel graph and T a SP-decomposition tree of G, let (x_u, y_u) and (x_v, y_v) be the coordinates associated with vertices u and v of G by the algorithm. Then, there is a path from u to v in G if and only $x_u < x_v$ and $y_u < y_v$.*

Proof. (outline) First suppose there is a path from u to v in G. Then either (u, v) is an edge or there are subgraphs G_1 and G_2 of G such that $u \in V(G_1)$, $v \in V(G_2)$ and G_1 and G_2 are joined in a series step. If the former, we can directly verify the result. If the latter, then if G_1 is embedded in a $(a_1, b_1) \times (a_2, b_2)$-box then G_2 is embedded in a $(a_2, b_2) \times (a_3, b_3)$-box where $a_1 < a_2 < a_3$ and $b_1 < b_2 < b_3$ and the result follows.

Conversely suppose that $x_u < x_v$ and $y_u < y_v$ but that u is not an ancestor of v. Then either v is an ancestor of u or u and v are incomparable. In the first case, it follows by the above argument that $x_v < x_u$ and $y_v < y_u$, a contradiction. In the second case, there must be subgraphs G_1 and G_2 of G such that $u \in V(G_1)$, $v \in V(G_2)$ and G_1 and G_2 are joined in a parallel step and u and v are not the source or sink of G_1 or G_2. Now, we can verify by structural induction on G that for all vertices u' of G_1 and v' of G_2 either $x_{u'} < x_{v'}$ and $y_{u'} > y_{v'}$ or $x_{v'} < x_{u'}$ and $y_{v'} > y_{u'}$, a contradiction. ∎

Theorem 3 *Let G be a series-parallel graph. Then, there is an $O(n)$ time algorithm for the temporal resoning problem restricted to $\leq$ relations.*

Proof. We begin by constructing the temporal time window $L(u) = [a_u, b_u]$ for all vertices u of G. By Lemma 1, we can construct a SP-decomposition tree T of G in $O(n)$ time and then use this in our embedding algorithm to assign coordinates to all vertices of G.

Let (x_u, y_u) be the coordinates of vertex u. Now answering a query $Q(u, v)$ is performed as follows: If $b_u < a_v$ then $\ell(u) < \ell(v)$ for all temporal labelings ℓ. Similarly, if $b_v < a_u$ then $\ell(v) < \ell(u)$ for all ℓ. Otherwise, $\ell(u) \leq \ell(v)$ for all temporal labelings ℓ if and only if u is an ancestor of v, that is, if and only if $x_u < x_v$ and $y_u < y_v$. Notice that the query is answered in $O(1)$ time. ∎

Answering arbitrary queries

In the previous section, we presented an algorithm for embedding a series-parallel graph on the plane that allows ancestor information to be computed in constant time. Here we augment that representation so that if there is a path from u to v we can also determine whether there is an edge on the path labeled by a "$<$".

The basic idea is to associate with each vertex u both a coordinate (x_u, y_u) as before and a line with equation $y = -x + c_u$ such that there is a path from a vertex u to a vertex v if and only if $x_u < x_v$ and $y_u < y_v$ and there is an edge on some u to v path labeled by "$<$" if and only if $y_u > -x_u + c_v$ and $y_v < -x_v + c_v$. Notice that in general, for vertex u, we need only specify the y-intercept c_u of the associated line. We can therefore

view this as associating the triple (x_u, y_u, c_u) with the vertex u.

For our algorithm, we will allow the coordinates (x_u, y_u, c_u) to lie at rational points; it is not difficult to subsequently translate these to integer coordinates. As well, once again when we are embedding a graph G with source s and sink t, we will associate a box B on the plane in which all the vertices must lie. With B we will associate three lines, call them l_1, l_2, l_3. Suppose $y = -x + c_i$ is the equation of line l_i. Then, $c_1 < c_2 < c_3$, l_1 will be below B, and l_2 and l_3 will pass through B. All vertices of G will be mapped to points in B that lie between l_2 and l_3. Furthermore, line l_1 will be associated with a vertex v of G if and only if all paths from s to v are only labeled by $\leq$.

Algorithm: Modified embedding problem.
Inputs: SP-decomposition tree T, box $B = (a_1, b_1) \times (a_2, b_2)$ such that $a_2 - a_1 = b_2 - b_1$, lines l_1, l_2, l_3 where l_i has equation $y = -x + c_i$, $c_1 < c_2 < c_3$ and l_2 and l_3 pass through B but l_1 is outside B.
Output: A triple (x_u, y_u, c_u) for every vertex u of the graph G induced by T.

1. let α be the root of T.
2. if $|V(T)| = 1$ then
2.1 let $e = (s, t)$ be the label on T.
2.2 assign s the coordinates (a_1, b_1, c_1).
2.3 if e is labeled by $\leq$ in G then
2.2.1. assign t the coordinates (a_1, b_1, c_1).
2.2.2. else assign t the coordinates (a_1, b_1, c_2).
3. if α is labeled by "*series*" then
3.1 let β_1 and β_2 be the children of α.
3.2 choose (a_3, b_3) in B such that $-a_3 + c_2 < b_3 < -a_3 + c_3$.
3.3 let l_3' be a line $y = -x + c_3'$ passing through $B_1 = (a_1, b_1) \times (a_3, b_3)$, $c_2 < c_3'$.
3.4 embed T_{β_1} in B_1 with lines l_1, l_2, l_3'.
3.5 let l_2' be a line $y = -x + c_2'$ passing through $B_2 = (a_3, b_3) \times (a_2, b_2)$, $c_2' < c_3$.
3.6 embed T_{β_2} in B_2 with lines l_3', l_2', l_3.
4. if α is labeled by "*parallel*" then
4.1 let β_1 and β_2 be the children of α.
4.2 choose (a_3, b_3) in B so that $-a_3 + c_2 < b_3 < -a_3 + c_3$.
4.3 let $B_1 = (a_1, b_3) \times (a_3, b_2)$ and $B_2 = (a_3, b_1) \times (a_2, b_3)$.
4.4 adjust l_2, l_3 so they pass through both B_i.
4.5 embed T_{β_1} in B_1 with lines l_1, l_2, l_3.
4.6 embed T_{β_2} in B_2 with lines l_1, l_2, l_3.
4.7 assign s coordinates (a_1, b_1, c_1).
4.8 let (a_3, b_2, c') and (a_2, b_3, c'') be the coordinates of t in B_1 and B_2.
4.9 assign t coordinates $(a_2, b_2, \min\{c', c''\})$.

For G the graph induced by α, let u be a vertex of G. If all paths from the source s to vertex u do not have any edges labeled by "<", then we choose coordinates (x_u, y_u, c_u) for u such that (x_u, y_u) lie between the lines l_1 and l_2 and $c_u = c_1$. Similarly, for all vertices v such that all paths in G^{-1} from t to v only involve edges labeled by "$\leq$", we assign coordinates (x_v, y_v, c_v) where $y_v > -x_v + c_3$ and assign $c_v = c_3$. Any other vertex w is given coordinates (x_w, y_w, c_2) where (x_w, y_w) lies between l_2 and l_3.

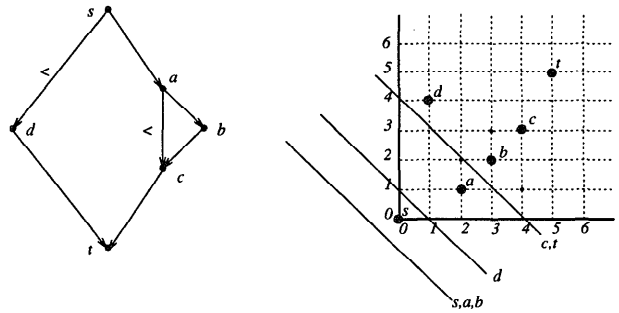

Figure 3: The planar embedding of one labeling of the series-parallel graph given in Figure 1. All unlabeled edges of the graph are assumed to be $\leq$. The sets of points associated with a particular line in the embedding is indicated on the line.

Notice that the algorithm does not specify, for example, how the point (a_3, b_3) is chosen. One simple method is to choose a point half way between the lines l_2 and l_3. Similarly, we can use any simple method of choosing the lines l_2' and l_3' in the series step. More difficult is the adjustment of l_2 and l_3 in step 4.4 of the parallel case. A simple method of making this adjustment is to move l_2 and l_3 closer together until they both lie in B_1 and B_2. However, if there are a sequence of parallel steps in the SP-tree, this will affect the coordinates assigned to other points outside of T_{β_1} and T_{β_2}. Instead, we do this embedding the entire sequence of parallel graphs at the same time. This then allows for a straightforward choice of l_2 and l_3.

Finally, we note that the above algorithm places coordinates at rational points instead of integer points. However, it is not difficult to see that since there are $O(n)$ different values of each coordinate, a common denominator can be found of size $O(n^3)$; a more careful study shows that $O(n^2)$ points suffice. Figure 3 illustrates an embedding of the graph of Figure 1 for a particular labeling of edges.

Proof of Theorem 1

Our proof of Theorem 1 relies on the following lemma whose proof is similar to that of Theorem 3.

Lemma 4 *Suppose G is a series-parallel graph with each edge labeled by either "<" or "$\leq$" and $L = [0, b]$ is a time window. Then, for u and v vertices of G, $\ell(u) < \ell(v)$ for every temporal labeling ℓ if and only if one of the following holds:*

1. *For $L(u) = [a_u, b_u]$ and $L(v) = [a_v, b_v]$, $b_u < a_v$; or*
2. *u is an ancestor of v and there is some directed path from u to v with an edge labeled by "<".*

Proof. (Theorem 1) We first solve the three subproblems outlined above in $O(n)$ time. To answer a query $Q(u, v)$ in $O(1)$ time, we check whether either of the two conditions in Lemma 4 holds. The first condition is easily checked once we have vertex time windows. For the second condition we need to check whether there is a path from u to v (the second subproblem) and if so whether there is a edge of this path labeled by "$<$" (the third subproblem). ■

Extending the Approach

In this section we indicate how the preceding results may be extended to arbitrary graphs, representing arbitrary assertions in the point algebra (VK86; VKvB90), and from there to stronger systems. This represents work in progress; however we argue that the direction and benefits of this extension are clear. Again, events are represented by nodes but relationships now are represented by directed edges labeled by $\leq$ or $<$, and undirected edges labeled $=$ or $\neq$. Our development borrows from (vB92) and (GS95). Given an arbitrary graph, we can efficiently eliminate $=$ relations by identifying the strongly-connected components. As well we can efficiently determine (so called) *implicit* $<$ relations and make these relations *explicit*. Lastly we can isolate $\neq$ relations, so that determining $\neq$ relationships can be accomplished by table lookup. So we obtain a graph where we have $<$ and $\leq$ edge relations only. Call the resultant graph the $(<, \leq)$-graph of the original.

The second part of this development borrows from (and extends) the *timegraph* approach of (GS95). In Gerevini and Schubert's (GS) approach, temporal reasoning is centred on *chains* of events. GS asume that temporal event histories are composed of such chains, along with assertions (cross-edges) between them. Reasoning within a chain is constant time; reasoning between chains is less efficient, but is determined (essentially) by the graph resulting from collapsing "runs" in the chains into single nodes, rather than the original graph.

In extending our results to arbitrary graphs, we generalise the chains of the GS approach to the more general $(<, \leq)$-series-parallel graphs. In the resulting structure, reasoning within a $(<, \leq)$-series-parallel graph is constant time; reasoning between such graphs is less efficient, but again is determined by the graph resulting from having the series-parallel subgraphs collapsed into single nodes, rather than the original graph. We argue that this this represents an improvement on the GS approach, for two reasons. First, the GS approach performs arbitrarily poorly on series-parallel graphs. For a series parallel graph with branching factor n, in the worst case in the GS approach $\frac{n-1}{n}$ of the edges are cross-edges. Second, since $(<, \leq)$-series-parallel graphs are essentially generalizations of chains, if we can replace time chains by (subsuming) series parallel graphs, then we would expect improved performance.

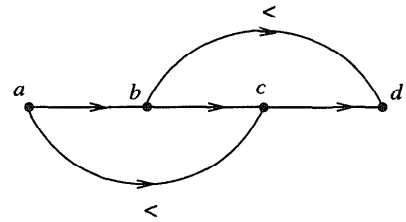

Figure 4: Forbidden subgraph for series-parallel graphs.

The simplest means of incorporating our approach into that of GS is to take a timegraph of GS and, wherever possible, merge time chains to form series parallel graphs. This possibility has the advantage that it is simple and straightforward and can only improve a GS timegraph; it has the disadvantage that it is ad hoc.

A second possibility is to decompose a $(<, \leq)$-graph into a set of maximal $(<, \leq)$-series-parallel subgraphs, connected by some number of cross-edges. Since the class of $(<, \leq)$-series-parallel subgraphs subsumes the class of time chains, this would represent a strict generalisation of the GS approach. There is one obstacle to this approach; by appeal to a result in (VTL82), a graph is a $(<, \leq)$-series parallel graph iff it does not contain the graph of Figure 4 as an induced subgraph (where unlabeled edges are $\leq$).[3]

Very briefly, we circumvent this difficulty as follows. For an arbitrary $(<, \leq)$-graph we consider only those edges (u, v) for which there is no other directed path from u to v. From this graph it is straightforward to isolate a number of vertex-disjoint maximal series parallel graphs in linear time. Edges not included in this set of maximal series parallel graphs are considered as cross-edges; they may be either within a series parallel subgraph, or between series parallel subgraphs. In either case an arbitrary node u is linked to these cross-edges as follows: Consider the set of ancestors of u where there is a path from u to that ancestor; where that ancestor is a vertex of a cross-edge; and where no nodes on that path is a vertex of a cross-edge.

1. If there is only a single such ancestor, link u directly to that ancestor.
2. Otherwise link u to the nearest node v with more than one incoming edge, that discriminates among these ancestors.

These cross-edges then are dealt with exactly as in the GS approach; moreover it is easily shown that this approach never generates more cross-edges (again, because series parallel graphs generalise chains).

[3] We note that this graph is one that GS handles easily.

Conclusions and Open Problems

We have shown that for a broadly interesting class of graphs, series parallel graphs, there is a highly efficient algorithm for determining temporal relations. Our preprocessing step requires linear time (as opposed to the standard $O(n^2)$ time algorithm for dags) with constant time required for answering queries. As well our representation allows us to handle updates efficiently.

Series parallel graphs are an instance of a broader class of graphs, which we have called elsewhere *local* graphs, and for which these results hold. Informally, a local graph is a dag in which nodes may be additionally ordered so that if u precedes v in this second order, then none of the descendants of v precede all those of u. This class includes, along with the series parallel graphs, edge parallel series graphs (VTL82), directed planar graphs, and as a subcase, threaded graphs, corresponding roughly to sets of intersecting chains. As well, local graphs constitute that maximal set of graphs for which the embedding described in this paper may be used for determining paths between nodes.

We are interested in extending this work in several directions. First, we are interested in studying the relation between classes of graphs for which efficient preprocessing and querying algorithms exist, and the class of general dags. As indicated, the results given here can be applied to general graphs so that we could obtain improved expected performance in querying general graphs. Nonetheless we have not fully worked out the details, nor has the full approach yet been implemented. We are also interested in identifying other classes of graphs for which efficient algorithms may be obtained, and relations among these classes. To this end we have obtained similar results for the class of *outer* graphs. Such graphs may be used to model two communicating agents, each of which can send messages to the other. Messages may take an arbitrarily long time to propagate from one agent to another. Again, the relative precedence of events can be determined in $O(1)$ time following $O(n)$ preprocessing.

Acknowledgements

We thank the reviewers for their careful reading of the manuscript and a number of useful suggestions. This work was supported by the Natural Sciences and Engineering Council of Canada. The second author also acknowledges the support of the British Columbia Advanced Systems Institute.

References

James Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(1):832–843, 1983.

J. Bondy and U.S.R. Murty. *Graph Theory with Applications*. North-Holland, 1976.

Jurgen Dorn. Temporal reasoning in sequence graphs. In *Proc. AAAI-92*, pages 735–740, 1992.

Malik Ghallab and Amine Mounir Alaoui. Managing efficiently temporal relations through indexed spanning trees. In *Proc. IJCAI-89*, pages 1297–1303, Detroit, 1989.

Martin Golumbic and Ron Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. *JACM*, 40(5):1108–1133, 1993.

Alfonso Gerevini and Lenhart Schubert. Efficient algorithms for qualitative reasoning about time. *Artificial Intelligence*, 74(2):207–248, April 1995.

S.A. Miller and L.K. Schubert. Time revisited. *Computational Intelligence*, 6:108–118, 1990.

Peter van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58(1-3):297–326, 1992.

Peter van Beek and Robin Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6(3):132–144, 1990.

Marc Vilain and Henry Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. AAAI-86*, pages 377–382, Philadelphia, PA, 1986. temporal reasoning.

Marc Vilain, Henry Kautz, and Peter van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1990.

Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11(2):298–313, May 1982.