ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/220541901

Exact and approximate reasoning about temporal relations

Article *in* Computational Intelligence · August 1990

Impact Factor: 0.67 · DOI: 10.1111/j.1467-8640.1990.tb00130.x · Source: DBLP

CITATIONS
179

READS
17

2 authors, including:

Peter van Beek
University of Waterloo

88 PUBLICATIONS **3,066** CITATIONS

SEE PROFILE

All in-text references underlined in blue are linked to publications on ResearchGate, letting you access and read them immediately.

Available from: Peter van Beek
Retrieved on: 09 May 2016

Exact and Approximate Reasoning about Temporal Relations

Peter van Beek
Department of Computing Science
University of Alberta
Edmonton, Alberta
CANADA T6G 2H1

Robin Cohen
Department of Computer Science
University of Waterloo
Waterloo, Ontario
CANADA N2L 3G1

Abstract

Allen gives an algebra for representing qualitative temporal information about the relationships between pairs of intervals. In this paper, we address a fundamental reasoning task that arises in applications of the algebra: Given (possibly indefinite) knowledge about the relationships between intervals, find all feasible relationships between two intervals. We call this the minimal labels problem. Finding the minimal labels can be viewed as computing the deductive consequences of our knowledge. 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, determine analytically under what conditions the algorithms are exact, and examine, through some computational experiments, the quality of the approximate solutions produced by the algorithms. We also give a simple test for predicting when the approximation algorithms will and will not produce good quality approximations. Finally, we survey three example applications of the interval algebra chosen from the literature to show where the results of this paper could be useful.

Keywords: temporal reasoning, interval algebra, point algebra, constraint satisfaction.

Appears in: *Computational Intelligence* **6**, 132-144, 1990.

A preliminary version of this paper appeared in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* Morgan Kaufmann, San Mateo, Calif., 1989, 1291-1296.

This work was done while the first author was in the Department of Computer Science at the University of Waterloo.

1. Introduction

Much temporal information is information such as "The FLQ crisis took place during Trudeau's prime ministership". Here no quantitative information such as date or duration information is specified, only the qualitative information that the interval of time associated with one event occurred during the interval of time of another event. Allen (1983) gives an algebra for representing such temporal information about the relations between pairs of intervals. Application areas of the algebra include natural language processing (Allen, 1984; Song and Cohen, 1988), planning (Allen and Koomen, 1983; Hogge, 1987), and knowledge representation (Koubarakis et al., 1989).

In this paper, we address a reasoning task that arises in applications of this algebra: Given (possibly indefinite) knowledge about the relationships between intervals, find all feasible relationships between two intervals. We call this the minimal labels problem. Finding the minimal labels can be viewed as computing the deductive consequences of our knowledge. As an example, consider the following description of events: "He informed his friend of the decision during the afternoon but did not inform his family until after the evening meal." From the given temporal information and the knowledge that the afternoon is before the evening, we can make the simple inference that the "inform friend" event occurred before the "inform family" event.

Vilain and Kautz (1986, 1989) show that the minimal labels problem is NP-Complete for the interval algebra, where the problem size is the number of intervals. 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.
- **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.
- **Easy special cases**: Identify interesting special cases of the NP-Complete problem that are solvable in polynomial time. This alternative often takes the form of limiting the expressive power of the representation language.

In this paper, we investigate the latter two alternatives: efficient algorithms for computing approximations to the minimal labels problem and some special cases where the approximation algorithms are exact. We consider two versions of the problem: an all-to-all version where we determine 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.

Allen (1983) gives an $O(n^3)$ approximation algorithm for the all-to-all minimal labels problem. We explore better (and, unfortunately, more expensive) approximation algorithms. In particular, we develop an $O(n^4)$ algorithm that computes a better approximation to the minimal labels. We also explore how far we must restrict the expressive power of the representation language to guarantee that (i) Allen's algorithm is exact, and (ii) our approximation algorithm is exact. Vilain and Kautz (1986) define a point algebra and show that a class of minimal labels problems in the interval algebra can be phrased as minimal labels problems in their point algebra. Vilain and Kautz (1986) then claim that

Allen's algorithm is exact for computing the minimal labels between points. However, we present a counter-example to their theorem. We characterize the subset of the point algebra and the interval algebra for which Allen's algorithm is exact. We also show that our $O(n^4)$ approximation algorithm is exact for Vilain and Kautz's point algebra and the subset of the interval algebra that can be translated into the point algebra.

In some applications we are only interested in the minimal labels between a few of the intervals. If we nevertheless compute the minimal labels between all intervals, we may be doing too much work. We give an algorithm for a one-to-all version of the problem, where we only determine approximations to the minimal labels between a single source interval and every other interval. The algorithm produces good quality approximations for certain classes of problems and is shown to take $O(n^2)$ time. As with the all-to-all algorithms, 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.

To test the quality of the approximations produced by Allen's and our algorithms, we performed some computational experiments. We randomly generated instances of the problem and determined how often an approximation to the minimal label differed from the true minimal label. We found that how well the approximation algorithms do is heavily dependent on the distribution from which the relations between intervals are randomly generated. We present a simple test for predicting when the approximation algorithms will and will not produce good quality approximations.

An outline of the rest of the paper is as follows. Section 2 contains the definitions and terminology used throughout the paper. Section 3 examines the all-to-all minimal labels problem, first giving approximation algorithms, then identifying the easy (polynomial time) special cases where the algorithms are exact. Section 4 examines the one-to-all minimal labels problem, first giving an approximation algorithm, then identifying the easy (polynomial time) special cases where the algorithm is exact. Section 5 contains the results of some computational experiments and a test for predicting when the approximation algorithms will and will not produce good quality approximations. Section 6 surveys selected applications of the interval algebra with the intent of showing where the results of this paper will be of use.

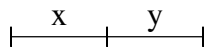
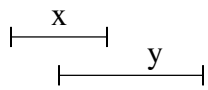
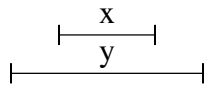
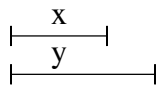
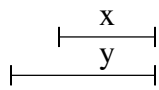
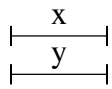
2. Definitions and Terminology

In this section we review Allen's interval algebra and Vilain and Kautz's point algebra. We then formalize our reasoning task using networks of binary relations (Montanari, 1974) and give an example. We end with the definitions of two new algebras that will be prominent in the rest of the paper.

2.1. Specification of the Algebras

A set, together with one or more operations on the set, is called an algebra. The set must be closed under the operations.

The interval algebra, **IA** (Allen, 1983) is defined as follows. There are thirteen **basic** relations (including inverses) that can hold between two intervals.

Relation	Symbol	Inverse	Meaning
x before y	b	bi	
x meets y	m	mi	
x overlaps y	o	oi	
x during y	d	di	
x starts y	s	si	
x finishes y	f	fi	
x equal y	eq	eq	

We want to be able to represent indefinite information so we allow the relation between two intervals to be a disjunction of the basic relations. We use sets to list the disjunctions. Let I be the set of all basic relations, {eq, b, bi, m, mi, o, oi, d, di, s, si, f, fi}. **IA** is the algebra with underlying set 2^I, the power set or set of all subsets of I, unary operator inverse, and binary operators intersection and composition (denoted "constraints" in Allen, 1983).

The point algebra, **PA** (Vilain and Kautz, 1986) is defined as follows. There are three **basic** relations that can hold between two points <, =, and >. As in the interval algebra, we want to be able to represent indefinite information so we allow the relation between two points to be a disjunction of the basic relations. **PA** is the algebra with underlying set $\{\emptyset, <, \leq, =, >, \geq, \neq, ?\}$, unary operator inverse, and binary operators intersection and composition (denoted addition and multiplication in Vilain and Kautz, 1986; there the operators are defined over bit vector representations of the underlying set). Note that $\leq$, for example, is an abbreviation of $\{<, =\}$ and ? means there is no constraint between two points, $\{<, =, >\}$.

Ladkin and Maddux (1988a, 1988b) consider algebras that are additionally closed under the operations complement and union and show that the algebras are relation algebras.

2.2. Formalization of the Reasoning Task

We formalize the minimal labels problem using networks of binary relations (Montanari, 1974). Our development follows that found in Dechter et al. (1989) and Ladkin and Maddux (1988a, 1988b). This approach allows us to use previously known algorithms and eases their proofs of correctness.

A **network of binary relations** (Montanari, 1974) is defined as a set X of n variables $\{X_1, X_2, \ldots, X_n\}$, a domain D_i of possible values for each variable, and binary relations between variables that preclude certain combinations of instantiations of the

variables. An **instantiation** of the variables in X is an n-tuple $(x_1, x_2, \ldots, x_n)$, representing an assignment of $x_i \in D_i$ to X_i. A **consistent instantiation** of a network is an instantiation of the variables such that the relations between variables are satisfied. A network is said to be **inconsistent** if no consistent instantiation exists.

The relation between variables X_i and X_j is denoted R_{ij}. We require that $x_j R_{ji} x_i \equiv x_i R_{ij} x_j$. For the networks of interest here, the R_{ij} will be disjunctions of the basic relations from one of the algebras. That is,

$$x_i R_{ij} x_j \equiv x_i R_1 x_j \text{ or } \cdots \text{ or } x_i R_m x_j$$

The basic relations of the algebras are disjoint. Hence, if an instantiation of variables X_i and X_j satisfies R_{ij}, then one and only one of the disjuncts R_k in R_{ij} is satisfied.

An **IA network** is a network of binary relations where the variables represent time intervals, the domains of the variables are the set of ordered pairs of real numbers $\{< s, e > \mid s < e\}$, with s and e representing the start and end points of the interval, respectively, and the binary relations between variables are disjunctions of the basic interval relations.

A **PA network** is a network of binary relations where the the variables represent time points, the domains of the variables are the set of real numbers, and the binary relations between variables are disjunctions of the basic point relations.

An **IA** network or a **PA** network can be represented by a labeled graph where the vertices $V = (1, \ldots, n)$ represent the variables $X_1, \ldots, X_n$, and each edge $(i, j) \in V \times V$ is assigned an element from the appropriate algebra. We describe the element of the algebra assigned to an edge as its **label**. The label on an edge (i, j) specifies the binary relation between variable X_i and X_j. That is, the label on edge (i, j)

$$\{R_1, \ldots, R_m\}$$

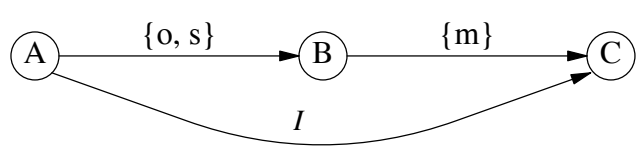
specifies the relation

$$R_{ij}\colon\ X_i\, R_1\, X_j \text{ or } \cdots \text{ or } X_i\, R_m\, X_j$$

A labeled graph will be described by an $n \times n$ adjacency matrix C where entry C_{ij} is the label on edge (i, j).

An element $R_k \in C_{ij}$ is **feasible** with respect to a network if and only if there exists a consistent instantiation of the network where R_k is satisfied. Given an **IA** network or a **PA** network, the **minimal label** between two variables X_i and X_j in the network is the set consisting of *all and only* the $R_k \in C_{ij}$ that are feasible. The reasoning task is to determine the minimal labels of the network. Call this the minimal labels problem. An algorithm is **exact** for a class of input if it, depending on the version of the problem, correctly finds the minimal labels between all pairs of variables or between one variable and every other variable, for all instances in that class.

Here is an example of an **IA** network. Suppose we know that interval A either overlaps or starts interval B, but we are not sure which, and that interval B meets interval C. We represent this as follows



where the label I, the set of all basic relations, shows we have no direct knowledge of the relation between A and C. As a graphical convention, we never show the edges (i, i), and if we show the edge (i, j), we do not show the edge (j, i). For an example of our reasoning task, we will find the minimal labels between every pair of intervals. The relations shown between A and B and between B and C are already the minimal labels. The minimal label between A and C is {b}, the "before" relation. No other basic relation between A and C is feasible. Thus, in every consistent instantiation of this network A is before C and B meets C. Further, there exists a consistent instantiation where A overlaps B and one where A starts B. We show these two possible arrangements of the intervals in the diagram below.

2.3. Additional Terminology

Mackworth (1977) defines three properties of networks that characterize local consistency of networks. A network is **node consistent** if, for every node i,

$$\forall x\ (x \in D_i) \ \rightarrow\ x\, R_{ii}\, x$$

IA and **PA** networks are always node consistent as R_{ii} is always the equality relation (eq for **IA**, = for **PA**). A network is **arc consistent** if, for every arc (i, j),

$$\forall x\ (x \in D_i) \ \rightarrow\ \exists y\ (y \in D_j) \text{ and } x\, R_{ij}\, y$$

It is clear that **IA** and **PA** networks are also always arc consistent. A network is **path consistent** if, for every triple (i, k, j) of vertices,

$$\forall x \forall z\ x\, R_{ij}\, z \ \rightarrow\ \exists y\ (y \in D_k) \text{ and } x\, R_{ik}\, y \text{ and } y\, R_{kj}\, z$$

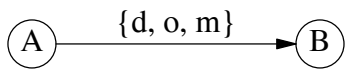
In words, for every instantiation of X_i and X_j that satisfies the direct relation, R_{ij}, there exists an instantiation of X_k such that R_{ik} and R_{kj} are also satisfied. **IA** and **PA** networks are not always path consistent as the example in the previous section shows.

Freuder (1978) generalizes this to k-consistency. A network is **k-consistent** if and only if given any instantiation of any $k-1$ variables satisfying all the direct relations among those variables, there exists an instantiation of any kth variable such that the k values taken together satisfy all the relations among the k variables. Node, arc, and path consistency correspond to one, two, and three consistency, respectively. Section 3 contains examples of 3-consistent but not 4-consistent **IA** networks. Freuder (1982) defines **strong k-consistency** as j-consistent for all $j \leq k$.

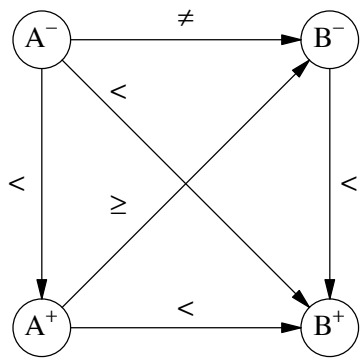
The importance of these definitions is the following. Strong k-consistency implies that, for every choice of k of the n variables, every pair of values that satisfies a direct relation appears in some consistent instantiation of the k variables. Hence, if the network is strongly k-consistent, every basic relation in the relations between variables is feasible with respect to every possible subnetwork of k variables. In particular, if the network is strongly n-consistent, the relations between variables are the minimal labels.

2.4. New Algebras from Old

Vilain and Kautz (1986) show that a class of minimal labels problems in **IA** networks can be phrased as minimal labels problems in **PA** networks. Let **SA** be the algebra with underlying set restricted to be the subset of **IA** that can be translated into relations between the endpoints of the intervals using only the relations in **PA** (see the appendix for an enumeration of **SA** and the translation into **PA**; also in Granier, 1988 and Ladkin and Maddux, 1988a, where the relations are called the "pointisable" relations). The **IA** networks that can be so translated are those whose labels are elements of **SA**. As an example translation, the **IA** network



translates into the following **PA** network (where A^- and A^+ represent the start and end points of interval A, respectively)



We define a new point algebra and new corresponding subset of the interval algebra that will be of importance throughout the rest of the paper. $\mathbf{PA_c}$ is the algebra with the same operators and underlying set as **PA** with the exception that ≠ is excluded from the underlying set. The subscript 'c' is to indicate that the sets of tuples defining the relations in $\mathbf{PA_c}$ are convex. Let $\mathbf{SA_c}$ be the algebra with the same operators as **IA** and with underlying set restricted to be the subset of **IA** that can be translated into relations between the endpoints of the intervals using only the relations in $\mathbf{PA_c}$ (see the appendix for an enumeration of $\mathbf{SA_c}$ and the translation into $\mathbf{PA_c}$ and Nökel, 1988 for a graphical representation of $\mathbf{SA_c}$).

Proposition 1. *The following properties of* $\mathbf{SA_c}$ ($\mathbf{PA_c}$) *are easily verified:*

(i) *The underlying set of* $\mathbf{SA_c}$ ($\mathbf{PA_c}$) *is closed under the operations inverse, intersection, and composition.*

(ii) *Composition distributes over intersection, provided the intersection does not give the empty set. 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 the underlying set of* $\mathbf{SA_c}$ ($\mathbf{PA_c}$), *if* $b \cap c \neq \varnothing$

Clause (i) of Proposition 1 verifies that $\mathbf{SA_c}$ and $\mathbf{PA_c}$ are indeed algebras. Clause (ii) will prove useful later in the paper.

3. The All-to-All Problem

3.1. Approximation Algorithms

Allen (1983) gives an $O(n^3)$ approximation algorithm for the all-to-all minimal labels problem that is a special case of path consistency algorithms (Montanari, 1974; Mackworth, 1977). In this section we describe Mackworth's (1977) path consistency algorithm with some small simplifications because of the properties of **IA**. We then develop a better but more expensive algorithm for determining approximations to the minimal labels between all intervals. The labels computed by the algorithm, as with Allen's algorithm, will always be a superset (not necessarily proper) of the minimal labels. The algorithm computes a better approximation in that there are fewer infeasible relations in the computed labels.

3.1.1. The Path Consistency Algorithm

The path consistency algorithm works as follows (see Fig. 1; a more detailed description can be found in Allen, 1983 and Mackworth, 1977). The input is the adjacency matrix description of a network where entry 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 does not change the result (Mackworth, 1977, p. 113). It does, however, influence the amount of work done. In practice, sorting the triples at the start according to their labels and adding new triples to the front of the queue works well. We defer until Section 4 a discussion about defining an order over **IA**.

Theorem 1 (Montanari, 1974; Mackworth and Freuder, 1985). *The algorithm in Fig. 1 achieves path consistency and requires $O(n^3)$ time, where n is the number of intervals or points.*

To use the path consistency algorithm we need the operations of inverse, intersection, and composition of relations (equation 3.2 of Fig. 1). These operations have their usual first-order definitions: inverse: $xR^{-1}y \equiv yRx$, intersection: $x(R \cap S)y \equiv xRy$ and xSy, and composition: $xR \cdot Sz \equiv \exists y\ xRy$ and ySz. Below we discuss how to make this algorithmic and some implementation considerations. Recall that the operands of the algebraic operations are sets of basic relations. The intersection of two labels is simply set intersection. For **PA** and $\mathbf{PA_c}$ the tables for the inverse of a label and the composition of two labels are easily specified as the underlying sets are small (see Vilain and Kautz, 1986). Similarly for **SA** and $\mathbf{SA_c}$. For **IA**, however, the tables for inverse and composition are too large to be practical. Hogge's (1987) approach for inverse is to use a clever bit swapping technique and for composition is to use four tables with an indexing scheme. Allen's approaches take less space but more time. For the inverse of a label, if $C_{ij} = \{R_1, \ldots, R_m\}$, then $C_{ij}^{-1} = \{R_1^{-1}, \ldots, R_m^{-1}\}$. This holds because inverse distributes over union. For the

Input: A matrix C where entry C_{ij} is the label on edge (i, j).

Output: A path consistency approximation to the minimal labels for C_{ij}, $i, j = 1, \ldots, n$.

procedure PC
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 (3.1)
    $t \leftarrow C_{ij} \cap C_{ik} \cdot C_{kj}$ (3.2)
    if $(t \neq C_{ij})$ **then begin**
      $C_{ij} \leftarrow t$
      $C_{ji} \leftarrow$ INVERSE (t)
      $Q \leftarrow Q \cup$ RELATED PATHS (i, j)
    end
  end
end

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

Fig. 1. Path Consistency Algorithm (Mackworth, 1977)

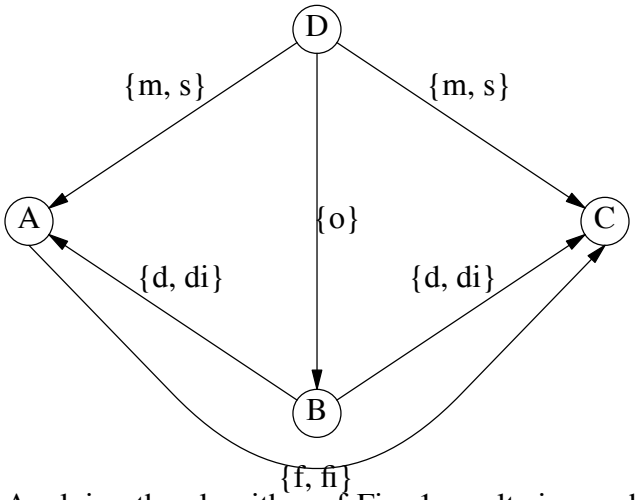
composition of two labels, take the union of the pairwise composition of the basic relations,

$$C_{ik} \cdot C_{kj} \equiv \{R \cdot S \mid R \in C_{ik}, S \in C_{kj}\} \tag{3.3}$$

(see Allen, 1983 for the composition table for the basic relations; there equation 3.3 is denoted constraints and the composition table for the basic relations is denoted the transitivity table). Equation 3.3 holds because composition of relations distributes over union of relations.

3.1.2. Improving the Approximation

Where does the path consistency algorithm fail? Determining this will help us develop better approximation algorithms. Recall that **IA** networks are guaranteed to be node and arc consistent. Thus, if an **IA** network is also made path consistent, then the network is strongly three consistent. Strongly three consistent means that the network has the property that every $R_k \in C_{ij}$ is feasible with respect to every possible subgraph of three vertices. For **IA** networks this is insufficient for even deciding whether the network is consistent. We give an example from Allen (1983) ascribed to Henry Kautz.



Applying the algorithm of Fig. 1 results in no changes to the labels; the network is path consistent. However, the network is inconsistent. The minimal labels are all $\{\varnothing\}$.

As mentioned, the path consistency algorithm, as an approximation, ensures that the $R_k \in C_{ij}$ are locally feasible—feasible not necessarily with respect to the entire graph, but with respect to every possible subgraph of three vertices. Another way to view the combined effect of equation (3.2) and equation (3.3) for the composition of relations is that C_{ij} gets updated to be the set of the $R_k \in C_{ij}$ that are feasible with respect to the triangle (i, k, j). That is, we ensure that every $R_k \in C_{ij}$ is feasible with respect to every possible subgraph of three vertices (or 3-cliques in the graph) and we only ask for the composition of labels on edges that share a vertex.

The simple idea for improving the approximation is then to ensure that every $R_k \in C_{ij}$ is feasible with respect to every possible subgraph of four vertices (or 4-cliques) and we only ask for the composition of labels on triangles that share an edge. Equation (3.2a) and the definition of composition over triangles (equation 3.3a) ensure that C_{ij} gets updated to be the set of the $R_k \in C_{ij}$ that are feasible with respect to the subgraph of four vertices (i, k, l, j):

$$\Delta_{ikl} \cdot \Delta_{klj} \equiv \{ (P \cdot S) \cap (Q \cdot T) \mid P \in C_{ik}, R \in C_{kl}, T \in C_{lj}, \quad (3.3a)$$
$$Q \in (P \cdot R) \cap C_{il}, S \in (R \cdot T) \cap C_{kj} \}$$

The algorithm iterates until this property holds for all possible subgraphs of four vertices. 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 Fig. 1 are summarized in Fig. 2.

Theorem 2. *Procedure* AAC, *the path consistency algorithm with the changes of Fig. 2, ensures the labels are minimal with respect to all subgraphs of four vertices and requires $O(n^4)$ time, where n is number of intervals or points.*

Input: A matrix C where entry C_{ij} is the label on edge (i, j).

Output: An approximation to the minimal labels for C_{ij}, $i, j = 1, \ldots, n$.

procedure AAC

$$\text{select and delete a } 4-\text{tuple } (i, k, l, j) \text{ from } Q \tag{3.1a}$$

$$t \leftarrow C_{ij} \cap \Delta_{ikl} \cdot \Delta_{klj} \tag{3.2a}$$

procedure RELATED PATHS (i, j)

return $\{ (k, i, j, l) \mid 1 \le k < l \le n,\ k, l \ne i, j \} \cup$
$\{ (i, j, l, k), (k, l, i, j) \mid 1 \le k, l \le n,\ k \ne l,\ k, l \ne i, j \}$

Fig. 2. New All-to-All Consistency Algorithm. Shown are changes to the path consistency algorithm.

Proof. A label is minimal with respect to all subgraphs of four vertices if, for every 4-tuple (i, k, l, j) of vertices,

$$\forall x_i \forall x_j\ x_i R_{ij} x_j \ \rightarrow\ \exists x_k \exists x_l\ x_u R_{uv} x_v, \qquad u, v = i, j, k, l$$

Let a singleton labeling be a labeling of a graph such that the labels on edges are singleton sets (sets containing a single basic relation). The effect of equations (3.2a) and (3.3a) is to test, for each $R_k \in C_{ij}$, whether there exists a singleton labeling of the subgraph of four vertices that is path consistent. By Theorem 3 and the fact that the basic relations are in $\mathbf{SA_c}$, this is sufficient to test whether a consistent instantiation exists.

The proof of the time bound is similar to that for the path consistency algorithm (Mackworth and Freuder, 1985) and is omitted. □

The algorithm also achieves strong four consistency for **SA** and **PA** networks (see the inductive proof of Theorem 4 and Corollary 2). But it is easy to find examples of **IA** network of size four that are minimal but not four consistent. This corrects a claim in (van Beek, 1989) where the algorithm is also said to achieve four consistency for **IA** networks. Freuder (1978) gives an algorithm for achieving k-consistency for any k for general networks of relations.

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.

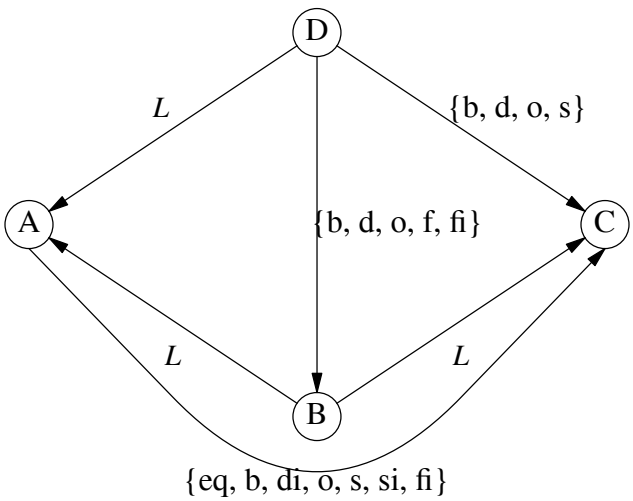
3.2. Easy (Polynomial Time) Special Cases

In this section we explore how far we must restrict the expressive power of the representation language to guarantee that (i) the path consistency algorithm is exact, and (ii) the new all-to-all consistency algorithm is exact.

3.2.1. Easy Special Cases of the Path Consistency Algorithm

Montanari (1974) shows that the path consistency algorithm is exact for a restricted class of binary relations. However, the relations of interest here do not all fall into this class. Valdés-Pérez (1986, 1987) shows that the path consistency algorithm correctly finds the minimal labels when applied to networks that are labeled only with the basic relations of **IA**.

Vilain and Kautz (1986) claim something stronger. Vilain and Kautz assert (Theorem 4, p. 380) that the path consistency algorithm correctly finds the minimal labels for **PA** networks. If their claim is true we can also find the minimal labels for **SA** networks by first translating into a **PA** network, finding the minimal labels, and then translating back. However, their claim is false. Here we present a counter-example demonstrating that the path consistency algorithm is not exact for **PA** networks. The counter-example also shows that path consistency is not exact for **SA** networks if, instead of first translating into a **PA** network, we use the interval algebra representation directly. Below is the graphical representation of the example **SA** network.



where

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

The adjacency matrix representation of the translation into a **PA** network is the following.

	A^-	A^+	B^-	B^+	C^-	C^+	D^-	D^+
A^-	=	<	≠	≤	≤	<	≠	≤
A^+	>	=	>	?	≠	?	>	?
B^-	≠	<	=	<	≠	<	≠	≠
B^+	≥	?	>	=	≥	?	>	≥
C^-	≥	≠	≠	≤	=	<	?	≠
C^+	>	?	>	?	>	=	>	>
D^-	≠	<	≠	<	?	<	=	<
D^+	≥	?	≠	≤	≠	<	>	=

Applying the algorithm of Fig. 1 results in no changes; the network is already path consistent. However, the relation ≤ between A^- and B^+ is not the minimal label, thus demonstrating that the path consistency algorithm is not exact for **PA** networks. The minimal label is <. 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 removed because it is not feasible. 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 does determine the minimal labels. To reiterate, the counter-example shows that the path consistency algorithm does not correctly compute the minimal labels for **SA** and **PA** networks. Ladkin and Maddux (1988a, 1988b) show that path consistency is sufficient for deciding whether a **PA** network is inconsistent. They give an $O(n^2)$ algorithm for finding a consistent instantiation of a path consistent **PA** network.

We next prove that the path consistency algorithm is exact for $\mathbf{SA_c}$ networks. A corollary shows that path consistency is exact for $\mathbf{PA_c}$ networks as well, where $\mathbf{PA_c}$ is the point algebra that excludes the ≠ relation. The following theorem on the intersection of convex sets will be useful in the proof of exactness.

Helly's theorem. (ref. Chvátal, 1983). *Let F be a finite family of at least n+1 convex sets in R^n. If 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 correctly finds the minimal labels between all pairs of intervals when applied to* $\mathbf{SA_c}$ *networks.*

Proof. The theorem is proved by showing that if all labels are from $\mathbf{SA_c}$ and the network is path consistent, then the network is k-consistent for all $k \le n$. Hence, the network is strongly n-consistent and the labels between vertices are the minimal labels.

Basis: $k = 1, 2$, or 3. True for $k = 1, 2$ since $\mathbf{SA_c}$ networks are always node and arc consistent and for $k = 3$ by the assumption of path consistency.

Inductive step: We assume strongly $(k-1)$-consistent and show k-consistent, and thus strongly k-consistent. The domain of variable X_i is the set of ordered pairs of real numbers $< X_i^s, X_i^e >$ with $X_i^s < X_i^e$. The inductive assumption implies that variables $X_1, \ldots, X_{k-1}$ can be consistently instantiated. Let $< s_1, e_1 >, \ldots, < s_{k-1}, e_{k-1} >$ be an instantiation such that

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

is satisfied. To show that the network is k-consistent, we must show that there exists at least one instantiation of variable X_k such that

$$< s_i, e_i > \; R_{ik} \; < X_k^s, X_k^e > \qquad i = 1, \ldots, k-1 \tag{3.4}$$

is satisfied. We do so as follows. The $< s_1, e_1 >, \ldots, < s_{k-1}, e_{k-1} >$ restrict the allowed instantiations of X_k. These restrictions, because the network is labeled with elements of $\mathbf{SA_c}$, can be expressed as conjunctions of the relations $\{<, \leq, =, \geq, >, ?\}$ between the endpoints of the intervals. For example, if the relation R_{1k} is the disjunction of the "before" and the "meets" relations,

$$(< s_1, e_1 > \text{ before } < X_k^s, X_k^e >) \quad \text{or} \quad (< s_1, e_1 > \text{ meets } < X_k^s, X_k^e >)$$

the bounds on the instantiations of X_k^s and X_k^e are

$$s_1 < X_k^s, \quad e_1 \leq X_k^s, \quad s_1 < X_k^e, \quad e_1 < X_k^e, \quad \text{and} \quad X_k^s < X_k^e$$

For each i in equation (3.4) we get bounds on instantiations of X_k^s and X_k^e. The key is that all these bounds define convex sets so by Helly's theorem it is sufficient to show that any three bounds have a point in common to show that they all have a point in common. There are two cases depending on whether one of the three bounds is $X_k^s < X_k^e$.

Case 1: Each of the three bounds is strictly in one or the other of X_k^s and X_k^e, the bound that involves both is not included. Because each bound is only in one variable it is sufficient to show that any two bounds have a point in common to show that together the three bounds have a point in common. But any two bounds are always part of a single triangle and have a point in common by the assumption of (strong) path consistency.

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

Hence, all the bounds have a point in common and there exists at least one instantiation of X_k that satisfies equation (3.4) for all i. Because we require that $x_j R_{ji} x_i \equiv x_i R_{ij} x_j$ we have also shown that $< X_k^s, X_k^e > \; R_{ki} \; < s_i, e_i >$, $i = 1, \ldots, k-1$ is satisfied. Hence, we have shown that, for any consistent instantiation of $k-1$ variables, there exists an instantiation of any kth variable such that

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

is satisfied. Hence, the network is k-consistent. This proves the inductive step and thus the theorem. □

Corollary 1. *The path consistency algorithm correctly finds the minimal labels between all pairs of points when applied to* $\mathbf{PA_c}$ *networks, where* $\mathbf{PA_c}$ *is the 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 $\mathbf{SA_c}$ and $\mathbf{PA_c}$. Aho et al. (1974) give an algorithm for the algebraic path problem that achieves path consistency if certain conditions hold. $\mathbf{SA_c}$ and $\mathbf{PA_c}$ can be shown to meet these conditions with one provision. Composition does distribute over intersection *provided* the intersection of two labels is not equal to the empty set (Proposition 1). Clearly, this will not always be true. Does this restrict the applicability of the algorithm? Fortunately not. In the algorithm, the intersection of two labels to get the empty set means the network is inconsistent. If this occurs the algorithm can stop and report inconsistency.

3.2.2. Easy Special Cases of the New Consistency Algorithm

In this section we show that our new all-to-all consistency algorithm (AAC) is exact for **SA** and **PA** networks. The strategy is to first identify why path consistency is not sufficient and where the proof of Theorem 3 fails for **SA** and **PA** networks (recall **PA** includes $\neq$, $\mathbf{PA_c}$ does not).

In the proof of Theorem 3 for the exactness of path consistency for $\mathbf{SA_c}$ and $\mathbf{PA_c}$ networks 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 defined convex sets. If $\neq$ is permitted in the language of the point algebra, the bounds no longer define convex sets. If the network is path consistent and not inconsistent, the intersection of any two bounds cannot be empty, but the intersection of three can be empty. Here is an example **PA** network.

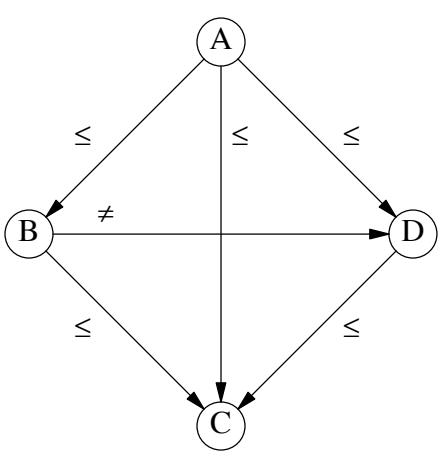


Fig. 3. PA Network Counter-Example

The network 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 a ≤ D, b ≠ D, and c ≥ D. Using standard interval notation and substitution of equals the three bounds are $[a, +\infty)$, $(-\infty, a) \cup (a, +\infty)$, and $(-\infty, a]$. It is easily seen that any two of the bounds have a point in common but together the three bounds have no point in common. If the network was also minimal with respect to all subgraphs of four vertices, say by applying algorithm AAC, the relation between A and C would be < and this counter-example could not occur.

Fig. 3 shows one counter-example of four vertices. But are there other counter-examples of size $n \geq 4$? The following lemma answers this question and is the basis of a proof that the all-to-all consistency algorithm is exact for **SA** and **PA** networks.

Lemma 1. *Any path consistent* **PA** *network for which the labels between vertices are not the minimal labels has a subgraph of four vertices isomorphic to the network in Fig. 3.*

Proof. The networks under consideration are k-consistent for $k = 1, 2, 3$. For the labels of a network not to be the minimal labels there must exist some $k \geq 4$ such that the network is strongly $(k-1)$ consistent but not k-consistent. That is, any $k-1$ variables can be consistently instantiated but the intersection of the bounds on the instantiation of a kth variable are empty, they do not have a point in common.

Summarizing, we know that (i) every two bounds have a point in common, by the assumption of path consistency, (ii) the intersection of all the bounds is empty, and (iii) at least one of the bounds must involve a ≠, otherwise the bounds would define convex sets. But, the sets defined by the bounds and intersections of the bounds are *almost* convex: except for at most $k-1$ holes. So, for the intersection of all the bounds to be empty, it must be that one of the bounds asserts ≠ and the intersection of two or more bounds is 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. Hence, three of the bounds must be $[a, +\infty)$, $(-\infty, a) \cup (a, +\infty)$, and $(-\infty, a]$. And it is easily shown by enumeration that the network in Fig. 3 is the only four-vertex path-consistent network that, up to isomorphism, gives rise to these bounds. □

The counter-example then is unique and cannot occur if the network is minimal with respect to all subgraphs of four vertices. This leads to the following theorem.

Theorem 4. *The all-to-all consistency algorithm of Fig. 2 correctly finds the minimal labels between all pairs of intervals when applied to* **SA** *networks.*

Proof. The proof of theorem 4 follows the inductive proof of theorem 3 except that we can no longer rely on Helly's theorem and the convexity of the sets defined by the bounds to show the bounds have a point in common. The key is that, by the discussion in the proof of Lemma 1, the only way to have $k \geq 3$ bounds pairwise have a point in common but together not have a point in common, is to have the three bounds shown in the example above. These bounds cannot arise if the **SA** network is minimal with respect to all subgraphs of four vertices. Hence, the bounds have a point in common. □

Corollary 2. *The all-to-all consistency algorithm of Fig. 2 correctly finds the minimal labels between all pairs of points when applied to* **PA** *networks.*

We remark that Ghallab and Mounir Alaoui (1989) also give a procedure for **PA** networks. It is based on a structure called a maximal indexed spanning tree and is shown to work well in practice.

4. 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 minimal labels between one interval and every other interval or between two particular intervals then, in computing the minimal labels 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.

4.1. A One-to-All Approximation Algorithm

The algorithm (see Fig. 4) 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. As a result, it produces poor quality approximations when applied to **IA** networks.

In the algorithm of Fig. 4, 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 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 make the algorithm exponential in the worst case. In this context, though, the algorithm is $O(n^2)$.

Theorem 5. *The one-to-all consistency algorithm of Fig. 4 requires $O(n^2)$ time, where n is the number of intervals or points.*

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 one-to-all consistency algorithm requires the operation of finding the minimum of a set of labels. The final result of the algorithm is independent of how the minimum is chosen, but the choice does affect the number of iterations. In practice, choosing the minimum based on the following order on the set of all labels halved the number of iterations of the algorithm compared to a random choice. We assign weights to the 13 basic

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}, $j = 1, \ldots, n$.

```
procedure OAC
begin
    L ← V − { s }
    while L is not empty do begin
        select a vertex v from L such that C_sv is a minimum
        L ← L − { v }
        for each t in V do begin
            l ← C_st ∩ C_sv · C_vt
            if (l ≠ C_st) then begin
                C_st ← l
                L ← L ∪ { t }
            end
        end
    end
end
```

Fig. 4. One-to-All Consistency Algorithm

interval relations based on the sum of the cardinality of the basic relation successively composed with every possible label. The weight of a basic relation is a measure of how restrictive the relation is. With suitable scaling we get the following weights for the 13 basic relations: (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 algorithms.

4.2. 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 (OAC) is exact. Note that the all-to-all algorithms compute approximations to the minimal labels between all pairs of vertices, but even the labels we are not interested in help us by further constraining the labels we are interested in. OAC 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, OAC is exact for the same subset of **IA** for which the path consistency algorithm (PC) is exact.

Theorem 6. *The one-to-all consistency algorithm of Fig. 4 correctly finds the minimal labels between a source interval and every other interval when applied to* $\mathbf{SA_c}$ *networks, provided the network is consistent.*

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

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

$$C_{sj} \subseteq C_{sk} \cdot C_{kj}, \quad j, k = 1, \ldots, n \tag{4.1}$$

Suppose, to the contrary, that there exists a C_{st} such that PC would compute a better approximation than OAC. We know that,

$$C_{st} \subseteq C_{sv} \cdot C_{vt}, \quad v = 1, \ldots, n \tag{4.2}$$

For such a C_{st} to exist, there also must exist some path $v, w_1, w_2, \ldots, w_m, t$ that OAC does not look at but PC would look at, and that constrains the label on the edge (v, t) and invalidates equation (4.2). That is, a path such that

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

But by distributivity (clause (ii) of Proposition 1) 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 (4.1) repeatedly,

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

$$= C_{st}$$

A contradiction. □

From clause (ii) of Proposition 1 stating a distributivity property of $\mathbf{PA_c}$, the point algebra that excludes the $\neq$ relation, we have also proved that OAC is exact for $\mathbf{PA_c}$ networks.

Corollary 3. *The one-to-all consistency algorithm of Fig. 4 correctly finds the minimal labels between a source point and every other point when applied to* $\mathbf{PA_c}$ *networks, provided the network is consistent.*

The proof of the theorem and the corollary uses the property that composition distributes over intersection. By Proposition 1 this property is true for $\mathbf{SA_c}$ and $\mathbf{PA_c}$, respectively, only if it can be guaranteed that the intersection of two labels will never result in the empty set. This corresponds to guaranteeing that the network is consistent. It is easy to show that the above theorem is false if the network is inconsistent.

Thus, to know whether the algorithm has computed the minimal labels we must first know the answer to the decision problem: is the network inconsistent. In (van Beek, 1990) we give an $O(n^2)$ algorithm that answers this decision problem for **SA** and **PA** networks (and thus for $\mathbf{SA_c}$ and $\mathbf{PA_c}$ networks) and so can be used effectively with the one-to-all algorithm given here. Alternatively, there are applications where it is safe to assume that the network will be consistent (see Section 6 where 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).

Finally, for the all-to-all problem we have demonstrated polynomial time algorithms that correctly find the minimal labels for $\mathbf{SA_c}$, $\mathbf{PA_c}$, **SA**, and **PA** networks. For the one-to-all problem we have demonstrated it only for $\mathbf{SA_c}$ and $\mathbf{PA_c}$ networks. An interesting problem for future work is to develop one-to-all algorithms that correctly find the minimal labels for **SA** and **PA** networks.

5. 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.

Distribution 1: About 75% of the time the uncertainty added is I, the set of all basic relations, and the remaining time consists of sets of from 0 to 3 of the basic relations.
Distribution 2: All elements of **IA** are equally likely to be added as uncertainty.

	Distribution 1			Distribution 2		
n	OAC	PC	AAC	OAC	PC	AAC
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

Fig. 5. Percentage differences between the approximation algorithms and an exact algorithm for various problem sizes. 150 tests performed for each problem size, n.

We randomly generated **IA** networks of size n as follows. We first generated an "instantiation" by randomly generating values for the end points of n intervals. This was turned into a consistent instantiation of an **IA** network by determining the basic relations which were satisfied by this instantiation. Finally, we then added indefiniteness to the relations between intervals by adding basic relations.

We then applied the three approximation algorithms, chose a particular edge, determined the minimal 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 indefiniteness is randomly generated. Fig. 5 summarizes the results for two distributions. Distribution one was chosen to approximate instances that may arise in

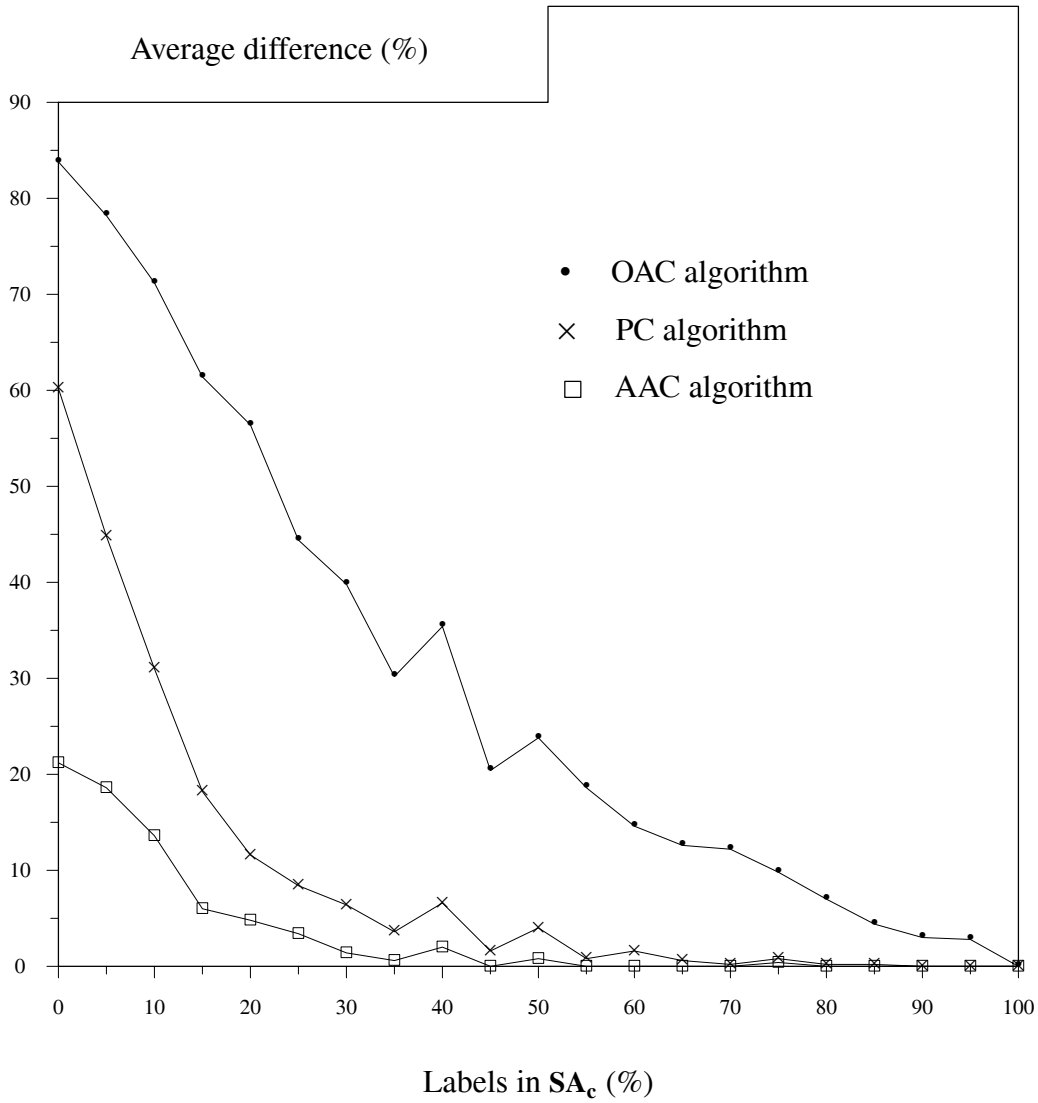


Fig. 6. Percentage differences between the approximation algorithms and an exact algorithm for various percentage of labels in $\mathbf{SA_c}$. 250 tests performed for each subinterval; problem size is 25.

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 (represented as 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 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 the minimal labels.

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 if it is true in general that the quality of the approximation improves as the problem size increases (as exhibited in Fig. 5). There are indications that if some of the labels on edges of a random instance have, before adding indefiniteness, at least two feasible elements, this is *not* the case but few 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 $\mathbf{SA_c}$ be the subset of **IA** discussed earlier for which the path consistency algorithm is exact. Computational evidence shows a strong correlation between the percentage of the total labels that are from $\mathbf{SA_c}$ and how well the OAC, PC, and AAC algorithms approximate the exact solution. Recall that Theorem 3 (Theorem 6) states that PC (OAC) is exact when all the labels are from $\mathbf{SA_c}$ so we cannot improve on that. But, as the percentage of the total labels that are from $\mathbf{SA_c}$ nears zero, up to three-fifths of the labels (on average) assigned by PC and more than four-fifths of the labels assigned by OAC are not the minimal labels (see Fig. 6). Thus we have an effective test for predicting whether it would be useful to apply a more expensive algorithm.

6. Applications

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

Example 6.1.

Koubarakis et al. (1989) use the 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 completeness in favor of guaranteed exact answers in quick time. (To be precise, their system maintains in a $\mathbf{PA_c}$ network the relations between the end points of every interval.) However, the basic interval relations are also a subset of $\mathbf{SA_c}$. Hence, our result that the path consistency algorithm is exact for $\mathbf{SA_c}$ and $\mathbf{PA_c}$ networks 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 $\mathbf{PA_c}$ networks or reason directly with $\mathbf{SA_c}$ networks, 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 6.2.

Song and Cohen (1988) use the 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 **IA** is also a subset of

$\mathbf{SA_c}$. Thus, once we have extracted the possibly vague relations between some of the events mentioned in the narrative, we can determine *exactly* the strongest possible assertions about the relations between all of the events using the path consistency algorithm. As well, we can use the one-to-all algorithm to respond to queries about the ordering of two specific events (such as "Is event A before event B?").

Example 6.3.

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 **IA** 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 (whether useful planning can be done with the possible relations restricted to **SA** and $\mathbf{SA_c}$ is worth further exploration).

Before search begins for a plan (step (ii) above), the planner tests whether the problem specification is temporally consistent. During the search for a plan, the planner adds an operator to the plan if, among other things, the resulting expanded network is temporally consistent. The all-to-all consistency algorithm (AAC) may be useful here as it detects inconsistent networks that the path consistency algorithm does not. Doing these checks for consistency will take resources, but if 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). Allen (1983) suggests using a backtracking algorithm to choose an ordering. The backtracking algorithm tests the consistency of the ordering of the operators selected so far, and backtracks to select again if an inconsistency is detected. In (van Beek, 1990) we use the result that looking at paths is sufficient for deciding whether a **PA** network is inconsistent (a result that follows from Corollary 1 and Lemma 1) as an aid in the design of a backtracking algorithm that is shown to be useful for planning problems (see that reference for the details).

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 basic relation from the disjunction of possible basic relations between two operators and propagating this choice by running the path consistency algorithm. The iterative procedure would stop when no disjunctions remain. A complication is that the procedure may need to backtrack. This would happen if one of the user's choices turned out to be infeasible, but this was only discovered at a later stage. However, the results of the computational experiments suggest that this would be a rare occurrence. The experiments show that the path consistency algorithm almost always determines the minimal labels for problems that arise in planning. Hence, at each step the user can almost always only choose an element of a label that is feasible.

7. Conclusions

We reviewed a popular representation and reasoning framework for qualitative temporal information introduced by James Allen. We then addressed a fundamental reasoning task that arises in applications of this algebra: Given (possibly indefinite) knowledge about the relationships between intervals, find all feasible relationships between two 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.

Acknowledgements. We thank Peter Ladkin for many fruitful discussions and the referees for their detailed comments which improved the paper. We also thank Marc Vilain for showing how to improve one of the proofs and Rina Dechter for asking a question that led to the correction of an overly strong claim.

Appendix

Below we enumerate **SA**, the subset of **IA** that can be translated, using the relations $\{<, \leq, =, \geq, >, ?, \neq\}$, into conjunctions of relations between the endpoints of the intervals. We partition the elements into two sets dependent on whether $\neq$ is required in the translation. Thus, $\mathbf{SA_c}$ is enumerated as well. We remark that $\mathbf{SA_c}$ contains the 13 basic relations of **IA** plus all the entries in the composition table of the basic relations (see Allen, 1983). A^- and A^+ represent the start and end points of interval A, respectively, and $A^- < A^+$ and $B^- < B^+$ are true for every translation. In the interests of succinctness, if R is shown, the inverse of R is not (except, of course, if R is its own inverse).

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

{b,o}	<	<	≠	<	{d,di,o,oi,f,fi}	≠	<	>	?
{d,o}	≠	<	>	<	{d,di,o,oi,m,mi}	≠	≤	≥	≠
{d,oi}	>	<	>	≠	{d,di,o,oi,s,si}	?	<	>	≠
{s,si}	=	<	>	≠	{b,bi,d,di,o,oi,m}	≠	≠	?	≠
{f,fi}	≠	<	>	=	{b,d,di,o,oi,f,fi}	≠	<	≠	?
{b,d,o}	≠	<	≠	<	{b,d,di,o,oi,m,mi}	≠	≤	?	≠
{b,di,o}	<	<	≠	≠	{b,d,di,o,oi,s,si}	?	<	≠	≠
{b,o,s}	≤	<	≠	<	{d,di,o,oi,m,f,fi}	≠	<	≥	?
{b,o,fi}	<	<	≠	≤	{d,di,o,oi,m,s,si}	?	<	≥	≠
{d,o,m}	≠	<	≥	<	{eq,b,d,o,s,f,fi}	?	<	≠	≤
{d,oi,mi}	>	≤	>	≠	{eq,b,di,o,s,si,fi}	≤	<	≠	?
{b,d,o,m}	≠	<	?	<	I −{eq,m,mi,s,si}	≠	≠	≠	?
{b,d,o,s}	?	<	≠	<	I −{eq,s,si,f,fi}	≠	?	?	≠
{b,di,o,fi}	<	<	≠	?	I −{eq,m,mi,f,fi}	?	≠	≠	≠
{b,di,o,m}	<	<	?	≠	I −{eq,bi,mi,s,si}	≠	≠	≥	?
{d,di,o,oi}	≠	<	>	≠	I −{eq,bi,mi,f,fi}	?	≠	≥	≠
{d,o,f,fi}	≠	<	>	≤	I −{eq,bi,m,s,si}	≠	?	>	?
{d,oi,s,si}	≥	<	>	≠	I −{eq,bi,m,f,fi}	?	?	>	≠
{b,d,di,o,oi}	≠	<	≠	≠	I −{eq,b,bi,s,si}	≠	≤	≥	?
{b,d,o,f,fi}	≠	<	≠	≤	I −{eq,b,bi,f,fi}	?	≤	≥	≠
{b,di,o,s,si}	≤	<	≠	≠	I −{eq,mi,s,si}	≠	≠	?	?
{d,di,o,oi,m}	≠	<	≥	≠	I −{eq,mi,f,fi}	?	≠	?	≠
{d,o,m,f,fi}	≠	<	≥	≤	I −{eq,bi,s,si}	≠	≤	?	?
{d,oi,mi,s,si}	≥	≤	>	≠	I −{eq,bi,f,fi}	?	≤	?	≠
{eq,b,o,s,fi}	≤	<	≠	≤	I −{eq,s,si}	≠	?	?	?
{b,bi,d,di,o,oi}	≠	≠	≠	≠	I −{eq,f,fi}	?	?	?	≠
{b,d,di,o,oi,mi}	≠	≤	≠	≠	I −{bi,m,mi}	?	<	≠	?
{b,d,di,o,oi,m}	≠	<	?	≠	I −{m,mi}	?	≠	≠	?
{b,d,o,m,f,fi}	≠	<	?	≤	I −{bi,m}	?	≤	≠	?
{b,di,o,m,s,si}	≤	<	?	≠	I −{mi}	?	≠	?	?

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.

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

Dechter, R., I. Meiri, and J. Pearl. 1989. Temporal Constraint Networks. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ont., 83-93.

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.

Ghallab, M., and A. Mounir Alaoui. 1989. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Mich., 1297-1303.

Granier, T. 1988. Contribution à l'étude du temps objectif dans le raisonnement. Rapport LIFIA RR 716-I-73, Grenoble. Cited in: Ghallab, M., and A. Mounir Alaoui. 1989.

Hogge, J. C. 1987. TPLAN: A Temporal Interval-Based Planner with Novel Extensions. University of Illinois Department of Computer Science Technical Report UIUCDCS-R-87-1367, 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. Knowledge Representation and Reasoning Technical Report KRR-TR-89-4, Department of Computer Science, University of Toronto.

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

Ladkin, P. B., and R. Maddux. 1988b. The Algebra of Constraint Satisfaction Problems and Temporal Reasoning. Technical Report, Kestrel Institute, Palo Alto, Calif.

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.

Näkel, K. 1988. Convex Relations Between Time Intervals. SEKI Report SR-88-17, Universität Kaiserslautern, W. Germany.

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.

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.

van Beek, P. 1989. Approximation Algorithms for Temporal Reasoning. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Mich., 1291-1296.

van Beek, P. 1990. Reasoning about Qualitative Temporal Information. *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI),* Boston, Mass.

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.

Vilain, M., H. Kautz, and P. van Beek. 1989. Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report. In *Readings in Qualitative Reasoning about Physical Systems*, D. S. Weld and J. de Kleer (eds.), Morgan-Kaufman, 373-381.