Approximation Algorithms for Temporal Reasoning

Peter van Beek
Robin Cohen

Department of Computer Science
University of Waterloo

Research Report CS-89-12

May, 1989

Approximation Algorithms for Temporal Reasoning

Peter van Beek and Robin Cohen
Department of Computer Science
University of Waterloo
Waterloo, Ontario
CANADA N2L 3G1

ABSTRACT

We consider a popular representation for temporal relationships between intervals introduced by James Allen and its associated computational or reasoning problem of, given possibly indefinite knowledge of the relations between some intervals, computing the strongest possible assertion about the relations between some or all intervals. Determining exact solutions to this problem has been shown to be (almost assuredly) intractable. Allen gives an approximation algorithm based on constraint propagation. We present new approximation algorithms, examine their effectiveness, and determine under what conditions the algorithms are exact.

Keywords: temporal reasoning, constraint satisfaction, interval and point algebras.

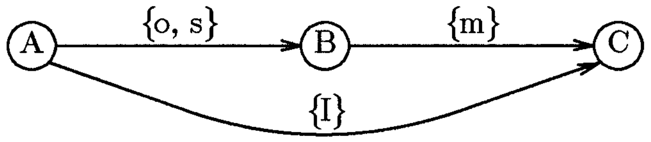
Table of Contents

1. Introduction

Allen (1983) gives an algebra for representing and reasoning about temporal relations between events represented as intervals. Possible application areas of the algebra include natural language processing (Allen, 1984; Song and Cohen, 1988), planning (Allen and Koomen, 1983; Hogge, 1987), and a part of a knowledge representation language intended for software development applications (Koubarakis et al., 1989). This algebra has been cited by others for its simplicity and ease of implementation with constraint propagation algorithms. The elements of the algebra are sets of the thirteen basic relations that can hold between two intervals.

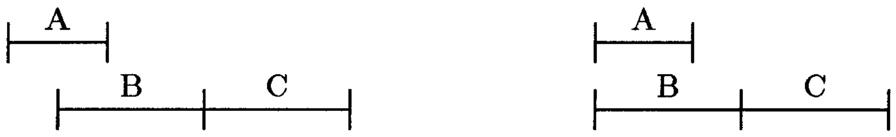
relation	symbol	converse	meaning
x before y	b	bi	xxx yyy
x meets y	m	mi	xxxyyy
x overlaps y	o	oi	xxx yyy
x during y	d	di	xxx yyyyy
x starts y	s	si	xxx yyyyy
x finishes y	f	fi	xxx yyyyy
x equal y	eq	eq	xxx yyy

There is a natural graphical notation where the vertices represent intervals and the directed edges are labeled with elements from the algebra representing the set of possible relations between the two intervals. Here is an example.



$$\begin{tikzcd} A \arrow[r, "\{o, s\}"] \arrow[rr, bend right, "\{I\}"'] & B \arrow[r, "\{m\}"] & C \end{tikzcd}$$

When the relationship between two intervals is ambiguous or indefinite we label the edge with the set of all the possible relations. So in our example, interval A either overlaps or starts interval B (but not both since the thirteen basic relations are mutually exclusive). Let {I} be the set of all basic relations, {b, bi, m, mi, o, oi, d, di, s, si, f, fi, eq}. The set of all possible labels on edges is $2^{\{I\}}$, the power set of {I}. Any edge for which we have no direct knowledge of the relationship is labeled with {I}; hence, the graphs are complete. Inference is done in this scheme through composition of relations: given a relation between A and B and between B and C we can compute a constraint on the relation between A and C. Doing this for our example we determine that our knowledge of the relationship between A and C can be strengthened to {b}. To see that this is true we show the two possible arrangements of the intervals along an imaginary time line.



A overlaps B in the diagram on the left, A starts B in the one on the right, and B meets C in both. We see that in both diagrams A is before C. Hence the result.

1.1. Statement of the Problem

Suppose we are given a set of events, represented as the intervals they occur over, and knowledge of the relationships between some of the intervals. The problem is to make explicit the strongest possible assertions about the relationships between intervals. We now make this somewhat more formal. Given is a directed graph with labels on the edges from the set of elements of the interval algebra. A *consistent singleton labeling* of the graph is a labeling where it is possible to map the intervals to a time line and have the single relations between intervals hold (as in the example above). The *minimal label* corresponding to a label consists of only the elements of that label capable of being part of a consistent singleton labeling of the graph. The problem then is to determine the minimal labels, removing only those elements from the labels that could not be part of a consistent singleton labeling. Call this the *minimal labeling problem (MLP).* Vilain and Kautz (1986) show that determining an exact solution to the MLP is NP-hard. This strongly suggests that no polynomial time algorithm exists.

Supposing that we still wish to solve instances of the problem, several alternatives present themselves:

- **Exponential algorithms:** Solve the problem exactly but devise efficient exponential algorithms. These may still be practical even though their worst case is exponential. Valdés-Pérez (1987) gives a dependency-directed backtrack algorithm but it only finds one consistent singleton labeling of the graph or reports unsatisfiability.
- **Easy special cases:** Interesting special cases of an NP-Hard problem may be solvable in polynomial time. This alternative often takes the form of limiting the expressive power of the representation language.
- **Approximation algorithms:** Solve the problem approximately using an algorithm that is guaranteed polynomial. That is, design algorithms that do not behave badly—in terms of the quality of the produced solution—too often, assuming some probabilistic distribution of the instances of the problem. Allen's (1983) O(n^3) algorithm is just such an approximation algorithm.

1.2. Overview

The purpose of this paper is to investigate the latter two alternatives: efficient algorithms for computing approximations to the minimal labeling problem and some special cases where approximation algorithms are exact. We consider two versions of the problem: an all-to-all version where we compute the minimal labels between every pair of intervals, and a one-to-all version where we determine the minimal labels between one interval and every other interval. Below we give an overview of our results.

In general, Allen's algorithm, being an approximation algorithm, will not always compute the minimal label between two intervals. In section 2, we explore better (and, unfortunately, more expensive) approximation algorithms. Allen's algorithm is a special case of path consistency algorithms for constraint satisfaction problems (Montanari, 1974; Mackworth, 1977). We develop an $O(n^4)$ consistency algorithm that computes a better approximation to the exact solution. The labels computed by the algorithm, as with Allen's algorithm, will always be a superset (not necessarily proper) of the minimal or true labels. The algorithm computes a better approximation in that there are fewer disjuncts that could not be part of a consistent singleton labeling of the graph.

In section 2, we also explore how far we must restrict the expressive power of the representation language to guarantee that we can solve instances of the all-to-all problem exactly in polynomial time. Valdés-Pérez (1986) shows that graphs with labels restricted to the thirteen basic relations can be solved exactly in $O(n^3)$ time using Allen's algorithm. Vilain and Kautz (1986) define a time point algebra and claim that Allen's algorithm is exact for computing the minimal labels between points. The consequences for the interval algebra are the following. Vilain and Kautz show that a subset of the interval algebra can be translated into the point algebra. If their claim is true we can solve that subset of the interval algebra exactly by first translating into the point algebra. However, their claim is false. We present a counter-example to their theorem. We show that Vilain and Kautz's point algebra and the subset of the interval algebra that can be translated into the point algebra can be solved exactly using the $O(n^4)$ consistency algorithm discussed above. We also characterize the subset of the point algebra and the interval algebra for which Allen's algorithm is exact. Unfortunately the subsets of the interval algebra are small. We must quite severely restrict our representation language to guarantee efficient and exact solutions.

Allen's algorithm determines an approximation to the minimal labels between every interval and every other interval (the all-to-all version of the problem). If we are only interested in the relationships between a few of the intervals then, in computing the relationships between all intervals, we may be doing too much work. In section 3, we define a one-to-all version of the problem, where we only determine the relationship between a single source interval and every other interval. Our solution to this version of the problem is an adaptation of Dijkstra's (1959) algorithm for computing the shortest paths in a graph. The algorithm produces good quality solutions for certain classes of instances of the problem and is shown to take $O(n^2)$ time. As in the previous section, we characterize how far we must restrict the expressive power of the representation language to guarantee that our one-to-all approximation algorithm is exact.

In section 4, we present the results of some computational experiments designed to test the quality of the solutions produced by Allen's and our approximation algorithms. We randomly generated instances of the problem and determined how often the less expensive approximate solutions differed from the exact solution, thus allowing us to determine with what degree of confidence we can rely on the less expensive solutions. We found that how well the approximation algorithms do is heavily dependent on the distribution from which the relations between intervals are randomly generated. In this section we also present a simple test for predicting when the approximation algorithms will and will not produce good quality approximations.

Finally, in section 5, we survey selected applications of the interval algebra with the intent of showing where the results of this paper will be of use.

2. The All-to-All Problem

The minimal labeling problem (MLP) is related to two well known problems: the algebraic path problem (Aho et al., 1974) and the constraint satisfaction problem (Montanari, 1974, Mackworth, 1977). Some of the algorithms and results developed for these related problems can also be applied to the MLP. We begin by relating the MLP to constraint satisfaction and show that Allen's algorithm is a special case of the path consistency algorithm for constraint satisfaction.

Constraint Satisfaction Problem (CSP)

Given are a set V of n variables $\{v_1, v_2, ..., v_n\}$, a domain D_i of possible values for each variable, and binary constraint relations between variables that preclude certain combinations of instantiations of the variables. A *consistent instantiation* of the variables in V is an n-tuple $(x_1, x_2, ..., x_n)$, representing an assignment of $x_i \in D_i$ to v_i, such that the constraint relations between variables are satisfied.

Valdés-Pérez (1986), Tsang (1987), and Ladkin (1988) show how an MLP can be viewed as a CSP. An interval is represented by its end points $<A^-, A^+>$ where $A^- < A^+$ and A^- and A^+ are real numbers. An interval is now a variable and its domain is the set of all ordered pairs of time points. A constraint between two variables is the label (the allowed relations) between those variables. We remark that in viewing an MLP as a CSP the only change is that the qualitative intervals are now seen as quantitative variables with associated domains of ordered pairs of real numbers. The domains, however, are implicit and need never be represented. The formulation as a CSP is then just a theoretical fiction but one that will ease subsequent proofs and allow us to use previously known algorithms.

Mackworth (1977) discusses approximation algorithms for the CSP, called consistency algorithms, that remove local inconsistencies that could never be part of a global solution. One, two, and three consistency are generally referred to as node, arc, and path consistency, respectively. Freuder (1978) generalizes this to k-consistency: For every choice of $k-1$ variables along with instantiations for each that satisfy all the constraints among them, the domains are *k-consistent* if, for any choice of a kth variable there exists an instantiation of that variable such that all the constraints between the k variables hold. Freuder (1982) defines strong k-consistency as j-consistent for all $j \leq k$. Strong k-consistency is equivalent to ensuring that, for every choice of k of the n variables, every pair of values permitted by a direct constraint is capable of being part of a consistent instantiation of the k variables.

How do strong k-consistency and the consistency algorithms relate to the original MLP? The following lemma connects the two.

Lemma 1. *If the underlying CSP of an MLP is strongly k-consistent then in the MLP, for every choice of k vertices, every element of the associated labels is capable of being part of a consistent singleton labeling of the subgraph of k vertices. In particular, strongly n-consistent in the CSP corresponds to a minimal labeling in the MLP.*

Proof. Strong k-consistency means that, for every choice of k variables, every pair of values permitted by a direct constraint is capable of being part of a consistent instantiation of the k variables. But a consistent instantiation of the k variables is exactly a consistent singleton labeling of the subgraph of k vertices: the instantiation of the variables is our mapping of the intervals to a time line, the labels between variables are singletons sets because the basic relations between intervals are mutually exclusive, and by definition the relations are satisfied.

2.1. The Path Consistency Algorithm

In this section we present Mackworth's (1977) path consistency algorithm with some small simplifications because of the properties of the interval algebra. The resulting algorithm is equivalent to Allen's (see Figure 1).

The algorithm works as follows (a more detailed description can be found in Allen, 1983 and Mackworth, 1977). We represent the graphs of intervals and their relationships as a matrix where element C_{ij} is the label on edge (i, j). Procedure RELATED PATHS, given an edge (i, j), returns a set of triples representing all the paths of length two in which edge (i, j) participates. The labels on these paths of length two potentially constrain the label on the third edge that completes the triangle. We maintain a queue of triples that still need to be processed. Each time through the loop we process a triple. If the path of length two does constrain the third edge we update the entry. This updated edge may further constrain other edges so its set of RELATED PATHS is added to the queue. But note that only those triples not already in the queue are added. How a triple is selected doesn't change the result (Mackworth, 1977, p. 113). It does, however, influence the amount of work done. In practice, sorting the triples at the start in ascending order according to their labels and adding new triples to the front of the queue works well. We defer until section 3 a discussion about

Input: A matrix C where element C_{ij} is the label on edge (i, j). *Output:* A path consistency approximation to the minimal labels for C_{ij}, $i, j = 1,...,n$.

```
procedure ALL3
begin
    $Q \leftarrow \bigcup_{1 \le i < j \le n}$ RELATED PATHS $(i, j)$
    while $Q$ is not empty do begin
        select and delete a path $(i, k, j)$ from $Q$                (2.1)
        $t \leftarrow C_{ij} \cap C_{ik} \cdot C_{kj}$                (2.2)
        if $(t \ne C_{ij})$ then begin
            $C_{ij} \leftarrow t$
            $C_{ji} \leftarrow$ CONVERSE $(t)$
            $Q \leftarrow Q \cup$ RELATED PATHS $(i, j)$
        end
    end
end

procedure RELATED PATHS $(i, j)$
    return $\{ (i, j, k), (k, i, j) \mid 1 \le k \le n, k \ne i, k \ne j \}$
```

Figure 1. An All-to-All Path Consistency Algorithm (Mackworth, 1977).

defining an order over the interval algebra.

Theorem 1 (Montanari, 1974; Mackworth and Freuder, 1985). *The algorithm in Figure 1 achieves path consistency and requires $O(n^3)$ time.*

To use the path consistency algorithm we need to define the operations of intersection and composition of two labels (equation 2.2 of Figure 1). Intersection is just set intersection. Given that labels on edges can represent a disjunction of possible relations between two intervals, Allen defines the composition of two labels as the pair-wise multiplication of the elements,

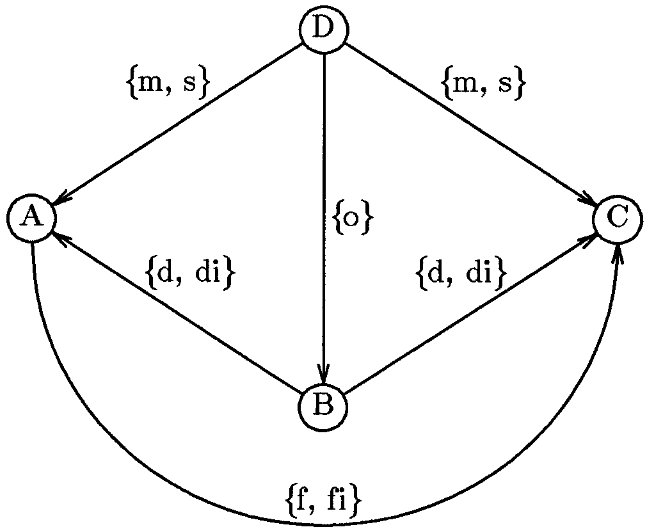
$$C_{ik} \cdot C_{kj} \Leftrightarrow \{ a \times b \mid a \in C_{ik}, b \in C_{kj} \} \qquad (2.3)$$

where $\times$ is defined over the seven basic relations and their converses and is easily implemented as a table lookup (see appendix A for the complete table taken from Allen, 1983).

2.2. Improving the Approximation

In this section we explore better and more expensive algorithms for determining approximations to the minimal labels (the strongest possible assertions about the relationships) between all intervals.

Where does the path consistency algorithm fail? Determining this will help us develop better approximation algorithms. It can be shown that for the interval algebra path consistency ensures that for the triangle formed by any three vertices in the graph, the labels on edges are locally minimal—minimal not necessarily with respect to the entire graph, but with respect to the triangle. In the interval algebra this is insufficient to even guarantee overall consistency. We give an example from Allen (1983) ascribed to Henry Kautz.



Applying the algorithm of Figure 1 results in no changes to the labels; each of the elements can be part of a consistent singleton labeling of the triangles to which the edges belong. However, no consistent singleton labeling exists for the entire graph. The minimal labels are the empty set.

Recall the definition of a minimal label: every element of the label is capable of being part of a singleton labeling of the entire graph that can be consistently mapped to a time line. The path consistency algorithm, as an approximation, ensures the labels are minimal with respect to all subgraphs of size three. Another way to view the action of equation (2.2) together with the definition of composition of labels (equation 2.3) is that entry C_{ij}, the label on the edge (i, j), gets updated to be the set of the elements of the old label that can be part of a consistent singleton labeling of the triangle (i, k, j). That is, we ensure that the labels are minimal with respect to all triangles (or 3-cliques) and composition is defined over labels on edges that share a vertex. The simple idea for improving the approximation is then to ensure that the labels are minimal with respect to all subgraphs of four vertices (or 4-cliques) and define composition over labels of triangles that share an edge. Equation (2.2a) and the definition of composition over triangles that share an edge (equation 2.3a) ensure that entry C_{ij} gets

```
procedure ALL4

        select and delete a 4-tuple $(i, k, l, j)$ from $Q$          (2.1a)
        $t \leftarrow C_{ij} \cap \Delta_{ikl} \cdot \Delta_{klj}$          (2.2a)

procedure RELATED PATHS $(i, j)$
    return $\{ (k, i, j, l) \mid 1 \leq k < l \leq n,\ k, l \neq i, j \} \cup$
        $\{ (i, j, l, k), (k, l, i, j) \mid 1 \leq k, l \leq n,\ k \neq l,\ k, l \neq i, j \}$
```

Figure 2. An All-to-All Four-Consistency Algorithm. Shown are changes to the path consistency algorithm.

updated to be the set of the elements of the old label that can be part of a consistent singleton labeling of the subgraph of four vertices.

$$\Delta_{ikl} \cdot \Delta_{klj} \Leftrightarrow \{ (a \times d) \cap (b \times e) \mid a \in C_{ik}, c \in C_{kl}, e \in C_{lj}, \quad b \in (a \times c) \cap C_{il}, d \in (c \times e) \cap C_{kj} \} \tag{2.3a}$$

Procedure RELATED PATHS must also be altered. Instead of returning all paths of length two in which edge (i, j) participates it now must return all structures of four vertices in which the edge participates, taking into account symmetries to prevent redundant computation. The necessary changes to the algorithm of Figure 1 are summarized in Figure 2.

Theorem 2. *Procedure* All4, *the path consistency algorithm with the changes of Figure 2, achieves three and four consistency and requires $O(n^4)$ time.*

The idea for developing the initial better approximation algorithm can be generalized to develop successively more expensive algorithms that compute progressively better approximations. But this is of theoretical interest only since higher orders of consistency quickly become impractical for all but the smallest problems.

2.3. Easy (Polynomial Time) Special Cases

In this section we explore how far we must restrict the expressive power of the representation language to guarantee that we can compute exact solutions in polynomial time.

Montanari (1974) shows that for the general CSP the path consistency algorithm is exact for a restricted class of binary constraint relations. However, the relations of interest here do not all fall into this class. Valdés-Pérez (1986) shows that graphs that are not labeled with disjunctions can be solved exactly in $O(n^3)$ time using the path consistency algorithm.

Vilain and Kautz (1986) claim something stronger. They define a time point algebra for representing and reasoning about the possible relations between points, as opposed to intervals. Let $PA^{\neq}$ denote Vilain and Kautz's point algebra. $PA^{\neq}$ is the algebraic structure with underlying set $\{<, \leq, =, >, \geq, \neq, ?\}$ and binary operators intersection and composition. Note that $\leq$, for example, is an abbreviation of $\{<, =\}$ and $?$ means there is no constraint between two points, $\{<, =, >\}$. Intersection is then set intersection. Composition is defined as in the interval algebra (equation 2.3) except that multiplication is now given by,

$\times$	$<$	$=$	$>$
$<$	$<$	$<$	$?$
$=$	$<$	$=$	$>$
$>$	$?$	$>$	$>$

Vilain and Kautz show that a subset of the interval algebra can be translated into this time point algebra. As an example translation, the interval algebra label

$$\begin{tikzcd} A \arrow[r, "\{d, o, m\}"] & B \end{tikzcd}$$

translates into the following inequalities on the endpoints of the intervals (where A^- and A^+ represent the start and end points of interval A, respectively)

$$\begin{array}{ccccccccc} A^- & \neq & B^- & A^- & < & B^+ & A^+ & \geq & B^- \\ A^+ & < & B^+ & A^- & < & A^+ & B^- & < & B^+ \end{array}$$

Let $SP^{\neq}$ be the set of labels in the interval algebra that can be translated into relations between the endpoints of the intervals using the underlying set of $PA^{\neq}$ (see appendix B for an enumeration of $SP^{\neq}$).

Vilain and Kautz assert (Theorem 4, p. 380) that the path consistency algorithm (Figure 1) is exact for computing the minimal labels between points. The consequences for the interval algebra are the following. If their claim is true we can solve the subset $SP^{\neq}$ of the interval algebra exactly by first translating into the point algebra. However, their claim is false. Here we present a counter-example demonstrating that the path consistency algorithm is not exact for Vilain and Kautz's point algebra. The counter-example also shows that path consistency is not exact for $SP^{\neq}$ if, instead of first translating into the point algebra, we use the interval algebra representation directly. Below is the interval algebra representation of the example with labels chosen from $SP^{\neq}$.

$$\begin{tikzcd}
 & D \arrow[dl, "L"'] \arrow[dr, "{\{b, d, o, s\}}"] \arrow[dd, "{\{b, d, o, f, fi\}}"] & \\
A \arrow[rr, bend right=90, "{\{eq, b, di, o, s, si, fi\}}"'] & & C \\
 & B \arrow[ul, "L"] \arrow[ur, "L"'] &
\end{tikzcd}$$

where

L = {d, di, o, oi, m, f, fi}

The translation into the point representation is the following

	A^-	A^+	B^-	B^+	C^-	C^+	D^-	D^+
A^-	$=$	$<$	$\neq$	$\leq$	$\leq$	$<$	$\neq$	$\leq$
A^+	$>$	$=$	$>$	?	$\neq$	?	$>$	?
B^-	$\neq$	$<$	$=$	$<$	$\neq$	$<$	$\neq$	$\neq$
B^+	$\geq$	?	$>$	$=$	$\geq$	?	$>$	$\geq$
C^-	$\geq$	$\neq$	$\neq$	$\leq$	$=$	$<$	?	$\neq$
C^+	$>$	?	$>$	?	$>$	$=$	$>$	$>$
D^-	$\neq$	$<$	$\neq$	$<$	?	$<$	$=$	$<$
D^+	$\geq$	?	$\neq$	$\leq$	$\neq$	$<$	$>$	$=$

Applying the algorithm of Figure 1 results in no changes; the relations between points are all considered to be minimal. However, the relation $A^- \leq B^+$ is not minimal, thus demonstrating that the algorithm is not exact for the point algebra. The minimal or true relation is $A^- < B^+$. This change is also reflected in the original interval algebra representation: the minimal label between vertex A and vertex B is {d, di, o, oi, f, fi}, with the meets relation having been dropped because it could not participate in any consistent singleton labeling of the graph. Interestingly, the path consistency algorithm is also not exact when applied to the interval algebra representation of this example, whereas the algorithm we proposed in the previous section computes the minimal labeling. To reiterate, the counter-example shows that the path consistency algorithm is not exact for $SP^{\neq}$ and $PA^{\neq}$. Ladkin and Maddux (1988) show that for $PA^{\neq}$ path consistency does guarantee that, if the minimal labels on edges is the empty set, this will be detected.

Define a new point algebra, PA, with the same binary operators and underlying set as $PA^{\neq}$ with the exception that $\neq$ is excluded from the underlying set. Let SP be the set of labels in the interval algebra that can be translated into relations between the endpoints of the intervals using the underlying set of PA (see appendix B for an enumeration of SP). We next prove that the path consistency algorithm is exact for SP. A corollary shows that path consistency is exact for PA as well. The following lemma on the intersection of convex sets will be useful in the proof of exactness.

Lemma 2 (Helly's theorem; ref. Chvátal, 1983). *Let F be a finite family of at least $n+1$ convex sets in R^n such that every $n+1$ sets in F have a point in common. Then all the sets in F have a point in common.*

Theorem 3. *The path consistency algorithm of Figure 1 is exact if all labels are chosen from SP.*

Proof. The theorem is proved by showing that if all labels are from SP and there is path consistency then the graph is strongly k-consistent for all $k \leq n$. Hence, the graph is strongly n-consistent and by Lemma 1 the labeling is the minimal or exact labeling.

Basis: $k = 1$, 2, or 3. True by construction of CSP and by the assumption of path consistency.

Inductive step: We assume strongly $(k-1)$-consistent and show k-consistent. The domain of variable V_i is the set of ordered pairs of points $<V_i^s, V_i^e>$ with $V_i^s < V_i^e$. The inductive assumption implies that variables $V_1, \ldots, V_{k-1}$ can be consistently instantiated. That is, they can be given values $<s_1, e_1>, \ldots, <s_{k-1}, e_{k-1}>$ that satisfy relations C_{ij}, $i, j = 1,\ldots,k-1$. We wish to show that variable V_k can be given a value that, together with the previous values, satisfies relations C_{ij}, $i, j = 1,\ldots,k$. Restating the assumption,

$$<s_i, e_i> \; C_{ij} \; <s_j, e_j> \qquad i, j = 1,\ldots,k-1$$

At completion of the algorithm we have path consistency so every instantiation of variables V_i and V_j allowed by the direct relation between V_i and V_j is also allowed by the composition of relations along every path between V_i and V_j. And, in particular, is allowed by the path through variable V_k. So, for each $<s_i, e_i>$ and $<s_j, e_j>$ there exists instantiations of V_k, dependent on the instantiations of V_i and V_j, such that the relations hold.

$$<s_i, e_i> \ C_{ik} \ <s_k, e_k>_{ij} \ C_{kj} \ <s_j, e_j> \qquad i, j = 1,...,k-1 \qquad (2.4)$$

We must show that there exists at least one instantiation of V_k that makes all relations true. We do so as follows. View the constraints between variables (C_{ik} and C_{kj}) as being represented by their translation into the point algebra. Then, for each i, j in equation (2.4) we get bounds on instantiations of V_k^s and V_k^e. The key is that the bounds on the end points are convex sets so by Lemma 2 it is sufficient to show that any three bounds have a point in common. There are two cases depending on whether one of the three bounds is $V_k^s < V_k^e$.

Case 1: Each of the three bounds is strictly in one or the other of V_k^s and V_k^e, the bound that involves both is not included. Because each bound is only in one variable it is sufficient to show they are pair-wise consistent to show that together the three are consistent. But any two bounds are always part of a single triangle and are consistent by the assumption of path consistency.

Case 2: Two of the bounds are strictly in one or the other of V_k^s and V_k^e, the third bound is $V_k^s < V_k^e$. In this case, all three bounds are always part of a single triangle and again are consistent by the assumption of path consistency.

Hence, there exists at least one instantiation of V_k that makes equation (2.4) true for all i and j and we have shown the graph is k-consistent.

$$<s_i, e_i> \ C_{ij} \ <s_j, e_j> \qquad i, j = 1,...,k$$

Adding the inductive assumption gives the desired result.

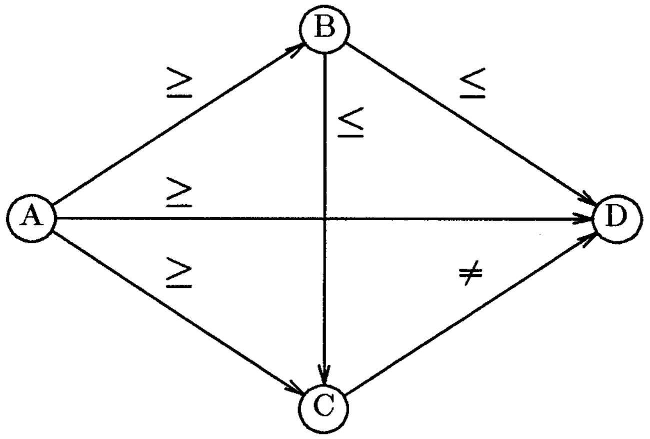
Corollary 1. *The path consistency algorithm of Figure 1 is exact for PA, the time point algebra that excludes the $\neq$ relation.*

Proof. The proof is similar. Here we need only show that the intersection of any two bounds is non-empty and this follows directly from path consistency.

Thus, any algorithm that achieves path consistency is exact for *SP* and *PA*. Aho et al. (1974) give an algorithm for the algebraic path problem that achieves path consistency if certain conditions hold. *SP* and *PA* can be shown to meet these conditions with one provision. Composition does distribute over intersection *provided* the intersection of two elements is not equal to the empty set (proposition 1 & 2 of section 3.2). Clearly, this will not always be true. Does this restrict the applicability of the algorithm? Fortunately not. In the algorithm, the intersection of two elements to get the empty set means the graph is inconsistent. If this occurs the algorithm can stop and report inconsistency.

In the remainder of this section we show that the four consistency algorithm developed in section 2.2 is exact for $SP^{\neq}$ and $PA^{\neq}$. The strategy is to first identify why path consistency is not sufficient and where the proof of Theorem 3 fails for $SP^{\neq}$ and $PA^{\neq}$ (recall $PA^{\neq}$ includes $\neq$, *PA* doesn't).

In the proof of Theorem 3 for the exactness of path consistency for *SP* and *PA* the inductive step showed that if $k-1$ of the variables were consistently instantiated then, for any choice of a kth variable, that variable could be instantiated such that all k variables together were consistently instantiated. Showing this relied on the fact that the bounds on the instantiations of the kth variable were convex sets. If $\neq$ is permitted in the language of the point algebra, the bounds are no longer convex sets and while the intersection of any two bounds cannot be empty, the intersection of three can be empty. Here is an example. The example is also the smallest counter-example to the exactness of path consistency for $PA^{\neq}$ and, up to isomorphism, is the only counter-example of four vertices.



The graph is path consistent. Let A, B, and C be instantiated as a, b, and c such that a = b = c. The instantiation is consistent. The bounds on the instantiation of D are D $\leq$ a, b $\leq$ D, and D $\neq$ c. Using standard interval notation and substitution of equals the bounds are $(-\infty, a]$, $[a, +\infty)$, and $(-\infty, a) \cup (a, +\infty)$. It is easily seen that the bounds are pair-wise consistent but together are inconsistent. If the graph was also four consistent, say by applying algorithm All4, the label between A and B would be $>$ and this counter-example could not occur.

The counter-example then is unique for $n = 4$ and cannot occur if the graph is three and four consistent. But can we find a counter-example for $n > 4$? No. Any larger counter-example must take $k > 3$ sets and show their intersection is empty. The counter-example must, of course, involve $\neq$. Suppose we have any two and any three bounds consistent but there exists a $k > 3$ such that k sets taken together do not have a point in common. The disjoint case can be dismissed immediately because some pair would also be disjoint. The bounds on instantiations of variables and their intersections are *almost* convex: except for at most n holes. So, for the intersection to be empty we must have one or more of the bounds assert $\neq$ and the intersection of two or more sets be exactly a point, that point being a hole. But if the intersection of a finite number of intervals is a point then some two of them must also intersect to be a point. But this case is ruled out. We have just shown the following.

Theorem 4. *The three and four consistency algorithm of Figure 2 is exact if all labels are chosen from $SP^{\neq}$.*

Proof. The proof of theorem 4 follows the inductive proof of theorem 3 except that we can no longer rely on Lemma 2 and the convexity of the bounds to show the bounds are consistent. But, by the above discussion, the bounds are consistent if the graph is path consistent and also four consistent.

Corollary 2. *The three and four consistency algorithm of Figure 2 is exact for $PA^{\neq}$, the time point algebra that includes the $\neq$ relation.*

We characterized the subsets of the interval algebra for which the path consistency algorithm and our four consistency algorithm are exact. Unfortunately these subsets are small. We must quite severely restrict our representation language to guarantee efficient and exact solutions.

3. The One-to-All Problem

The algorithms given in the previous section compute approximations to the minimal labels between every interval and every other interval (the all-to-all version of the problem). If we are only interested in the relationships between one interval and every other interval or between two particular intervals then, in computing the relationships between all intervals, we may be doing too much work. In this section we present an efficient algorithm for the one-to-all version of the problem and show that the algorithm is exact for a useful subset of the interval algebra and of the point algebra.

3.1. A One-to-All Approximation Algorithm

The algorithm (see Figure 3) is an adaptation of Dijkstra's (1959) algorithm for computing the shortest path from a single source vertex s to every other vertex. The algorithm maintains a list, L, of vertices to be processed that have not yet had their labels fixed. Each time through the while loop we choose a vertex, v, from L such that the label on the edge (s, v) is a minimum and use the label to update the remaining unfixed labels. In Dijkstra's algorithm this minimum label is now considered fixed. His algorithm, which can be categorized as a label-setting algorithm, produces poor quality solutions when applied to the interval algebra.

In the algorithm of Figure 3, a label is allowed to change after it has been tentatively fixed and perhaps further constrain other labels. This is accomplished through two simple changes to Dijkstra's algorithm: (i) the for loop now cycles through all vertices, V, rather than just through the unfixed vertices and (ii) a vertex is added to L if its edge label changes. These changes turn the algorithm into a label-correcting algorithm where no labels are considered final until the procedure halts. These changes to Dijkstra's algorithm also appear in Edmonds and Karp (1972) in the context of finding shortest paths where negative arc lengths are allowed. Johnson (1973) showed that, if the labels are integers, these changes makes the algorithm exponential in the worst

Input: A source vertex s and a matrix C where entry C_{ij} is the label on edge (i, j).
Output: An approximation to the minimal labels for C_{sj}, $j = 1,...,n$.

```
procedure ONE3
begin
    $L \leftarrow V - \{ s \}$
    while $L$ is not empty do begin
        select a vertex $v$ from $L$ such that $C_{sv}$ is a minimum
        $L \leftarrow L - \{ v \}$
        for each $t$ in $V$ do begin
            $l \leftarrow C_{st} \cap C_{sv} \cdot C_{vt}$
            if ($l \neq C_{st}$) then begin
                $C_{st} \leftarrow l$
                $L \leftarrow L \cup \{ t \}$
            end
        end
    end
end
```

Figure 3. A One-to-All Algorithm.

case. In this context, though, the algorithm is O(n^2).

Theorem 5. *The label-correcting algorithm of Figure 3 requires O(n^2) time.*

Proof. Initially our free list, L, is all the vertices. A vertex, t, is put back on the free list only if the label on edge (s, t) loses one or more of its elements. A label can have at most 13 elements initially, so each vertex can reappear on the free list at most 13 times. For each element in L we do O(n) work. Hence O(n^2).

The label-correcting algorithm requires the operation of finding the minimum of a set of labels. However, the final result of the algorithm is independent of the minimum used, so a minimum function that simply returned the first element in the list would suffice. But the choice of label does affect the number of iterations. We found that in practice the following order on the set of all labels halved the number of iterations of the algorithm compared to a random minimum. We assign weights to the 13 primitive relations based on the sum of the cardinality of the primitive relation successively composed with every possible label. The weight of a primitive relation is a measure of how restrictive the relation is. With suitable scaling we get the following weights for the 13 primitives: (1, eq), (2, fi), (2, f), (2, mi), (2, m), (2, si), (2, s), (3, bi), (3, b), (3, di), (8, d), (8, oi), (8, o). The weight of a label is then the sum of the weights of its elements. This same weighting was also found to be useful for pre-sorting the labels in the all-to-all algorithm of Figure 1.

3.2. Easy (Polynomial Time) Special Cases

In this section we explore how far we must restrict the expressive power of the representation language to guarantee that our one-to-all approximation algorithm of Figure 3.1 (One3) is exact. Note that the all-to-all algorithms compute approximations to all the minimal labels but even the labels we're not interested in help us by further constraining the labels we are interested in. One3 does not do this; it uses less information to compute its approximations. Hence, in general its approximations are poorer than those of the all-to-all algorithms. Surprisingly though, One3 is exact for the same subset of the interval algebra for which the path consistency algorithm (All3) is exact. First we state some of the properties of *SP*, the subset of the interval algebra for which All3 is exact, useful in the proof of exactness.

Proposition 1. *The following properties of SP are easily verified:*

(i) *SP is closed under composition.*

(ii) *SP is closed under intersection if we include the empty set or null relation.*

(iii) *Composition distributes over intersection, that is,* $a \cdot (b \cap c) = a \cdot b \cap a \cdot c$ *and* $(b \cap c) \cdot a = b \cdot a \cap c \cdot a$*, for all a, b, c in SP such that* $b \cap c \neq \emptyset$

Theorem 6. *The label-correcting algorithm of Figure 3 is exact if all labels are chosen from SP, provided there exists at least one consistent singleton labeling of the graph.*

Proof. We will prove that, for those labels computed by One3, the results are equivalent to those of All3. Then, since All3 is exact by Theorem 3, so is One3.

Let C_{sj}, $j = 1,...,n$ be the labels computed by One3 with source s. At completion of the algorithm the following is true,

$$C_{sj} \subseteq C_{sk} \cdot C_{kj}, \quad j, k = 1,...,n \tag{3.1}$$

Suppose, to the contrary, that there exists a C_{st} such that All3 computes a better approximation than One3. We know that,

$$C_{st} \subseteq C_{sv} \cdot C_{vt}, \quad v = 1,...,n \tag{3.2}$$

For such a C_{st} to exist, there also must exist some path $v, w_1, w_2, ..., w_m, t$ that the algorithm doesn't look at and that constrains the label on the edge (v, t) and invalidates equation (3.2). That is, a path such that

$$C_{sv} \cdot (C_{vw_1} \cdot C_{w_1 w_2} \cdot \cdots \cdot C_{w_m t} \cap C_{vt}) \subset C_{st} \subseteq C_{sv} \cdot C_{vt}$$

But by distributivity we have,

$$\text{l.h.s.} = C_{sv} \cdot C_{vw_1} \cdot C_{w_1w_2} \cdot \cdots \cdot C_{w_mt} \cap C_{sv} \cdot C_{vt}$$

By associativity,

$$= (((C_{sv} \cdot C_{vw_1}) \cdot C_{w_1w_2}) \cdot \cdots \cdot C_{w_mt}) \cap C_{sv} \cdot C_{vt}$$

Applying equation (3.1) repeatedly,

$$\supseteq C_{st} \cap C_{sv} \cdot C_{vt}$$

$$= C_{st}$$

A contradiction.

From Proposition 2 stating some of the properties of *PA*, the time point algebra that excludes the $\neq$ relation, we have also proved that One3 is exact for *PA*.

Proposition 2. *The following properties of PA are easily verified:*

(i) *PA is closed under composition.*

(ii) *PA is closed under intersection if we include the empty set or null relation.*

(iii) *Composition distributes over intersection, that is,* $a \cdot (b \cap c) = a \cdot b \cap a \cdot c$ *and* $(b \cap c) \cdot a = b \cdot a \cap c \cdot a$, *for all a, b, c in PA such that* $b \cap c \neq \emptyset$

Corollary 3. *The label-correcting algorithm of Figure 3 is exact for PA, provided there exists at least one consistent singleton labeling of the graph.*

The proof of the theorem and the corollary uses the property that composition distributes over intersection. By Propositions 1 and 2 this property is true for *SP* and *PA*, respectively, only if it can be guaranteed that the intersection of two labels will never result in the empty set. That is, only if all triangles are consistent ($C_{ij} \cap C_{ik} \cdot C_{kj} \neq \emptyset$, $i, j, k = 1,...,n$). Hence the proof fails for *SP* and *PA* if any triangle is inconsistent. The algorithm will correctly label the edges with the empty set if there exists an inconsistent triangle involving the source vertex, s, but since the algorithm does not look at all triangles it is easy to find counter-examples to show that the above theorem is false if there is even one inconsistent triangle in the graph.

The exactness results for the one-to-all algorithm have the appearance of being less than useful. To know whether the algorithm has computed the minimal labels we must first know whether the graph can be consistently labeled at all. But determining this is itself a difficult problem. Given that the graph is labeled with elements from *SP* or *PA*, the path consistency algorithm will correctly determine whether the graph can be consistently labeled but then the one-to-all algorithm is redundant. When and how then can we use the algorithm? There are two scenarios.

- We know *a priori* that the graph can be consistently labeled. (In section 5 we discuss an example application where this is reasonable: extracting the temporal relations between events mentioned in a narrative. The assumption is that the narrative is coherent.)

- We build the graph incrementally, applying the algorithm after the addition of each vertex. Suppose we know the relations between intervals are consistent and we add a new interval s and the relations between s and some of the previously entered intervals. The One3 algorithm is then guaranteed to detect any inconsistency since it will detect any inconsistent triangle involving the source vertex, s. This gives an online algorithm useful when the arrival of new intervals and queries about the relations between intervals are interspersed.

4. Experimental Results and a Predictive Test

In this section we present the results of some computational experiments comparing the quality of the solutions produced by Allen's and our approximation algorithms. The experiments give a partial answer to the question: with what degree of confidence can we rely on the less expensive approximate solutions? We also present a simple test for predicting when the approximation algorithms will and will not produce good quality approximations.

For each problem of size n we randomly generated a consistent singleton labeling and then added uncertainty. To generate a consistent singleton labeling we randomly generated the end points of the intervals as integers in a specified range and then translated this into the interval algebra. The uncertainty was in the form of additional disjuncts on the possible relations between two intervals. We then applied the three approximation algorithms, chose a particular edge, determined the minimal or exact label on that edge using an exact backtracking algorithm, and recorded whether the less expensive approximate solutions differed from the exact solution.

We found that how well the algorithms do is heavily dependent on the distribution from which the uncertainty is randomly generated. Figure 4 summarizes the results for two distributions. Distribution one was chosen to approximate instances that may arise in a planning application (as estimated from a block-stacking example in Allen and Koomen, 1983). The important parameter in the planning application is that the relations between most of the actions are originally unconstrained (denoted by {I}, the set of all basic relations). The values of n were also chosen to represent practical values. Fortunately, for the class of problems that may arise in the planning application, experimental results suggest that for a reassuringly large percentage of the time we can use All3, the path consistency algorithm, with near impunity: the outcome is the same as that of using an exact algorithm. With a different distribution, however, up to two-thirds of the labels on average were not minimal.

We note that the choice of how to generate random instances of the problem was largely dictated by what kinds of problems could be solved exactly in a reasonable amount of time. It would be interesting to know, for example, if it is true in general that the quality of the approximation improves as the problem size increases (as exhibited in Figure 4). There are indications that if the random instances have, before adding uncertainty, at least two distinct singleton labelings this is *not* the case but few

Distribution 1: About 75% of the time the uncertainty added is {I} and the remaining time consists of from 0 to 3 of the basic relations. *Distribution 2:* All labels are equally likely to be added as uncertainty. 150 tests performed for each problem size, *n*.

	Distribution 1			Distribution 2		
n	One3	All3	All4	One3	All3	All4
20	6.0	0.0	0.0	72.7	66.0	36.7
30	10.7	0.0	0.0	88.7	41.3	9.3
40	18.0	1.3	0.7	95.3	12.0	3.3
50	12.7	0.0	0.0	90.7	4.0	2.0
60	18.0	0.7	0.0	84.0	0.0	0.0

Figure 4. Percentage differences between the approximation algorithms and an exact algorithm for various problem sizes.

experiments were performed because exact solutions could not be computed in a reasonable amount of time.

We present a simple test for predicting when the approximation algorithms will and will not produce good quality approximations. Let *SP* be the subset of the interval algebra discussed earlier that can be solved exactly using the path consistency algorithm. Computational evidence shows a strong correlation between the percentage of the total labels that are from *SP* and how well the One3, All3, and All4 algorithms approximate the exact solution. Recall that Theorem 3 (Theorem 6) states that All3 (One3) is exact when all the labels are from *SP* so we cannot improve on that. But, as the percentage of the total labels that are from *SP* nears zero, up to three-fifths of the labels (on average) assigned by All3 and more than four-fifths of the labels assigned by One3 are not minimal (see Figure 5). Thus we have an effective test for predicting whether it would be useful to apply a more expensive algorithm.

5. Applications

In this section we survey three example applications of Allen's interval algebra chosen from the literature to show where the results of this paper could be useful.

Example 5.1.

Koubarakis et al. (1989) use Allen's interval algebra in a knowledge representation language for software development applications to allow the representation of and queries about the history of the domain and about the system's beliefs about that history. They use only the thirteen basic relations, foregoing representational

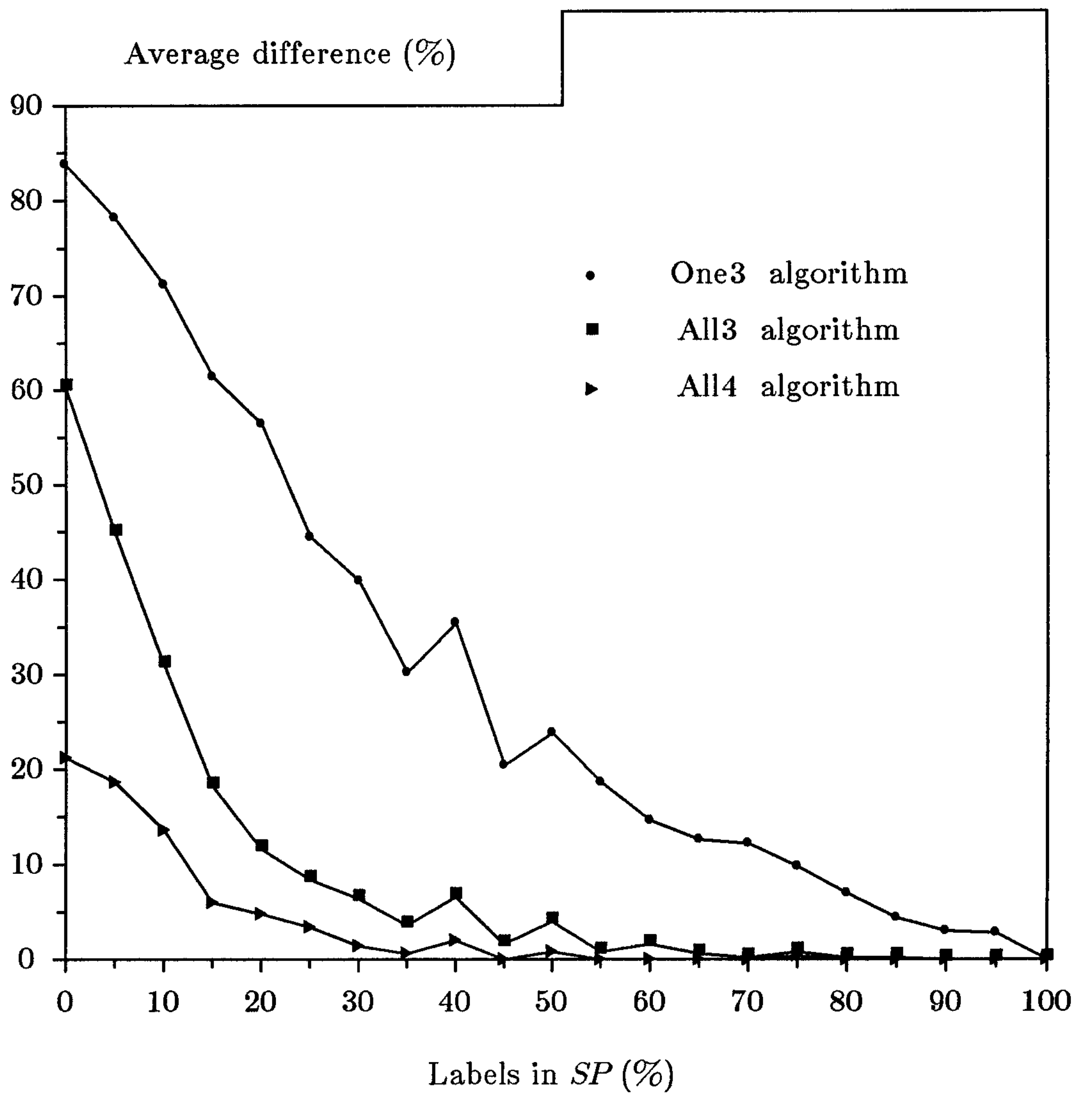


Figure 5. Percentage differences between the approximation algorithms and an exact algorithm for various percentage of labels in *SP*. 250 tests performed for each subinterval; problem size is 25.

completeness in favor of guaranteed exact answers in quick time. (Their system maintains the network of relations between the end points of every interval using the point algebra of Vilain and Kautz. Since they restrict the relationships between intervals to the 13 basic relations, the counter-example we gave in section 2.2 to the exactness of Vilain and Kautz's point algebra does not affect this application.) Our characterization of *SP*, the subset of the interval algebra for which the problem can be solved exactly and efficiently using the path consistency algorithm, shows that the expressive power of their temporal language could be expanded without compromising efficiency or exactness. As well, the one-to-all algorithm, whether we first translate into the point algebra or reason directly with *SP*, may be of significant use in a system that allows queries about the temporal relations between events in the domain. This will be especially true as the problems to be represented grow larger.

Example 5.2.

Song and Cohen (1988) use Allen's interval algebra in their solution to a problem in natural language processing: extracting and representing the temporal relations between the events mentioned in a narrative. In narrative, the relations between events are sometimes explicitly stated using adverbs or connectives but at other times are left vague. Song and Cohen restrict their representation language to the thirteen basic relations plus two defined relations—precedes (defined as {b, o, m}) and includes (defined as {eq, d, s, f})—to capture vagueness. It turns out, however, that this subset of the interval algebra is also a subset of *SP*. Thus, once we have extracted the possibly vague relationships between some of the events mentioned in the narrative, we can determine *exactly* the strongest possible assertion about the relationship between all of the events using the path consistency algorithm described in section 2.1.

Example 5.3.

The interval algebra is used in planning (Allen and Koomen, 1983; Hogge, 1987). In classical planning actions are viewed as instantaneous and thus the only allowed relations between actions are $<$, $>$, and $=$. Viewing actions as having temporal extent and using the interval algebra to represent the relations between actions allows plans to have actions that overlap. Given a plan library with temporal constraints, Hogge gives the following three steps for using his planner: (i) specify the planning problem as a set of facts, goals, and temporal constraints between them; (ii) run the planner; (iii) select among the possible temporal orderings of the operators applied in the plan. The full interval algebra is used in Hogge's planner so the exactness results do not apply here (whether useful planning can be done with the possible relations restricted to *SP* and $SP^{\neq}$ is worth further exploration).

The planner tests whether the problem specification is temporally consistent and adds an operator to the plan if, among other things, it is temporally consistent. The four consistency algorithm may be useful here as it detects inconsistencies that the path consistency algorithm does not. Doing these checks for consistency will take resources, but for certain applications it is important to produce a plan that will not fail. In cases where the temporal constraints between the goals are inconsistent, it is important to detect this early.

Suppose a disjunction of possible temporal relations between operators is determined by the planning component. A temporal ordering of the operators must be selected (step (iii) above). The aim should be to eliminate as much as possible disjunctions that cannot possibly be true as the selection process is simplified if no incorrect labels persist. Using the more expensive four consistency algorithm may be useful, as it allows greater confidence that no incorrect labels exist. Here extra work is sacrificed for extra confidence that there is not an impossible disjunct on a label. So, for applications where the tradeoff of extra work is worth the benefit of a smaller chance for incorrect plan selection, the four consistency algorithm will be useful.

We may also want to allow the user greater input in the selection process. The user could iteratively eliminate disjunctions of possible relations between operators by, at each stage, choosing a single relation from the disjunction of possible relations between two operators and propagating this choice by running the path consistency algorithm. The iterative procedure would stop when no disjunctions remain.

6. Conclusions

We considered a popular representation for temporal relationships between intervals introduced by James Allen and its associated computational or reasoning problem of, given possibly indefinite knowledge of the relations between some intervals, computing the strongest possible assertion about the relations between some or all intervals. Allen gives an approximation algorithm based on constraint propagation. We presented an algorithm for computing better approximations for the all-to-all version of the problem and a test for predicting when this more expensive algorithm is useful. We presented an algorithm for the one-to-all version of the problem and a test for predicting when this less expensive algorithm is useful. We gave a counter example to a result in the literature and identified easy (polynomial time) special cases of both versions of the problem.

Appendix A

Allen's transitivity table (reproduced from Allen, 1983). The "equals" relation is omitted, "con" is defined as {di, si, fi}, and "dur" is defined as {d, s, f}.

B r2 C / A r1 B	<	>	d	di	o	oi	m	mi	s	si	f	fi
"before" <	<	no info	< o m d s	<	<	< o m d s	<	< o m d s	<	<	< o m d s	<
"after" >	no info	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	>	>
"during" d	<	>	d	no info	< o m d s	> oi mi d f	<	>	d	> oi mi d f	d	< o m d s
"contains" di	< o m di fi	> oi di mi si	o oi dur con =	di	o di fi	oi di si	o di fi	oi di si	di fi o	di	di si oi	di
"overlaps" o	<	> oi di mi si	o d s	< o m di fi	< o m	o oi dur con =	<	oi di si	o	di fi o	d s o	< o m
"over-lapped-by" oi	< o m di fi	>	oi d f	> oi mi di si	o oi dur con =	> oi mi	o di fi	>	oi d f	oi > mi	oi	oi di si
"meets" m	<	> oi mi di si	o d s	<	<	o d s	<	f fi =	m	m	d s o	<
"met-by" mi	< o m di fi	>	oi d f	>	oi d f	>	s si =	>	d f oi	>	mi	mi
"starts" s	<	>	d	< o m di fi	< o m	oi d f	<	mi	s	s si =	d	< m o
"started by" si	< o m di fi	>	oi d f	di	o di fi	oi	o di fi	mi	s si =	si	oi	di
"finishes" f	<	>	d	> oi mi di si	o d s	> oi mi	m	>	d	> oi mi	f	f fi =
"finished-by" fi	<	> oi mi di si	o d s	di	o	oi di si	m	si oi di	o	di	f fi =	fi

Appendix B

Below we enumerate the restricted subset of the interval algebra that can be translated, using the relations $\{<, \leq, =, \geq, >, ?, \neq\}$, into disjunctions of relations between the endpoints of the intervals. We partition the elements into two sets dependent on whether $\neq$ is required in the translation. A^- and A^+ represent the start and end points of interval A, respectively, and $A^- < A^+$ and $B^- < B^+$ are true for every translation.

	A^-B^-	A^-B^+	A^+B^-	A^+B^+
{eq}	=	<	>	=
{b}	<	<	<	<
{bi}	>	>	>	>
{d}	>	<	>	<
{di}	<	<	>	>
{o}	<	<	>	<
{oi}	>	<	>	>
{m}	<	<	=	<
{mi}	>	=	>	>
{s}	=	<	>	<
{si}	=	<	>	>
{f}	>	<	>	=
{fi}	<	<	>	=
{eq, f}	≥	<	>	=
{eq, fi}	≤	<	>	=
{eq, s}	=	<	>	≤
{eq, si}	=	<	>	≥
{b, m}	<	<	≤	<
{bi, mi}	>	≥	>	>
{d, f}	>	<	>	≤
{di, fi}	<	<	>	≥
{d, s}	≥	<	>	<
{di, si}	≤	<	>	>
{o, m}	<	<	≥	<
{oi, mi}	>	≤	>	>
{o, s}	≤	<	>	<
{oi, si}	≥	<	>	>
{o, fi}	<	<	>	≤
{oi, f}	>	<	>	≥
{eq, f, fi}	?	<	>	=
{eq, s, si}	=	<	>	?
{b, o, m}	<	<	?	<
{bi, oi, mi}	>	?	>	>
{d, o, s}	?	<	>	<
{di, oi, si}	?	<	>	>
{d, oi, f}	>	<	>	?
{di, o, fi}	<	<	>	?
{o, m, fi}	<	<	≥	≤
{oi, mi, f}	>	≤	>	≥
{o, m, s}	≤	<	≥	<
{oi, mi, si}	≥	≤	>	>

	A^-B^-	A^-B^+	A^+B^-	A^+B^+
{eq, d, s, f}	≥	<	>	≤
{eq, di, si, fi}	≤	<	>	≥
{eq, o, s, fi}	≤	<	>	≤
{eq, oi, si, f}	≥	<	>	≥
{b, o, m, fi}	<	<	?	≤
{bi, oi, mi, f}	>	?	>	≥
{b, o, m, s}	≤	<	?	<
{bi, oi, mi, si}	≥	?	>	>
{d, o, m, s}	?	<	≥	<
{di, oi, mi, si}	?	≤	>	>
{d, oi, mi, f}	>	≤	>	?
{di, o, m, fi}	<	<	≥	?
{eq, o, m, s, fi}	≤	<	≥	≤
{eq, oi, mi, si, f}	≥	≤	>	≥
{b, d, o, m, s}	?	<	?	<
{bi, di, oi, mi, si}	?	?	>	>
{b, di, o, m, fi}	<	<	?	?
{bi, d, oi, mi, f}	>	?	>	?
{eq, b, o, m, s, fi}	≤	<	?	≤
{eq, bi, oi, mi, si, f}	≥	?	>	≥
{eq, d, o, s, f, fi}	?	<	>	≤
{eq, di, oi, si, f, fi}	?	<	>	≥
{eq, d, oi, s, si, f}	≥	<	>	?
{eq, di, o, s, si, fi}	≤	<	>	?
{eq, d, o, m, s, f, fi}	?	<	≥	≤
{eq, di, oi, mi, si, f, fi}	?	≤	>	≥
{eq, d, oi, mi, s, si, f}	≥	≤	>	?
{eq, di, o, m, s, si, fi}	≤	<	≥	?
{eq, b, d, o, m, s, f, fi}	?	<	?	≤
{eq, bi, di, oi, mi, si, f, fi}	?	?	>	≥
{eq, b, di, o, m, s, si, fi}	≤	<	?	?
{eq, bi, d, oi, mi, s, si, f}	≥	?	>	?
{eq, d, di, o, oi, s, si, f, fi}	?	<	>	?
{eq, d, di, o, oi, m, s, si, f, fi}	?	<	≥	?
{eq, d, di, o, oi, mi, s, si, f, fi}	?	≤	>	?
{eq, b, d, di, o, oi, m, s, si, f, fi}	?	<	?	?
{eq, bi, d, di, o, oi, mi, s, si, f, fi}	?	?	>	?
{eq, d, di, o, oi, m, mi, s, si, f, fi}	?	≤	≥	?
{eq, b, d, di, o, oi, m, mi, s, si, f, fi}	?	≤	?	?
{eq, bi, d, di, o, oi, m, mi, s, si, f, fi}	?	?	≥	?
{eq, b, bi, d, di, o, oi, m, mi, s, si, f, fi}	?	?	?	?

	A^-B^-	A^-B^+	A^+B^-	A^+B^+
{b, o}	<	<	≠	<
{bi, oi}	>	≠	>	>
{d, o}	≠	<	>	<
{d, oi}	>	<	>	≠
{di, o}	<	<	>	≠
{di, oi}	≠	<	>	>
{s, si}	=	<	>	≠
{f, fi}	≠	<	>	=
{b, d, o}	≠	<	≠	<
{b, di, o}	<	<	≠	≠
{b, o, s}	≤	<	≠	<
{b, o, fi}	<	<	≠	≤
{bi, d, oi}	>	≠	>	≠
{bi, di, oi}	≠	≠	>	>
{bi, oi, f}	>	≠	>	≥
{bi, oi, si}	≥	≠	>	>
{d, o, m}	≠	<	≥	<
{d, oi, mi}	>	≤	>	≠
{di, o, m}	<	<	≥	≠
{di, oi, mi}	≠	≤	>	>
{b, d, o, m}	≠	<	?	<
{b, d, o, s}	?	<	≠	<
{b, di, o, fi}	<	<	≠	?
{b, di, o, m}	<	<	?	≠
{bi, d, oi, f}	>	≠	>	?
{bi, d, oi, mi}	>	?	>	≠
{bi, di, oi, mi}	≠	?	>	>
{bi, di, oi, si}	?	≠	>	>
{d, di, o, oi}	≠	<	>	≠
{d, o, f, fi}	≠	<	>	≤
{d, oi, s, si}	≥	<	>	≠
{di, o, s, si}	≤	<	>	≠
{di, oi, f, fi}	≠	<	>	≥
{b, d, di, o, oi}	≠	<	≠	≠
{b, d, o, f, fi}	≠	<	≠	≤
{b, di, o, s, si}	≤	<	≠	≠
{bi, d, di, o, oi}	≠	≠	>	≠
{bi, d, oi, s, si}	≥	≠	>	≠
{bi, di, oi, f, fi}	≠	≠	>	≥
{d, di, o, oi, mi}	≠	≤	>	≠
{d, di, o, oi, m}	≠	<	≥	≠
{d, o, m, f, fi}	≠	<	≥	≤
{d, oi, mi, s, si}	≥	≤	>	≠
{di, o, m, s, si}	≤	<	≥	≠
{di, oi, mi, f, fi}	≠	≤	>	≥
{eq, b, o, s, fi}	≤	<	≠	≤
{eq, bi, oi, si, f}	≥	≠	>	≥
{b, bi, d, di, o, oi}	≠	≠	≠	≠
{b, d, di, o, oi, mi}	≠	≤	≠	≠
{b, d, di, o, oi, m}	≠	<	?	≠
{b, d, o, m, f, fi}	≠	<	?	≤
{b, di, o, m, s, si}	≤	<	?	≠
{bi, d, di, o, oi, mi}	≠	?	>	≠
{bi, d, di, o, oi, m}	≠	≠	≥	≠
{bi, d, oi, mi, s, si}	≥	?	>	≠
{bi, di, oi, mi, f, fi}	≠	?	>	≥
{d, di, o, oi, f, fi}	≠	<	>	?
{d, di, o, oi, m, mi}	≠	≤	≥	≠
{d, di, o, oi, s, si}	?	<	>	≠
{b, bi, d, di, o, oi, mi}	≠	?	≠	≠
{b, bi, d, di, o, oi, m}	≠	≠	?	≠
{b, d, di, o, oi, f, fi}	≠	<	≠	?
{b, d, di, o, oi, m, mi}	≠	≤	?	≠
{b, d, di, o, oi, s, si}	?	<	≠	≠
{bi, d, di, o, oi, f, fi}	≠	≠	>	?
{bi, d, di, o, oi, m, mi}	≠	?	≥	≠
{bi, d, di, o, oi, s, si}	?	≠	>	≠
{d, di, o, oi, m, f, fi}	≠	<	≥	?
{d, di, o, oi, m, s, si}	?	<	≥	≠
{d, di, o, oi, mi, f, fi}	≠	≤	>	?
{d, di, o, oi, mi, s, si}	?	≤	>	≠
{eq, b, d, o, s, f, fi}	?	<	≠	≤
{eq, b, di, o, s, si, fi}	≤	<	≠	?
{eq, bi, d, oi, s, si, f}	≥	≠	>	?
{eq, bi, di, oi, si, f, fi}	?	≠	>	≥
{b, bi, d, di, o, oi, f, fi}	≠	≠	≠	?
{b, bi, d, di, o, oi, m, mi}	≠	?	?	≠
{b, bi, d, di, o, oi, s, si}	?	≠	≠	≠
{b, d, di, o, oi, m, f, fi}	≠	<	?	?
{b, d, di, o, oi, m, s, si}	?	<	?	≠
{b, d, di, o, oi, mi, f, fi}	≠	≤	≠	?
{b, d, di, o, oi, mi, s, si}	?	≤	≠	≠
{bi, d, di, o, oi, m, f, fi}	≠	≠	≥	?
{bi, d, di, o, oi, m, s, si}	?	≠	≥	≠
{bi, d, di, o, oi, mi, f, fi}	≠	?	>	?
{bi, d, di, o, oi, mi, s, si}	?	?	>	≠
{d, di, o, oi, m, mi, f, fi}	≠	≤	≥	?
{d, di, o, oi, m, mi, s, si}	?	≤	≥	≠
{b, bi, d, di, o, oi, m, f, fi}	≠	≠	?	?
{b, bi, d, di, o, oi, m, s, si}	?	≠	?	≠
{b, bi, d, di, o, oi, mi, f, fi}	≠	?	≠	?
{b, bi, d, di, o, oi, mi, s, si}	?	?	≠	≠
{b, d, di, o, oi, m, mi, f, fi}	≠	≤	?	?
{b, d, di, o, oi, m, mi, s, si}	?	≤	?	≠
{bi, d, di, o, oi, m, mi, f, fi}	≠	?	≥	?
{bi, d, di, o, oi, m, mi, s, si}	?	?	≥	≠
{b, bi, d, di, o, oi, m, mi, f, fi}	≠	?	?	?
{b, bi, d, di, o, oi, m, mi, s, si}	?	?	?	≠
{eq, b, d, di, o, oi, s, si, f, fi}	?	<	≠	?
{eq, bi, d, di, o, oi, s, si, f, fi}	?	≠	>	?
{eq, b, bi, d, di, o, oi, s, si, f, fi}	?	≠	≠	?
{eq, b, d, di, o, oi, mi, s, si, f, fi}	?	≤	≠	?
{eq, bi, d, di, o, oi, m, s, si, f, fi}	?	≠	≥	?
{eq, b, bi, d, di, o, oi, m, s, si, f, fi}	?	≠	?	?
{eq, b, bi, d, di, o, oi, mi, s, si, f, fi}	?	?	≠	?

Appendix C

In this appendix we prove that the minimal labeling problem is NP-Complete. We do this by first proving that the following problem is NP-Complete.

Interval Algebra Consistency (IAC)

Given a directed graph $G = (V, E)$ with labels on the edges from the set of elements of the interval algebra, is there a consistent singleton labeling of the graph?

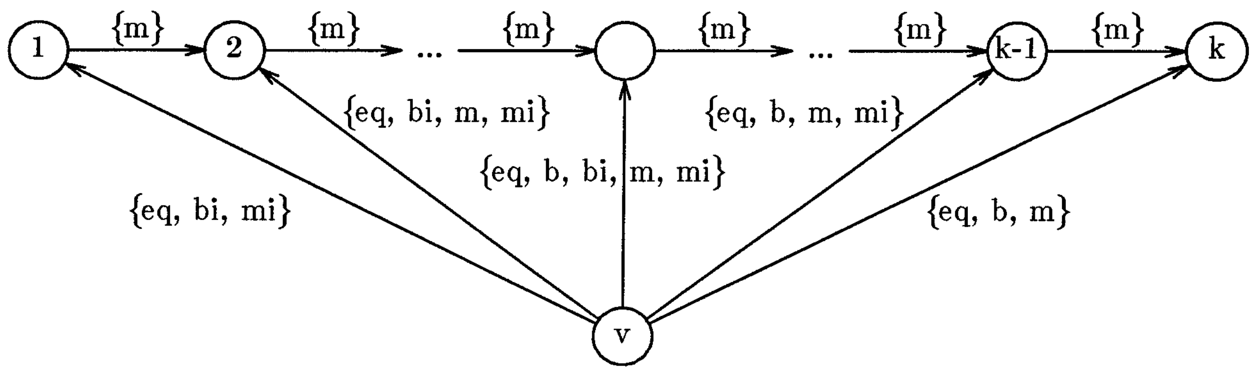
Vilain and Kautz (1986) give a proof sketch that IAC is NP-Hard but it is difficult to reconstruct the full proof. We will also need the following definition.

Graph Coloring

Given a graph $G = (V, E)$ and an integer k, is there a mapping $\chi : V \longrightarrow \{1, 2, ..., k\}$ such that (v, u) in E implies $\chi(v) \neq \chi(u)$?

Theorem A3. *Interval Algebra Consistency is NP-Complete.*

Proof. The Interval Algebra Consistency problem is clearly in NP since, for every yes instance of the problem, there exists a concise certificate, a consistent singleton labeling, that can be easily checked for validity. Furthermore, we can polynomial transform Graph Coloring, a known NP-Complete problem (ref. Aho et al., 1974), to it. Given an undirected graph $G = (V, E)$ and an integer k we show how to construct a labeled, directed graph $G_L = (V_L, E_L)$ such that G_L has a consistent singleton labeling if and only if there is a coloring of G using k colors. We construct k vertices in V_L that meet each other in sequence. (Note that any sensible instance of Graph Coloring will have $k \leq |V|$ so the transformation is still polynomial). For each v in V we create a vertex in V_L with the associated arcs and labels as follows.



The idea is that in any consistent singleton labeling, the vertex, v, is forced to be equal to only one of the k special vertices. We also must ensure that any two adjacent vertices do not map to the same value. Hence, for each (v, w) in E we create an arc and label as follows.

$$\begin{tikzcd} v \arrow[r, "{\{b, bi, m, mi\}}"] & w \end{tikzcd}$$

Thus, an efficient algorithm for IAC would imply an efficient algorithm for Graph Coloring. Hence, the IAC problem is NP-Complete. Vilain and Kautz show that if there is an efficient algorithm for the minimal labeling problem then there is an efficient algorithm for IAC. Hence, the minimal labeling problem is also NP-Complete.

References

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass.

Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. *Comm. ACM* **26**, 832-843.

Allen, J. F. 1984. Towards a General Theory of Action and Time. *Artificial Intelligence* **23**, 123-154.

Allen, J. F., and J. A. Koomen. 1983. Planning Using a Temporal World Model. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI),* Karlsruhe, W. Germany, 741-747.

Chvâtal, V. 1983. *Linear Programming.* W. H. Freeman and Company, New York.

Dijkstra, E. W. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik* **1**, 269-271.

Edmonds, J., and R. M. Karp. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* **19**, 248-264.

Freuder, E. C. 1978. Synthesizing Constraint Expressions. *Comm. ACM* **21**, 958-966.

Freuder, E. C. 1982. A Sufficient Condition for Backtrack-Free Search. *J. ACM* **29**, 24-32.

Hogge, J. C. 1987. TPLAN: A Temporal Interval-Based Planner with Novel Extensions. University of Illinois Department of Computer Science Technical Report UIUCDCS-R-87, Sept.

Johnson, D. B. 1973. A Note on Dijkstra's Shortest Path Algorithm. *J. ACM* **20**, 385-388.

Koubarakis, M., J. Mylopoulos, M. Stanley, and A. Borgida. 1989. Telos: Features and Formalization. Technical Report, Computer Systems Research Institute, University of Toronto, Feb.

Ladkin, P. B. 1988. Satisfying First-Order Constraints About Time Intervals. *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI),* Saint Paul, Minn., 512-517.

Ladkin, P. B., and R. Maddux. 1988. On Binary Constraint Networks. Technical Report, Kestrel Institute, Palo Alto, Calif., Oct.

Mackworth, A. K. 1977. Consistency in Networks of Relations. *Artificial Intelligence* **8**, 99-118.

Mackworth, A. K., and E. C. Freuder. 1985. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence* **25**, 65-74.

Montanari, U. 1974. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Inform. Sci.* **7**, 95-132.

Song, F., and R. Cohen. 1988. The Interpretation of Temporal Relations in Narrative. *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI),* Saint Paul, Minn., 745-750.

Tsang, E. P. K. 1987. The Consistent Labeling Problem in Temporal Reasoning. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI),* Seattle, Wash., 251-255.

Valdés-Pérez, R. E. 1986. Spatio-Temporal Reasoning and Linear Inequalities. Memo 875, MIT Artificial Intelligence Laboratory, May.

Valdés-Pérez, R. E. 1987. The Satisfiability of Temporal Constraint Networks. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI),* Seattle, Wash., 256-260.

Vilain, M., and H. Kautz. 1986. Constraint Propagation Algorithms for Temporal Reasoning. *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI),* Philadelphia, Pa., 377-382.