Reasoning about Qualitative Temporal Information*

Peter van Beek
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
vanbeek@cs.ualberta.ca

Appears in: *Artificial Intelligence,* 58:297-326, 1992.

Abstract

Representing and reasoning about incomplete and indefinite qualitative temporal information is an essential part of many artificial intelligence tasks. An interval-based framework and a point-based framework have been proposed for representing such temporal information. In this paper, we address two fundamental reasoning tasks that arise in applications of these frameworks: Given possibly indefinite and incomplete knowledge of the relationships between some intervals or points, (i) find a scenario that is consistent with the information provided, and (ii) find the feasible relations between all pairs of intervals or points.

For the point-based framework and a restricted version of the interval-based framework, we give computationally efficient procedures for finding a consistent scenario and for finding the feasible relations. Our algorithms are marked improvements over the previously known algorithms. In particular, we develop an $O(n^2)$ time algorithm for finding one consistent scenario that is an $O(n)$ improvement over the previously known algorithm, where n is the number of intervals or points, and we develop an algorithm for finding all the feasible relations that is of far more practical use than the previously known algorithm. For the unrestricted version of the interval-based framework, finding a consistent scenario and finding the feasible relations have been shown to be NP-complete. We show how the results for the point algebra aid in the design of a backtracking algorithm for finding one consistent scenario that is shown to be useful in practice for planning problems.

*A preliminary version of this paper appeared in the *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Mass., 1990. AAAI Press / The MIT Press, Cambridge, Mass., pp. 728-734.

1 Introduction

Representing and reasoning about incomplete and indefinite qualitative temporal information is an essential part of many artificial intelligence tasks. Allen [2] has proposed an interval algebra framework and Vilain and Kautz [41] have proposed a point algebra framework for representing such qualitative information. The frameworks are influential and have been applied in such diverse areas as natural language processing [3], planning [4], plan recognition [20], and diagnosis [18]. In this paper, we address two fundamental temporal reasoning tasks that arise in these application areas: Given possibly indefinite and incomplete knowledge of the relations between some intervals or points,

- find a scenario that is consistent with the information provided, and
- find the feasible relations between all pairs of intervals or points.

The frameworks have in common that the representations of temporal information can be viewed as binary constraint networks and that constraint satisfaction techniques can be used to reason about the information.

For point algebra networks and a restricted class of interval algebra networks, we present new, more efficient, algorithms for both of the reasoning tasks. In particular, for finding one consistent scenario, we develop an $O(n^2)$ time algorithm that is an $O(n)$ improvement over the previously known algorithm [22], where n is the number of points. For finding the feasible relations, we develop an $O(\max(mn^2, n^3))$ time algorithm for finding all pairs of feasible relations, where n is the number of points and m is the number of pairs of points that are asserted to be not equal. The new algorithm is of far more practical use than the previously known algorithm [40].

For general interval algebra networks, finding a consistent scenario and finding the feasible relations has been shown to be NP-complete and thus almost assuredly intractable in the worst case [41, 42]. For finding a consistent scenario, we show how the results for the point algebra aid in the design of a backtracking algorithm. The algorithm is shown experimentally to be useful in practice for planning problems and problems with similar characteristics. For finding the feasible relations, the intractability of finding solutions for the general problem has led us elsewhere to explore algorithms that find approximate solutions [40].

The rest of the paper proceeds as follows. We begin by reviewing each of the frameworks and illustrating each with examples from natural language, showing how the temporal information is represented and giving examples of the reasoning tasks. We then formalize the reasoning tasks as binary constraint satisfaction problems. We then develop our algorithms for solving the two reasoning tasks, first for point algebra networks and a restricted class of interval algebra networks and, second for general interval algebra networks.

2 Representing Temporal Information

In this section, we review Allen's framework [2] for representing relations between intervals and Vilain and Kautz's framework [41] for representing relations between points, and illustrate the kinds of temporal information that can be represented within each framework. We then formalize the reasoning tasks using networks of binary constraints [27].

2.1 Allen's framework

There are thirteen **basic** relations that can hold between two intervals (see Fig. 1, [2, 6]). In order to represent indefinite information, the relation between two intervals is allowed to be a disjunction of the basic relations. Sets are used to list the disjunctions. For example, the relation {m,o,s} between events A and B represents the disjunction,

$$\text{(A meets B)} \vee \text{(A overlaps B)} \vee \text{(A starts B)}.$$

Let I be the set of all basic relations, {b,bi,m,mi,o,oi,s,si,d,di,f,fi,eq}. Allen allows the relation between two events to be any subset of I.

We use a graphical notation where vertices represent events and directed edges are labeled with sets of basic relations. 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). Any edge for which we have no explicit knowledge of the relation is labeled with I; by convention such edges are also not shown. We call networks with labels that are arbitrary subsets of I, interval algebra or **IA** networks.

Example 1. As an example of representing temporal information using **IA** networks and of the reasoning tasks of finding a consistent scenario and of finding the feasible relations, consider the description of events shown in Fig. 2a. Not all of the temporal relations between events are explicitly or unambiguously given in the description. The first sentence tells us only that the interval of time over which Fred read the paper intersects with the interval of time over which Fred ate breakfast. We represent this as "paper {o,oi,s,si,d,di,f,fi,eq} breakfast." The second sentence fixes the relationship between some of the end points of the intervals over which Fred read his paper and over which Fred drank his coffee but it remains indefinite about others. We represent this as "paper {o,s,d} coffee."[1] But we also know that drinking coffee is a part of breakfast and so occurs during breakfast. We represent this as "coffee {d} breakfast." Finally, the information in the third sentence is represented as "walk {bi} breakfast." The resulting network is shown in Fig. 2a, where we have drawn a directed edge from "breakfast" to "walk" and so have labeled the edge with the inverse of the "bi" (after) relation.

[1] Another possibility is the relation {b,m,o,s,d}, since the scenario where reading the paper occurred entirely before drinking the coffee is not explicitly ruled out by the sentence.

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

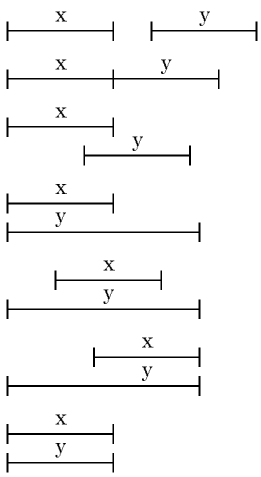


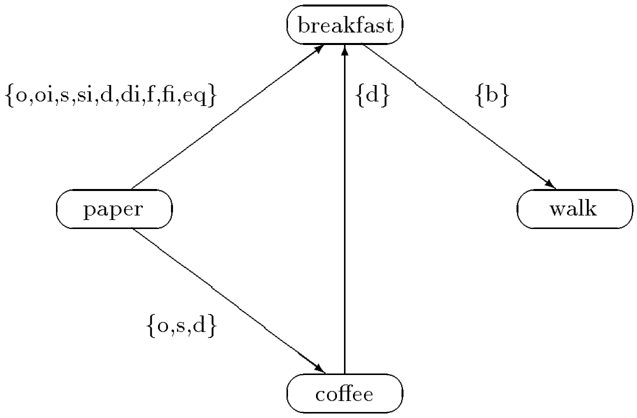
Figure 1: Basic relations between intervals

One scenario consistent with the description of events is shown in Fig. 2c. Another possible consistent scenario is one where Fred starts to read his paper before he starts his breakfast. The feasible relations between all pairs of intervals are shown in Fig. 2b. Determining the feasible relations can be viewed as determining the deductive consequences of our temporal knowledge. We are able to derive, for example, that Fred went for a walk after reading his paper and drinking his coffee and that Fred finished his paper before he finished his breakfast.

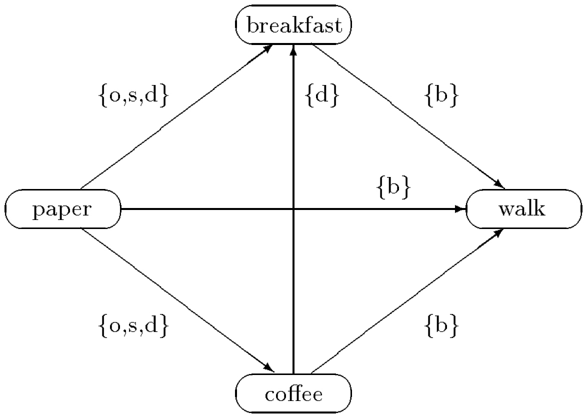
2.2 Vilain and Kautz's framework

There are three **basic** relations that can hold between two points: $<$, $=$, and $>$. In order to represent indefinite information, the relation between two points is allowed to be a disjunction of the basic relations. Sets are used to list the disjunctions. For example, the relation $\{<,=\}$ between points A and B represents the disjunction, $(A < B) \vee (A = B)$. Let ? be the set of all basic relations, $\{<,=,>\}$. The set of possible relations between two points is $\{\emptyset, <, \leq, =, >, \geq, \neq, ?\}$, where $\leq$, for example, is an abbreviation of $\{<,=\}$. We call networks with labels that are subsets of ?, point algebra or **PA** networks.

(a) Example: *Fred was reading the paper while eating his breakfast. He put the paper down and drank the last of his coffee. After breakfast he went for a walk.*



(b) Feasible relations:



(c) Consistent scenario:

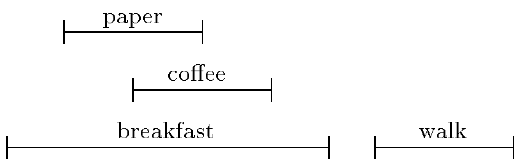
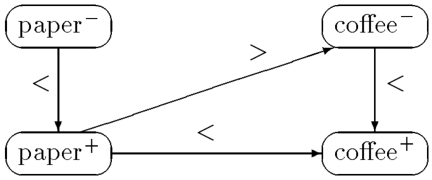


Figure 2: Representing qualitative relations between intervals

(a) Example: *Fred put the paper down and drank the last of his coffee.*



(b) Consistent scenario:

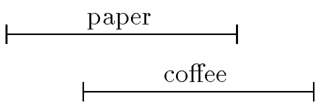


Figure 3: Representing qualitative relations between points

Example 2. As an example of representing temporal information using **PA** networks, consider the description of events shown in Fig. 3a. As discussed in Ex. 1, this sentence fixes the relationship between some of the end points of the intervals of time over which Fred read his paper and over which Fred drank his coffee but it remains indefinite about others. We represent this by the network shown in Fig. 3a, where paper$^-$ and paper$^+$ represent the start and end points of the event. One scenario consistent with the temporal information is shown in Fig. 3b. If a directed edge from paper$^-$ to coffee$^+$ labeled $<$ was added to the network shown in Fig. 3a, the resulting network would show the feasible relations between all pairs of points.

2.3 Translations between representations

Vilain and Kautz [41] show that a restricted class of **IA** networks, denoted here as **SA** networks, can be translated without loss of information into **PA** networks. In **IA** networks, the relation between two intervals can be any subset of I, the set of all thirteen basic relations. In **SA** networks, the allowed relations between two intervals are only those subsets of I that can be translated, using the relations $\{<, \leq, =, >, \geq, \neq, ?\}$, into conjunctions of relations between the endpoints of the intervals. For example, the **IA** network in Fig. 2a is also an **SA** network. As a specific example, the part of the interval network "paper {o,s,d} coffee" can be expressed as the conjunction of point relations,

$$(\text{paper}^- < \text{paper}^+) \wedge (\text{coffee}^- < \text{coffee}^+) \wedge$$
$$(\text{paper}^+ > \text{coffee}^-) \wedge (\text{paper}^+ < \text{coffee}^+),$$

and the equivalent representation as a **PA** network is shown in Fig. 3a, where paper$^-$ and paper$^+$ represent the start and end points of the interval denoted paper, respectively. (See [40] for an enumeration of the allowed relations for **SA** networks and the translation into **PA** relations; also enumerated by Güther [17] and Ladkin and Maddux [23], where the relations are called the "pointisable" relations).

The allowed relations for **SA** networks is a small but important and useful subset of the 2^{13} relations allowed for **IA** networks, as many applications of **IA** networks in the literature actually only use **SA** networks. For example, Almeida [5] and Song [35], in independent work on computer understanding of English narratives, both adopt Allen's framework but choose to use only relations that are allowed for **SA** networks. Hamlet and Hunter [18] adopt Allen's framework for representing temporal information in medical expert systems but, with the exception of the disjointedness relation {b,bi,m,mi}, choose to use only relations that are allowed for **SA** networks (in their example, later temporal information is used to strengthen the disjointedness relation to {b,m} which is allowed). Nökel [29] uses **SA** networks in a diagnostic setting. With the exception of Nökel [29], it does not appear that the authors intentionally restricted their representation language or were aware of the computational advantages; rather, the relations used were simply the right ones for the task at hand.

As alluded to above, what *cannot* be expressed in **SA** networks that can be expressed in **IA** networks is "disjointedness" of intervals. For example, we cannot say that "A {b,bi} B", i.e., that A is either before or after B, since this interval relation cannot be expressed as simply a conjunction of point relations between the endpoints of the two intervals. It requires the disjunction,

$$((\mathrm{A}^- < \mathrm{B}^-) \wedge (\mathrm{A}^- < \mathrm{B}^+) \wedge (\mathrm{A}^+ < \mathrm{B}^-) \wedge (\mathrm{A}^+ < \mathrm{B}^+)) \vee \\ ((\mathrm{A}^- > \mathrm{B}^-) \wedge (\mathrm{A}^- > \mathrm{B}^+) \wedge (\mathrm{A}^+ > \mathrm{B}^-) \wedge (\mathrm{A}^+ > \mathrm{B}^+)).$$

The nearest approximation using only conjunction is,

$$(\mathrm{A}^- \neq \mathrm{B}^-) \wedge (\mathrm{A}^- \neq \mathrm{B}^+) \wedge (\mathrm{A}^+ \neq \mathrm{B}^-) \wedge (\mathrm{A}^+ \neq \mathrm{B}^+).$$

and so, the nearest approximation to "A {b,bi} B" in **SA** networks is "A {b,bi,o,oi,d,di} B". But as can be seen, this allows, for example, A to overlap B, which we did not intend.

2.4 Formalization of the reasoning tasks

We formalize our reasoning tasks using networks of binary constraints [27]. The reasoning tasks are then special cases of a general class of problems known as constraint satisfaction problems. Our development borrows from that found in Dechter et al. [12] and Ladkin and Maddux [22, 23]. This approach allows us to use some previously known algorithms and eases the development of new algorithms.

A **network of binary constraints** [27] is defined as a set X of n variables $\{x_1, x_2, ..., x_n\}$, a domain D_i of possible values for each variable, and binary constraints between variables. A **binary constraint**, C_{ij}, between variables x_i and x_j, is a subset of the Cartesian product of their domains that specifies the allowed pairs of values for x_i and x_j (i.e., $C_{ij} \subseteq D_i \times D_j$). For the networks of interest here, we require that $(x_j, x_i) \in C_{ji} \Leftrightarrow (x_i, x_j) \in C_{ij}$. An **instantiation** of the variables in X is an n-tuple $(X_1, X_2, ..., X_n)$, 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 constraints between variables are satisfied. A network is **inconsistent** if no consistent instantiation exists.

An **IA network** is a network of binary constraints where the variables represent time intervals, the domains of the variables are the set of ordered pairs of rational numbers $\{(s, e) \mid s < e\}$, with s and e representing the start and end points of the interval, respectively, and the binary constraints between variables are represented implicitly by sets of the basic interval relations[2]. For example, let $C_{ij} = \{\text{m,o}\}$ be the relation between variables x_i and x_j in some **IA** network. The set of allowed pairs of values for variables x_i and x_j is given by,

$$\{((s_i, e_i), (s_j, e_j)) \mid (s_i, e_i) \text{ meets } (s_j, e_j) \ \vee \ (s_i, e_i) \text{ overlaps } (s_j, e_j)\}.$$

A **PA network** is a network of binary constraints where the variables represent time points, the domains of the variables are the set of rational numbers, and the binary constraints between variables are represented implicitly by sets of the basic point relations[3].

The reasoning tasks that we want to solve are finding a consistent scenario and finding the feasible relations. A network S is a **consistent scenario** of a network C if and only if

(a) $S_{ij} \subseteq C_{ij}$,

(b) $|S_{ij}| = 1$, for all i, j, and

(c) there exists a consistent instantiation of S.

The basic relations are disjoint. Hence, if an instantiation of variables x_i and x_j satisfies C_{ij}, then one and only one of the basic relations in C_{ij} is satisfied. Thus, given a consistent instantiation of a **PA** or **IA** network, the basic relations

[2] Our interests are in temporal reasoning and hence we speak of *time* intervals. However, **IA** networks and the results presented in this paper have other applications. Two examples are DNA sequencing and optimal arrangement of records on secondary storage (see [16, pp. 182-184]). As well, it should be noted that, by adopting the rationals as the underlying representation of time, we are committing ourselves to a particular view of time, namely, that time is dense, linear, and unbounded. This is appropriate in many temporal reasoning applications. In other applications, however, we may want discrete, branching, or bounded time.

[3] With the exclusion of the $\neq$ relation, a **PA** network is simply a system of linear inequalities where each inequality is in two variables and each variable has unit coefficient. This is discussed further in Section 3.1. Along similar lines, Dean and McDermott [8] and Dechter et al. [12] propose difference constraints, and Malik and Binford [26] propose linear inequalities to represent and reason about temporal information.

between variables satisfied by that consistent instantiation define a consistent scenario. As an example, one possible consistent instantiation of the network in Fig. 2a that would give the consistent scenario in Fig. 2c is, paper $\leftarrow (1,3)$, breakfast $\leftarrow (0,5)$, walk $\leftarrow (6,7)$, and coffee $\leftarrow (2,4)$. While there are either zero or an infinite number of different consistent instantiations of a **PA** or **IA** network, there are only a finite number of different consistent scenarios.

A basic relation $B \in C_{ij}$ is **feasible** with respect to a network if and only if there exists a consistent instantiation of the network where B is satisfied. Given an **IA** network or a **PA** network, C, the **set of feasible relations** between two variables x_i and x_j in the network is the set consisting of *all and only* the $B \in C_{ij}$ that are feasible. The **minimal** network representation, M, of a network, C, is the network for which M_{ij} is the set of feasible relations between variables x_i and x_j in C for every $i, j = 1, \ldots, n$. As an example, the network in Fig. 2b is the minimal network of the network in Fig. 2a.

3 Point Algebra Networks and a SubClass of Interval Algebra Networks

In this section we examine the computational problems of finding consistent scenarios and finding the feasible relations of **PA** networks and **SA** networks.

3.1 Finding a consistent scenario

Related work. One method of finding a consistent scenario of a **PA** network is to first find a consistent instantiation of the network. The basic relations between variables satisfied by the consistent instantiation then give a consistent scenario.

Topological sort (see Knuth [21]) can be used to find a consistent instantiation if the temporal information is a strict partial order; i.e., if the allowed relations are restricted to $\{<, >, ?\}$. Topological sort is $O(n^2)$.

If the allowed relations are restricted to $\{<, \leq, =, >, \geq, ?\}$, i.e., we do not allow disequality, **PA** networks can be viewed as a set of linear equalities and inequalities. The equalities and inequalities are of the form: $x_i - x_j < 0$, $x_i - x_j \leq 0$, and $x_i - x_j = 0$. A solution to the set of linear inequalities is precisely a consistent instantiation of the network. Solving a set of linear inequalities—or recognizing that no solution exists—is easily done using algorithms for solving linear programs (see Chvátal [7]). Thus, the simplex algorithm or Karmarkar's algorithm can be used to find a solution. However, more efficient algorithms are known if the linear program is of a particular form that arises in what is known as the shortest-path problem.

The shortest-path problem is to find the shortest path in a labeled graph from a vertex s to a vertex t. This can be made into a linear program as follows (see Papadimitriou and Steiglitz [31]). Let l_{ij} be the label on the directed edge (i, j) and let x_i denote the length of the shortest path from s to i. The shortest path from s to itself is 0. We want to minimize x_t, the length of the shortest path from s to t. Since the shortest path from s to j might pass through i, we must have $x_j \leq x_i + l_{ij}$, i.e., $x_j - x_i \leq l_{ij}$. The result is a linear program of the form,

$$\begin{aligned}
&\min x_t \\
&x_i - x_j \leq l_{ij}, \quad i, j = 1, \ldots, n \\
&x_i \text{ unconstrained} \\
&x_s = 0.
\end{aligned}$$

If all the l_{ij} are non-negative then we can use Dijkstra's algorithm [13] to find a solution to this linear program. Dijkstra's algorithm is $O(n^2)$. If some of the l_{ij} are negative then we can use the Floyd-Warshall algorithm [1] to find a solution. The Floyd-Warshall algorithm is $O(n^3)$.

It remains to show how much of our problem can be translated into a shortest-path problem. The translation is as follows,

$$
\begin{array}{lcll}
x_i = x_j & \rightarrow & x_i - x_j \leq 0, & x_j - x_i \leq 0, \\
x_i \leq x_j & \rightarrow & x_i - x_j \leq 0, & x_j - x_i \leq +\infty, \\
x_i < x_j & \rightarrow & x_i - x_j \leq -\epsilon, & x_j - x_i \leq +\infty,
\end{array}
$$

where the left column shows the relation between variables in a **PA** network and the right columns shows the translation into constraints for the shortest-path linear program. Note the use of a small negative value, $-\epsilon$, for turning a strict inequality into a weak inequality (see [7, p. 451] for how to choose a value for ϵ such that solutions are preserved and no new solutions are introduced). In summary, if the allowed relations are restricted to $\{\leq, =, \geq, ?\}$, then Dijkstra's algorithm can be used to find a consistent instantiation in $O(n^2)$ time. If the allowed relations are restricted to $\{<, \leq, =, >, \geq, ?\}$, then the Floyd-Warshall algorithm can be used to find a consistent instantiation in $O(n^3)$ time.

Finally, Ladkin and Maddux [22] give an algorithm for finding one consistent scenario for **PA** networks that takes $O(n^3)$ time with n points. If no consistent scenario exists, the algorithm reports the inconsistency. Their algorithm relies on first applying the path consistency algorithm [24, 27] before finding a consistent scenario.

An improved algorithm. We develop an algorithm for finding one consistent scenario that takes $O(n^2)$ time for **PA** networks with n points. Our starting point is an observation by Ladkin and Maddux [22, p. 34] that topological sort alone will not work as the labels may be any one of the eight different relations, $\{\emptyset, <, \leq, =, >, \geq, \neq, ?\}$, and thus may have less information about the relation between two points than is required. For topological sort we need all edges labeled with $<$, $>$, or ?. The "problem" labels are then $\{=, \emptyset, \leq, \geq, \neq\}$. The intuition behind the algorithm is that we somehow remove or rule out each of these possibilities and, once we have, we can then apply topological sort to give a consistent scenario. The algorithm is summarized in Fig. 6 and a proof of correctness is given in Appendix A. The input to the algorithm is a **PA** network represented as an adjacency matrix C where element C_{ij} is the label on edge (i, j).

The = relation. To remove the = relation from the network, we identify all pairs of points that are necessarily equal and condense them into one vertex. When saying a pair of points are necessarily equal, we mean that in every consistent scenario the relation between the two vertices is the = relation. More formally, we want to partition the vertices into equivalence classes S_i, $1 \leq i \leq m$, such that vertices v and w are in the same equivalence class if and only if they are necessarily equal. It turns out that the vertices v and w are necessarily equal precisely when there is a cycle of the form,

$$v \leq \cdots \leq w \leq \cdots \leq v,$$

where one or more of the $\leq$ can be = (see Appendix A for a proof). This is the same as saying v and w are in the same equivalence class if and only if there is a

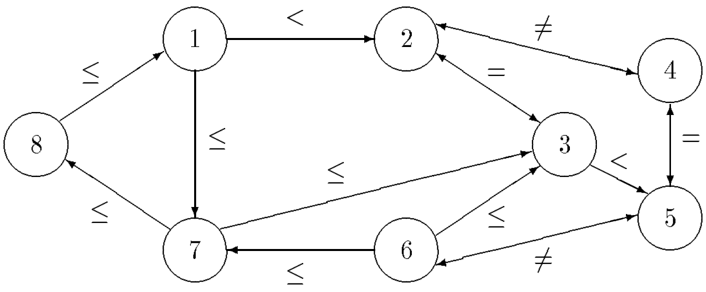


Figure 4: Example **PA** network

path from v to w and a path from w to v using only the edges labeled with $\leq$ or $=$. This is a well-known problem in graph theory. Determining the equivalence classes is the same as identifying the strongly connected components (SCCs) of the graph and an efficient $O(n^2)$ algorithm is known (Tarjan [36]).

We condense the graph by collapsing each strongly connected component into a single vertex. Let $\{S_1, S_2, \ldots, S_m\}$ be the SCCs we have found. The S_i partition the vertices in the graph in that each vertex is in one and only one of the S_i. We construct the condensed graph and its matrix representation, $\hat{C}$, as follows. Each S_i is a vertex in the graph. The labels on the edges between all pairs of vertices is given by,

$$\hat{C}_{S_i S_j} \leftarrow \bigcap_{\substack{v \in S_i \\ w \in S_j}} C_{vw}, \quad i, j = 1, \ldots, m.$$

Example 3. The network shown in Fig. 4 is used to illustrate the discussion. As usual, all edges (i, i) and all edges labeled ? are omitted. The four strongly connected components, S_1, S_2, S_3, and S_4, of the network are as shown in Fig. 5a. The condensed graph of the network of Fig. 4 is shown in Fig. 5b. To illustrate, condensing the strongly connected component S_1 gives,

$$\begin{aligned} \hat{C}_{S_1 S_1} &\leftarrow C_{17} \cap C_{18} \cap C_{71} \cap C_{78} \cap C_{81} \cap C_{87} \\ &\leftarrow \{<, =\} \cap \{>, =\} \cap \{>, =\} \cap \{<, =\} \cap \{<, =\} \cap \{>, =\} \\ &\leftarrow \{=\}, \end{aligned}$$

where we have omitted the self loops C_{ii} (these loops are always labeled with $\{=\}$ and so do not affect the result). As a further illustration, the labels on the edges between S_2 and S_3 are given by,

$$\begin{aligned} \hat{C}_{S_2 S_3} &\leftarrow C_{24} \cap C_{25} \cap C_{34} \cap C_{35} \\ &\leftarrow \{<, >\} \cap \{<, =, >\} \cap \{<, =, >\} \cap \{<\} \\ &\leftarrow \{<\}. \end{aligned}$$

(a) Strongly connected components:

$$\begin{array}{ll} S_1 = \{1, 7, 8\}, & S_3 = \{4, 5\}, \\ S_2 = \{2, 3\}, & S_4 = \{6\}. \end{array}$$

(b) Condensed PA network:

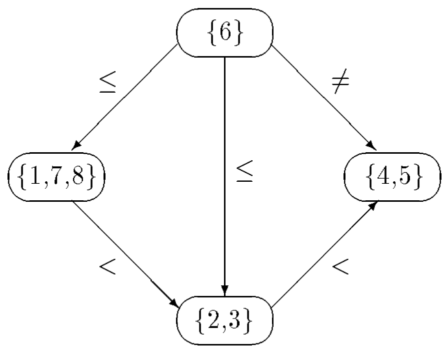


Figure 5: Condensing the strongly connected components

The $\emptyset$ relation. To rule out the $\emptyset$ relation we must determine if the network is inconsistent. It turns out that the network is inconsistent precisely when there is a cycle of the form,

$$v = \cdots = w \neq v,$$

or of the form,

$$v \leq \cdots \leq w \leq \cdots \leq v \neq w,$$

where some or all of the $\leq$ can be $=$, or of the form,

$$v < \cdots < w < \cdots < v,$$

where all but one of the $<$ can be $\leq$ or $=$ (see Appendix A for a proof). The first two cases are already detected when we identify all pairs of points that are necessarily equal and condense them into one vertex. That is, the inconsistencies are detected when the strongly connected components are condensed. But we can identify the third case simply by also looking at edges labeled with $<$ when identifying the strongly connected components. As before, the inconsistencies are then detected when the strongly connected components are condensed.

CSPAN(C)

1. Identify the strongly connected components (SCCs) of C using only edges labeled with $\{<\}$, $\{<,=\}$, and $\{=\}$. Let $S_1, \ldots, S_m$ be the SCCs found.
2. **for** $i, j \leftarrow 1, \ldots, m$
3.   **do** $\hat{C}_{S_i S_j} \leftarrow \{<,=,>\}$
4.   **for** each $v \in S_i$, $w \in S_j$
5.   **do** $\hat{C}_{S_i S_j} \leftarrow \hat{C}_{S_i S_j} \cap C_{vw}$
6.   **if** $\hat{C}_{S_i S_j} = \emptyset$
7.   **then return**("Inconsistent network")
8. Replace any remaining $\{<,=\}$ labels in $\hat{C}$ with $\{<\}$.
9. Perform a topological sort using only the edges in $\hat{C}$ labeled with $\{<\}$.

Figure 6: Consistent scenario algorithm for **PA** networks

Example 4. Suppose the label on the edge (1, 7) in the graph shown in Fig. 4 was $<$ instead of the $\leq$ shown. Condensing the strongly connected component S_1 would give,

$$\begin{aligned} \hat{C}_{S_1 S_1} &\leftarrow C_{17} \cap C_{18} \cap C_{71} \cap C_{78} \cap C_{81} \cap C_{87} \\ &\leftarrow \{<\} \cap \{>,=\} \cap \{>\} \cap \{<,=\} \cap \{<,=\} \cap \{>,=\} \\ &\leftarrow \emptyset, \end{aligned}$$

where again we have omitted the self loops C_{ii}.

The $\leq, \geq$ relations. To remove the $\leq$ relation from the network, we simply change all $\leq$ labels to $<$. This is valid because, assuming that the $\emptyset$ and $=$ relations have been removed, we know that a consistent scenario exists and that no remaining edge is forced to have $=$ as its label in all consistent scenarios. So, for any particular edge labeled with $\leq$ there exists a consistent scenario with $<$ as the singleton label. But, changing a $\leq$ to a $<$ can only force other labels to become $<$; it cannot force labels to become $=$. (Using the terminology of the algorithm in Fig. 6, no new strongly connected components are introduced by this step; hence no new labels are forced to be equal and no new inconsistencies are introduced.) So, after all the changes, a consistent scenario still exists.

The $\neq$ relation. We can now perform topological sort to find one consistent scenario. It can be shown that, because of the previous steps of the algorithm, the $\neq$ relations are now handled correctly (and implicitly) by topological sort. The output of topological sort is an assignment of numbers to the vertices that is consistent with the information provided.

Example 5. Consider the network shown in Fig. 5b. Depending on the particular implementation of topological sort, one possible result is that vertex {6} is assigned the number 0, vertex {1, 7, 8} is assigned 1, vertex {2, 3} is assigned 2, and vertex {4, 5} is assigned 3. The consistent scenario of the original network (Fig. 4) is easily recovered from this information.

Theorem 1 *Procedure* CSPAN *correctly finds a consistent scenario of a* **PA** *network in $O(n^2)$ time, where n is the number of points.*

Proof. See Appendix A for a detailed proof of correctness. For the time bound, finding the strongly connected components is $O(n^2)$ [36], condensing the graph looks at each edge only once, and topological sort is $O(n^2)$ [21]. □

We can find a consistent scenario of an **SA** network by first translating it into a **PA** network and using algorithm CSPAN to find a consistent instantiation. The consistent instantiation of the **PA** network also gives a consistent instantiation of the original **SA** network and hence also defines a consistent scenario of the original **SA** network. Each of the steps of (i) recognizing that an **IA** network is the special case of an **SA** network, (ii) translating it into a **PA** network, and (iii) finding a consistent scenario, can be done in $O(n^2)$ time.

Example 6. Consider the (small) **SA** subnetwork shown in Fig. 2a consisting of paper {o,s,d} coffee and its translation into a **PA** network shown in Fig. 3a. One consistent instantiation of the corresponding **PA** network is the assignments, paper$^-$ $\leftarrow$ 1, coffee$^-$ $\leftarrow$ 2, paper$^+$ $\leftarrow$ 3, and coffee$^+$ $\leftarrow$ 4. The corresponding consistent instantiation of the original **SA** network is simply, paper $\leftarrow (1,3)$ and coffee $\leftarrow (2,4)$, and the consistent scenario is given by, "paper overlaps coffee".

To summarize, if our temporal networks are **PA** networks or **SA** networks we can find a consistent scenario quickly using algorithm CSPAN.

3.2 Finding the feasible relations

Related work. Allen [2] shows that a path consistency algorithm [27, 24] can be used to find an approximation to the sets of all feasible relations (see Fig. 8; the path consistency procedure shown there is due to Mackworth [24] but is slightly simplified because of properties of the algebras). Path consistency algorithms, as their name suggests, ensure that a network is path consistent. A network is **path consistent** [24] if and only if, for every triple (i,k,j) of vertices,

$$\forall x_i \forall x_j [(x_i, x_j) \in C_{ij} \Rightarrow \exists x_k (x_k \in D_k \wedge (x_i, x_k) \in C_{ik} \wedge (x_k, x_j) \in C_{kj})].$$

In words, for every instantiation of x_i and x_j that satisfies the direct relation, C_{ij}, there exists an instantiation of x_k such that C_{ik} and C_{kj} are also satisfied.

To use the path consistency algorithm, we need the operators composition and intersection of relations (see [2, 41] for discussions of how the operations are implemented in this context).

Previous work has identified classes of relations for which the path consistency algorithm will find the minimal network. Montanari [27] shows that the path consistency algorithm finds the minimal network for a restricted class of binary constraint relations. However, the relations of interest here do not all fall into this class. Valdés-Pérez [37] shows that the path consistency algorithm finds the minimal network for **IA** networks which use only the basic interval relations. In [38, 40], we show that the path consistency algorithm finds the sets of feasible relations for the subclass of **PA** networks that do not contain the $\neq$ relation, and for a corresponding subclass of **SA** networks. But we also give examples there that show that, earlier claims to the contrary, the path consistency algorithm is not sufficient for finding the minimal network for general **PA** networks nor for general **SA** networks and we develop an $O(n^4)$ consistency algorithm that is sufficient, where n is the number of intervals or points.

An improved algorithm. Here we give an $O(\max(mn^2, n^3))$ time algorithm for finding all feasible relations, where n is the number of points and m is the number of pairs of points that are asserted to be not equal. The algorithm is of far more practical use than our previous algorithm (that algorithm is still of importance as an approximation algorithm for instances of the problem from the full interval algebra; see [38, 40] for the details).

Our strategy for developing an algorithm for **PA** networks is to first identify why path consistency is sufficient if we exclude $\neq$ from the language and is not sufficient if we include $\neq$. Fig. 7 gives the smallest counter-example showing that the path consistency algorithm does not correctly determine the minimal network representation of all **PA** networks. The network is path consistent. But it is easy to see that not every basic relation in the label between s and t is feasible. In particular, asserting $s = t$ forces v and w to also be equal to s and t. But this is inconsistent with $v \neq w$. Hence, the $=$ relation is not feasible as it is not capable of being part of a consistent scenario. The label between s and t should be $<$.

This is one counter-example of four vertices. But are there other counter-examples for $n \geq 4$? The following theorem answers this question and is the basis of an algorithm for finding all feasible relations for **PA** networks.

Theorem 2 (van Beek and Cohen [40]) *Any path consistent* **PA** *network which is not the minimal network, has a subgraph of four vertices isomorphic to the network in Fig. 7.*

The counter-example then is unique, up to isomorphism, if the network is path consistent. This leads to the following algorithm. We solve an instance of the feasible relations problem by first applying the path consistency algorithm and then systematically searching for "forbidden" subgraphs and appropriately

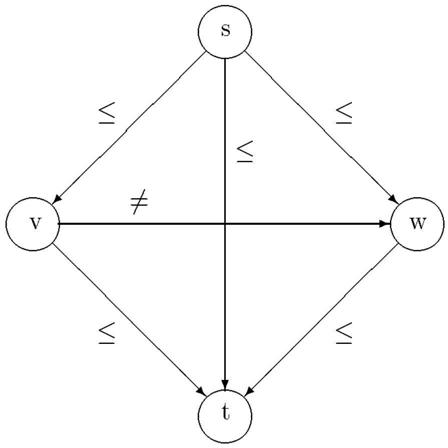


Figure 7: "Forbidden" subgraph

changing the labels. The algorithm is shown in Fig. 8. The input to the algorithm is a **PA** network represented as an adjacency matrix C where element C_{ij} is the label on edge (i, j). The algorithm also makes use of adjacency lists. For example, $\text{adj}_{\leq}(v)$ is the list of all vertices, w, for which there is an edge from v to w that is labeled with '$\leq$'.

Changing the label on some edge (s, t) from '$\leq$' to '$<$' may further constrain labels on other edges. The question immediately arises of whether we need to again apply the path consistency algorithm following our search for "forbidden" subgraphs to propagate the newly changed labels? Fortunately, the answer is no. Given a new label on an edge (s, t), if we were to apply the path consistency algorithm, the set of triples of vertices that would be examined is given by $\{(s, t, k), (k, s, t) \mid 1 \leq k \leq n, k \neq s, k \neq t\}$ (see RELATED-PATHS in Fig. 8). Thus there are two cases. For both, we can show that any changes that a second application of the path consistency algorithm would make will already have been made by procedure FIND-SUBGRAPHS.

Case 1: (s, t, k). Changing the label on the edge (s, t) from '$\leq$' to '$<$' would cause the path consistency algorithm to change the label on the edge (s, k) only in two cases:

$$s \leq t, t \leq k, \text{and } s \leq k$$

$$s \leq t, t = k, \text{and } s \leq k$$

In both, the label on (s, k) will become '$<$'. For (s, t) to change we must have the situation depicted in Fig. 7., for some v and w. But $v \leq t$ and $w \leq t$ together with $t \leq k$ (or $t = k$) imply that $v \leq k$ and $w \leq k$ (we can assume the relations were propagated because we applied the path consistency algorithm before the

```
FEASIBLE(C)
  PATH-CONSISTENCY(C)
  FIND-SUBGRAPHS(C)

PATH-CONSISTENCY(C)
  Q ←      ⋃      RELATED-PATHS(i, j)
      1 ≤ i < j ≤ n
  while (Q is not empty)
      do  select and delete a path (i, k, j) from Q
          t ← C_ij ∩ C_ik · C_kj
          if (t ≠ C_ij)
              then C_ij ← t
                   C_ji ← INVERSE(t)
                   Q ← Q ∪ RELATED-PATHS(i, j)

RELATED-PATHS(i, j)
  return { (i, j, k), (k, i, j) | 1 ≤ k ≤ n, k ≠ i, k ≠ j }

FIND-SUBGRAPHS(C)
  for each edge (v, w) such that w ∈ adj_≠(v)
      do  S ← (adj_≥(v) ∩ adj_≥(w))
          T ← (adj_≤(v) ∩ adj_≤(w))
          for each s ∈ S, t ∈ T
              do  C_st ← '<'
                  C_ts ← '>'
```

Figure 8: Feasible relations algorithm for **PA** networks

procedure for finding "forbidden" subgraphs). Hence, (s, k) also belongs to a "forbidden" subgraph and the label on that edge will have been found and updated.

Case 2: (k, s, t). Similar argument as Case 1.

Theorem 3 *Procedure* FEASIBLE *correctly finds the feasible relations between all pairs of points when applied to* **PA** *networks and requires* $O(\max(mn^2, n^3))$ *time, where* m *is the number of edges labeled with '*$\neq$*' and* n *is the number of points.*

Proof. Let P, M, and S be the propositions that "the network is path consistent", "the network is not the minimal network", and "the network contains a 'forbidden' subgraph", respectively. By Theorem 2 we have, $P \wedge \neg M \Rightarrow S$. Taking the contrapositive gives, $\neg S \Rightarrow \neg P \vee M$. But the algorithm removes all "forbidden" subgraphs, so $\neg S$ is true, and, by the case analysis above, the network remains path consistent, so $\neg P$ is false. Hence, the network is the minimal network. For the time bound, the path consistency procedure is $O(n^3)$, where n is the number of points [25]. The Find-Subgraphs procedure can be seen to be $O(mn^2)$, where m is the number of edges labeled with '$\neq$'. Hence the overall algorithm is $O(\max(mn^2, n^3))$. $\square$

We can find all pairs of feasible relations of an **SA** network by first translating it into a **PA** network, applying algorithm Feasible to the **PA** network, and translating the result back into an **SA** network. For many applications of **SA** networks in the literature, the translation into a **PA** network results in a network with few or no $\neq$ relations between points. Thus, a desirable feature of procedure Find-Subgraphs is that its cost is proportional to the number of edges labeled '$\neq$'.

To summarize, if our temporal networks are **PA** networks or **SA** networks we can find all pairs of feasible relations quickly using algorithm Feasible.

4 Interval Algebra Networks

In this section we examine the computational problems of finding consistent scenarios and finding the feasible relations of **IA** networks.

4.1 Finding a consistent scenario

Related work. Vilain and Kautz [41, 42] show that finding a consistent scenario is NP-Complete for **IA** networks. Thus the worst cases of the algorithms that we devise will be exponential and the best we can hope for is that the algorithms are still useful in practice. We discuss to what extent this is achieved below.

In the previous section we found a consistent scenario by first finding a consistent instantiation. An alternative method is as follows. Recall that a network S is a **consistent scenario** of a network C if and only if

(a) $S_{ij} \subseteq C_{ij}$,

(b) $|S_{ij}| = 1$, for all i, j, and

(c) there exists a consistent instantiation of S.

To find a consistent scenario we simply search through the different possible S's that satisfy conditions (a) and (b)—it is a simple matter to enumerate them—until we find one that also satisfies condition (c). Allen [2] was the first to propose using backtracking search to search through the potential S's. In this formulation of the problem the variables represent the relations between intervals, the domains of the variables are the set of basic interval relations, and the ternary constraints preclude certain combinations of relationships between three intervals. Note, however, that if the problem size is n in the original formulation, it is now n^2 in this alternative formulation.

There has been much work on improving the performance of backtracking that could be (or has been) adapted to this problem. This work can be classified according to the following four general considerations when designing a backtracking algorithm for a particular application [10]:

1. What kind of preprocessing to do (e.g. [19, 11]). For finding a consistent scenario, Reinefeld and Ladkin [33] give the results of extensive computational experiments characterizing how effective path consistency is in pruning as a preprocessing step before backtracking search.

2. Which variable to instantiate next (e.g. [30, 32]).

3. Which instantiation to give the variable (e.g. [19, 28]). For general constraint networks, Haralick and Elliott [19] propose a technique called forward checking where it is determined and recorded how the instantiation

of the current variable restricts the possible instantiations of future variables. This technique can be viewed as a hybrid of tree search and consistency algorithms (see [30]). For finding a consistent scenario, Reinefeld and Ladkin [33] give an algorithm that interleaves path consistency and backtracking search in the style of forward checking.

4. How to handle backtracking (e.g. [15, 9]). In chronological backtracking, when a dead end occurs in the search, the algorithm backs up to the last variable instantiated. For general constraint networks, Gaschnig [15] proposes backjumping as an improvement where the idea is to try and back up further to the source of the problem. For finding a consistent scenario, Valdés-Pérez [37] gives a backtracking algorithm in the style of backjumping.

Improving the algorithms. Here we show how the results for **SA** networks can be used to improve the performance of backtracking algorithms for finding a consistent scenario. We then design a backtracking algorithm modeled on that of Reinefeld and Ladkin [33] and present the results of experiments comparing the performance of the two algorithms on random problems drawn from two different distributions. Based on the experimental evidence, we postulate that the algorithm is useful in practice, in particular for planning problems and problems with similar characteristics.

Our proposal for improving the backtracking algorithms is the following. Rather than search directly for a consistent scenario of an **IA** network, as in previous work, we first search for something more general: a consistent **SA** subnetwork of the **IA** network. That is, we use backtrack search to find a subnetwork S of a network C such that,

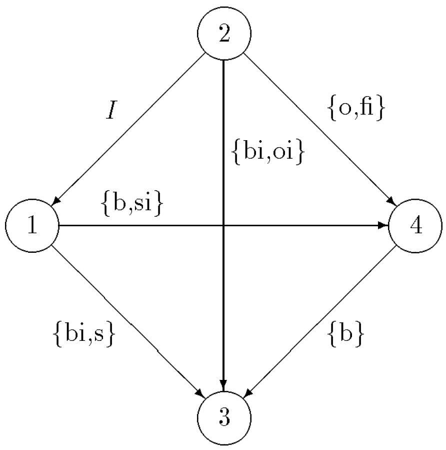
(a) $S_{ij} \subseteq C_{ij}$,

(b) S_{ij} is an allowed relation for **SA** networks, for all i, j, and

(c) there exists a consistent instantiation of S.

In previous work, the search is through the alternative singleton labelings of an edge, i.e., $|S_{ij}| = 1$. The key idea in our proposal is that we decompose the labels into the largest possible sets of basic relations that are allowed for **SA** networks and search through these decompositions. This can considerably reduce the size of the search space. An example will clarify this. Suppose the label on an edge is {b,bi,m,o,oi,si}. There are six possible ways to label the edge with a singleton label: {b}, {bi}, {m}, {o}, {oi}, {si}, but only two possible ways to label the edge if we decompose the labels into the largest possible sets of basic relations that are allowed for **SA** networks: {b,m,o} and {bi,oi,si}.

Example 7. Consider the network shown in Fig. 9. For illustration purposes only, suppose that we perform naive backtrack search (chronological backtracking and no forward checking) to find a consistent scenario and that the search



where I = {b,bi,m,mi,o,oi,s,si,d,di,f,fi,eq}

Figure 9: Example **IA** network

looks at the edges in the order (1,2), (1,3), (2,3), (1,4), (2,4), and (3,4). Fig. 11 shows a record of the search for both methods. Moving to the right and downward in the figure means a partial solution is being extended, moving to the left and downward means the search is backtracking. In this example, when searching through alternative singleton labelings, much search is done before it is discovered that no consistent scenario exists with edge (1,2) labeled with {eq}, but when decomposing the labels into the largest possible sets of basic relations that are allowed for **SA** networks and searching through the decompositions, no backtracking is necessary (in general, the search is, of course, not always backtrack free).

CSIAN(C)

1. Find a consistent **SA** subnetwork, S, of the **IA** network, C, using backtrack search.
2. Translate **SA** network, S, into **PA** network, P.
3. CSPAN(P)

Figure 10: Schema of a consistent scenario algorithm for **IA** networks

To test whether an instantiation of a variable is consistent with instantiations of past variables and with possible instantiations of future variables, we can either (i) translate the **SA** network into an equivalent **PA** network and use the $O(n^2)$ decision portion of procedure CSPAN (Steps 1–7 of Fig. 6), or (ii) use a

Single:

(1,2)	(1,3)	(2,3)	(1,4)	(2,4)	(3,4)
{eq}					
	{bi}				
		{bi}			
			{b}		
				{o}	
				{fi}	
			{si}		
				{o}	
				{fi}	
		{oi}			
	{s}				
		{bi}			
		{oi}			
{b}					
	{bi}				
		{bi}			
			{b}		
				{o}	
					{b}

SA:

(1,2)	(1,3)	(2,3)	(1,4)	(2,4)	(3,4)
{I}					
	{bi}				
		{bi,oi}			
			{b}		
				{o,fi}	
					{b}

Figure 11: Record of backtrack search using naive backtracking

path consistency algorithm. Finally, the result of the backtracking algorithm is a consistent **SA** subnetwork of the **IA** network (or a report that the **IA** network is inconsistent). After backtracking completes, the resulting **SA** network is translated into a **PA** network and then passed to algorithm CSPAN to find a consistent scenario of this network and, hence, a consistent scenario of the original **IA** network. A schema of the algorithm is shown in Fig. 10.

Experiments. We tested our ideas experimentally. For the purposes of the experiments we must make procedure CSIAN concrete by specifying a backtracking procedure. We chose to model our algorithm after that of Reinefeld and Ladkin [33] as the results of their experimentation suggests that it is very successful at finding consistent scenarios quickly. Following Reinefeld and Ladkin our algorithm has the following characteristics: path consistency preprocessing, randomly chosen and static order of instantiation of the variables, chronological backtracking, and forward checking or pruning using path consistency.

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 scenario by determining the basic relations which were satisfied by the instantiation. Finally, we then added indefiniteness to the relations between intervals by adding basic relations.

All experiments were performed on Sun 4/25's with 8 megabytes of memory. We implemented two versions of the algorithm that were identical except that one searched through the decompositions of the labels into the largest possible allowed relations for **SA** networks and the other searched through the decompositions of the labels into singleton labelings. The results are divided according to how the indefiniteness was randomly generated. Table 1 shows the results for random instances from a distribution designed to approximate planning applications (as estimated from a block-stacking in [4]; in planning, as formulated by Allen and Koomen [4], finding a consistent scenario corresponds to finding an ordering of the actions that will accomplish a goal). In this distribution, approximately 75% of the edges are labeled with I—meaning there is no constraint between the intervals, and the remaining edges have between 0 and 3 basic relations added as indefiniteness. For the results in Table 2, all subsets of I are equally likely to be added as indefiniteness to an edge. The timings do not include preprocessing time, as this was common to both methods, or the additional time for Steps 2&3 of CSIAN. However, this additional cost is small (for example, for $n = 100$, it takes about 0.15 seconds). For these two distributions, the experiments suggest that decomposing into "larger" relations improves the performance of backtracking search, sometimes by a factor of almost 3. The experiments also provide additional evidence for the efficacy of Reinefeld and Ladkin's algorithm.

To summarize, if our representation language is **IA** networks, we can use algorithm CSIAN, which is exponential in the worst case, to find a consistent scenario. One bright spot is that the algorithm seems to work well in practice for problems that arise in planning. The algorithm should work similarly well on any problem with the characteristics of a planning problem. The characteristics are: We do not have direct knowledge of the relations between most intervals and few of the relations are disallowed relations for **SA** networks. We remark that it is a simple matter to have a procedure that determines whether an **IA** network is also the special case of an **SA** network and then, depending on the outcome, calls either CSPAN or CSIAN to find a consistent scenario. This has the twofold advantage that the choice of algorithm can be hidden from the user and that no commitment need be made at the outset by the user to restrict the representation language (the more expensive CSIAN algorithm can simply be used as needed).

Table 1: Solving random instances of consistent **IA** networks from a distribution designed to model instances that arise in planning; 1000 tests for each n.

	SA		Single	
n	pc. calls	time (sec.)	pc. calls	time (sec.)
10	6.4	0.01	8.8	0.01
20	16.6	0.06	20.1	0.08
30	23.7	0.17	31.3	0.24
40	28.5	0.34	42.1	0.50
50	30.8	0.54	52.6	0.88
60	31.7	0.78	62.5	1.42
70	31.6	1.06	71.9	2.11
80	30.8	1.32	80.3	2.99
90	29.4	1.60	88.1	4.05
100	28.1	1.89	95.9	5.36

Table 2: Solving random instances of consistent **IA** networks from a distribution where all labels are added with equal likelihood; 1000 tests for each n.

	SA		Single	
n	pc. calls	time (sec.)	pc. calls	time (sec.)
10	13.3	0.02	9.9	0.02
20	30.3	0.21	96.2	0.45
30	84.8	0.88[a]	129.1	1.32[b]
40	67.9	0.85	41.6	0.49
50	27.5	0.49	52.9	0.82
60	22.9	0.57	50.6	0.96
70	20.5	0.70	54.9	1.38
80	17.8	0.82	57.7	1.87
90	15.3	0.93	59.9	2.44
100	13.8	1.07	61.5	3.08

[a] 1 test omitted as 10^5 limit on number of path consistency calls exceeded.
[b] 4 tests omitted as 10^5 limit on number of path consistency calls exceeded.

4.2 Finding the feasible relations

Vilain and Kautz [41, 42] show that finding the feasible relations is NP-Complete for **IA** networks. Freuder [14] and Seidel [34] give algorithms that find *all* consistent instantiations of a general constraint network. Hence, their algorithms can be used for finding the feasible relations. A difficulty is that both algorithms require the domains of the variables to be finite but **IA** networks have infinite domains. However, we can again reformulate the feasible relations problem as a network of ternary constraints with finite domains: the variables represent the relations between intervals, the domains of the variables are the set of basic interval relations, and the ternary constraints preclude certain combinations of relationships between three intervals. Seidel's algorithm [34] is useful for sparse constraint networks but here the networks are dense, there being a ternary constraint for every combination of three variables. For the problems of interest here, both algorithms appear to be practical only for small instances of the problem.

A backtracking algorithm similar to the one given in the previous section can be designed for finding all the feasible relations. Again, instead of searching through the alternative singleton labelings of the edges, we decompose the labels into the largest possible sets of basic relations allowed for **SA** networks and search through the decompositions. In the previous section when finding a consistent scenario we stopped the backtracking algorithm after one consistent **SA** network was found. To determine the feasible relations we must find all such consistent **SA** networks. For each such consistent **SA** network we find the feasible relations using the algorithm of Fig. 8. The feasible relations for the **IA** network are then just the union of all such solutions. Initial experience, however, suggests this method is practical only for small instances of the problem, or for instances where only a few of the relations between intervals fall outside of the allowed relations for **SA** networks. We conclude that in most cases a better approach is to, if possible, accept approximate solutions to the problem [2, 40].

5 Conclusions

Allen [2] and Vilain and Kautz [41] give frameworks for representing and reasoning about qualitative temporal information. We looked at two reasoning tasks that arise in applications of these frameworks: Given possibly indefinite and incomplete knowledge of the relationships between some intervals or points, (i) find a scenario that is consistent with the information provided, and (ii) find the feasible relations between all pairs of intervals or points.

For finding one consistent scenario, we give an $O(n^2)$ time algorithm for **PA** and **SA** networks. The algorithm is an $O(n)$ improvement over the previously known algorithm. The results for the point algebra are shown to aid in the design of a backtracking algorithm for **IA** networks. The backtracking algorithm is shown experimentally to be useful for planning problems. For finding the feasible relations, we give an algorithm for **PA** and **SA** networks that is of more practical use than the previously known algorithm.

The algorithms are of importance as the reasoning tasks arise in such diverse applications as natural language processing, plan recognition, planning, and diagnosis and within these applications the reasoning tasks often need to be solved repeatedly. As well, in related work we show how the algorithms can be used in answering a broader range of query types, including (i) determining whether a formula involving temporal relations between events is possibly true and necessarily true; and (ii) answering aggregation questions where the set of all events that satisfy a formula are retrieved [39].

A Appendix

In this appendix we prove Theorem 1 by showing statements (a) and (b) below. The rest of the algorithm is justified by the discussion in the text of Section 3.1.

a. The vertices v and w are necessarily equal if and only if there is a cycle of the form,

$$v \leq \cdots \leq w \leq \cdots \leq v, \tag{1}$$

where one or more of the $\leq$ can be $=$.

b. The network is inconsistent if and only if there is a cycle of the form,

$$v = \cdots = w \neq v, \tag{2}$$

or of the form,

$$v \leq \cdots \leq w \leq \cdots \leq v \neq w, \tag{3}$$

where some or all of the $\leq$ can be $=$, or of the form,

$$v < \cdots < w < \cdots < v, \tag{4}$$

where all but one of the $<$ can be $\leq$ or $=$.

Let M be the minimal network representation of a **PA** network. The idea behind the proof is that by Theorem 2 and Theorem 3, the path consistency algorithm correctly determines M_{ij} if M_{ij} is one of $\{\emptyset, <, =, >, \neq, ?\}$. That is, the path consistency algorithm correctly determines, in particular, whether a **PA** network is inconsistent (this was first proved in [23]) and whether two vertices are necessarily equal. Thus, we need to look at only the paths between vertices to prove statements (a) and (b).

To make the argument precise, we first give some notation. Let $P = (v, x_1), (x_1, x_2), \ldots, (x_m, w)$ be a path (possibly containing cycles) from vertex v to vertex w in a **PA** network. Let the **label of a path**, denoted $l(P)$, be the composition of the labels of the edges in the path, taken in order (see Fig. 12 for the composition table). For example, with reference to Fig. 4, let P be the path $(1,2), (2,3), (3,5), (5,4)$; the label of the path P is $< \cdot = \cdot < \cdot =$, which is simply $<$. Let $i(v, w)$ be defined as the intersection of the labels of all the paths from v to w. Montanari ([27, p. 113]; see also [24, p. 111], [1, p. 198]) shows that the path consistency algorithm computes $i(v, w)$, for each pair of vertices (v, w). Therefore, since the path consistency algorithm correctly determines whether a **PA** network is inconsistent and whether two vertices are necessarily equal, it is sufficient to look at the intersection of the labels of the paths between two vertices to prove statements (a) and (b).

Composition of two relations:

$\cdot$	$=$	$<$	$\leq$	$>$	$\geq$	$\neq$
$=$	$=$	$<$	$\leq$	$>$	$\geq$	$\neq$
$<$	$<$	$<$	$<$	?	?	?
$\leq$	$\leq$	$<$	$\leq$	?	?	?
$>$	$>$	?	?	$>$	$>$	?
$\geq$	$\geq$	?	?	$>$	$\geq$	?
$\neq$	$\neq$	?	?	?	?	?

Intersection of two relations:

$\cap$	$=$	$<$	$\leq$	$>$	$\geq$	$\neq$
$=$	$=$	$\emptyset$	$=$	$\emptyset$	$=$	$\emptyset$
$<$	$\emptyset$	$<$	$<$	$\emptyset$	$\emptyset$	$<$
$\leq$	$=$	$<$	$\leq$	$\emptyset$	$=$	$<$
$>$	$\emptyset$	$\emptyset$	$\emptyset$	$>$	$>$	$>$
$\geq$	$=$	$\emptyset$	$=$	$>$	$\geq$	$>$
$\neq$	$\emptyset$	$<$	$<$	$>$	$>$	$\neq$

Figure 12: Point algebra operations (Vilain and Kautz [41])

For vertices v and w to be necessarily equal (i.e., $i(v, w) =$ '$=$'), there must exist paths P_i and P_j from v to w such that the intersection of the labels of these paths is the $=$ relation. The following table gives all the possibilities.

	1	2	3	4
$l(P_i)$	$\leq$	$\geq$	$\leq$	$=$
$l(P_j)$	$=$	$=$	$\geq$	

By examination of the composition table for the point algebra (Fig. 12) it can be seen that these four cases arise only when there is a cycle of the form in Eqn. 1.

For the network to be inconsistent (i.e., $i(v, w) = \emptyset$), there must exist paths P_i, P_j, and P_k from v to w such that the intersection of the labels of these paths is the empty set. The following table gives all the possibilities.

	1	2	3	4	5	6	7
$l(P_i)$	$\leq$	$<$	$\leq$	$\geq$	$=$	$=$	$=$
$l(P_j)$	$\geq$	$>$	$>$	$<$	$<$	$>$	$\neq$
$l(P_k)$	$\neq$						

By examination of the composition table for the point algebra (Fig. 12) it can be seen that case (1) arises only when there is a cycle of the form in Eqn. 3, cases (2)-(6) arise only when there is a cycle of the form in Eqn. 4, and case (7) arises only when there is a cycle of the form in Eqn. 2. $\square$

References

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

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Comm. ACM*, 26:832–843, 1983.

[3] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[4] J. F. Allen and J. A. Koomen. Planning using a temporal world model. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 741–747, Karlsruhe, West Germany, 1983.

[5] M. J. Almeida. *Reasoning about the Temporal Structure of Narrative*. PhD thesis, State University of New York at Buffalo, 1987. Available as: Department of Computer Science Technical Report 87-10.

[6] B. C. Bruce. A model for temporal references and its application in a question answering program. *Artificial Intelligence*, 3:1–25, 1972.

[7] V. Chvátal. *Linear Programming*. W. H. Freeman, 1983.

[8] T. Dean and D. V. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[9] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[10] R. Dechter. Constraint networks. In *Encyclopedia of Artificial Intelligence, Second Edition*, pages 276–285. John Wiley & Sons, 1992.

[11] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 271–277, Detroit, Mich., 1989.

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

[13] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[14] E. C. Freuder. Synthesizing constraint expressions. *Comm. ACM*, 21:958–966, 1978.

[15] J. Gaschnig. Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisficing assignment problems. In *Proceedings of the Second Canadian Conference on Artificial Intelligence*, pages 268–277, Toronto, Ont., 1978.

[16] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

[17] S. Güther. Zur repräsentation temporaler beziehungen in SRL. Fachbereich Informatik, KIT Report 69, Technische Universität, Berlin, 1984. Cited in: A. Schmiedel, 1988.

[18] I. Hamlet and J. Hunter. A representation of time for medical expert systems. In *Proceedings of the European Conference on Artificial Intelligence in Medicine*, pages 112–119, Marseilles, 1987. Available as: Lecture Notes in Medical Informatics 33. Springer-Verlag.

[19] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[20] H. A. Kautz. *A Formal Theory of Plan Recognition*. PhD thesis, University of Rochester, 1987. Available as: Department of Computer Science Technical Report 215.

[21] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, 1973.

[22] P. B. Ladkin and R. Maddux. The algebra of constraint satisfaction problems and temporal reasoning. Technical report, Kestrel Institute, Palo Alto, Calif., 1988.

[23] P. B. Ladkin and R. Maddux. On binary constraint networks. Technical report, Kestrel Institute, Palo Alto, Calif., 1988.

[24] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[25] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[26] J. Malik and T. O. Binford. Reasoning in time and space. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 343–345, Karlsruhe, West Germany, 1983.

[27] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inform. Sci.*, 7:95–132, 1974.

[28] B. A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.

[29] K. Nökel. Temporal matching: Recognizing dynamic situations from discrete measurements. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1255–1260, Detroit, Mich., 1989.

[30] B. Nudel. Consistent-labeling problems and their algorithms: Expected-complexities and theory-based heuristics. *Artificial Intelligence*, 21:135–178, 1983.

[31] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[32] P. W. Purdom, Jr. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.

[33] A. Reinefeld and P. B. Ladkin. Fast solution of large interval constraint networks. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, Vancouver, B. C., 1992.

[34] R. Seidel. A new method for solving constraint satisfaction problems. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 338–342, Vancouver, B.C., 1981.

[35] F. Song and R. Cohen. The interpretation of temporal relations in narrative. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 745–750, Saint Paul, Minn., 1988.

[36] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[37] R. E. Valdés-Pérez. The satisfiability of temporal constraint networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 256–260, Seattle, Wash., 1987.

[38] P. van Beek. Approximation algorithms for temporal reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1291–1296, Detroit, Mich., 1989.

[39] P. van Beek. Temporal query processing with indefinite information. *Artificial Intelligence in Medicine, Special Issue on Temporal Reasoning*, 3:325–339, 1991.

[40] P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

[41] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 377–382, Philadelphia, Pa., 1986.

[42] M. Vilain, H. Kautz, and P. van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan-Kaufman, 1989.