Exact and Approximate Reasoning about Qualitative Temporal Relations

by

Peter van Beek

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 1990

Abstract

Much temporal information is qualitative information such as "The Cuban Missile crisis took place during Kennedy's presidency," where only the ordering of the end points of the two events is specified. A point and an interval algebra have been proposed for representing qualitative temporal information about the relationships between pairs of intervals and pairs of points, respectively. In this thesis, we address two fundamental reasoning tasks that arise in these algebras: Given (possibly indefinite) knowledge of the relationships between some intervals or points,

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

Solutions to these tasks have applications in natural language processing, planning, plan recognition, diagnosis, and knowledge-based systems.

For the task of finding consistent scenarios the main results are as follows. For the point algebra, we develop an O(n^2) time algorithm that is an O(n) improvement over the previously known algorithm, where n is the number of points. For the interval algebra, finding a consistent scenario has been shown to be (almost assuredly) intractable in the worst case. We show how the results for the point algebra aid in the design of a backtracking algorithm. The algorithm is shown to be useful in practice for planning problems.

For the task of finding the feasible relationships the main results are as follows. For the point algebra, we develop the first exact algorithm for finding all the feasible relationships that takes O(n^4) time, where n is the number of points, and show that a previously known algorithm is exact for a subset of the point algebra. For the interval algebra, finding the feasible relationships is again (almost assuredly) intractable. The intractability of finding exact solutions leads us to develop new algorithms that find approximate solutions. We examine the effectiveness of the approximation algorithms through computational experiments and determine under what conditions the algorithms are exact. We also give a simple test for predicting when the approximation algorithms will and will not produce good quality approximations.

Finally, we survey example applications chosen from the literature to show where the results of this thesis are useful.

Acknowledgements

This thesis was done under the supervision of Robin Cohen. To her I offer my warmest thanks for the many hours spent discussing and reading my work. I especially want to thank her for her encouragement and friendship. I also offer my thanks to the rest of my committee members: Charlie Colbourn, David Poole, Bill Pulleyblank, and my external examiner, Alan Mackworth. Charlie and David listened to me at various stages in the work and offered helpful advice for which I am grateful.

I also wish to thank Peter Ladkin, to whom I owe a large debt for many fruitful discussions about this and other related work, Marc Vilain for showing how to improve one of the proofs, Henry Kautz for helpful correspondence concerning work that he had done with Marc Vilain, and Rina Dechter for asking a question that led to the correction of an overly strong claim.

I also wish to thank friends made along the way for making the whole thing, for the most part, a very enjoyable experience: Bruce Spencer, Paul van Arragon, André Trudel, Fei Song, Scott Goodwin, Fahiem Bacchus, Chrysanne DiMarco, Qiang Yang, Kathy Broer van Arragon, Joanne Nonnekes, Bruce Cockburn, Dimpy Pathria, Janelle Harms, Vickie Martin, Faron Moller, Garrick Trowsdale, Harold Mantel, Rolf Edvinsson, Susan Kindersley, and Dita.

Finally, I wish to thank my parents Dick and Sharon Van Beek, my sister Susan Peters, brother-in-law Pete Peters, and niece Samantha Peters. To them I will ever be grateful for their love and support.

Peter van Beek
University of Waterloo
Summer 1990

1

Introduction

In this chapter we informally introduce the two problems that we address in this thesis and show why we want to solve these problems. We also summarize the contributions of the thesis.

1.1. The Problems

Representing and reasoning about temporal information is an essential part of such artificial intelligence tasks as natural language understanding, planning, plan recognition, and diagnosis. In these tasks, much of the temporal information that we want to represent is qualitative information. For example, in natural language we often make statements such as the following.

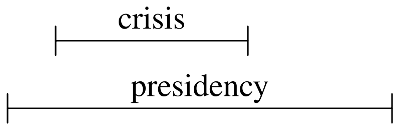
a. "The Cuban Missile crisis took place during Kennedy's presidency."

b. "Fred put the paper down and drank the last of his coffee."

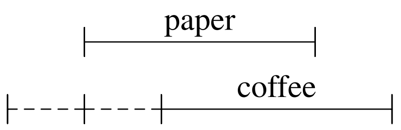
c. "Fred was reading the paper while eating his breakfast."

d. "I'll come by for a visit either before my class or after it."

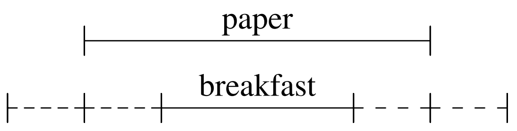
In these examples, no quantitative information such as date or duration information is specified. Statement (a) only specifies the qualitative information that the interval of time associated with one event occurred during the interval of time of another event; it specifies the ordering of all the end points of two intervals.



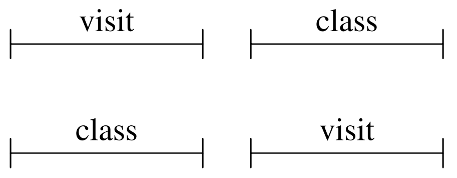
Statement (b) specifies the ordering of only some of the end points of two intervals and remains indefinite about others.



Statement (c) specifies only that two intervals of time intersect.



Statement (d) constrains, but remains indefinite about, the ordering of entire intervals.



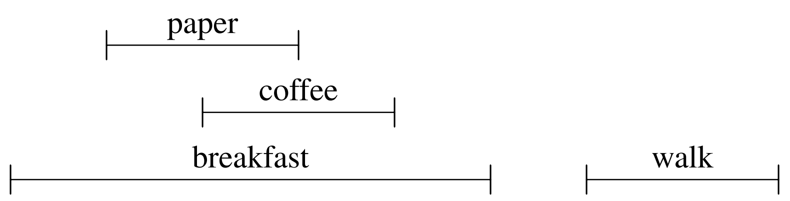
Allen (1983) has proposed an interval algebra and Vilain and Kautz (1986) have proposed a point algebra for representing such qualitative information. In this thesis, we address two fundamental reasoning tasks that arise in these algebras: Given (possibly indefinite) knowledge of the relationships between some intervals or points,

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

As an example of finding a consistent scenario, consider the following description of events:

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.

Not all of the temporal relationships between events are explicitly or unambiguously given in the description. The description does tell us, for example, that Fred finished reading his paper before he finished his coffee. However, it is ambiguous about the relationship between Fred starting to read his paper and starting his coffee or between Fred starting to read his paper and starting his breakfast. There are several possible scenarios that are consistent with the description of events. One such scenario is the following.



Another possible consistent scenario is one where Fred starts to read his paper before he starts his breakfast.

As an example of finding the feasible relationships, 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.

Again, not all of the temporal relationships between events are explicitly or unambiguously given. But 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. That is, the only feasible relationship between those two events is that "inform friend" occurred before "inform family". All other temporal relations, such as the two events occurring simultaneously, are infeasible. As this example illustrates, finding the feasible relationships can be viewed as computing the deductive consequences of our knowledge.

Specific applications of solutions to these tasks can be found in natural language processing (Allen, 1984; Almeida, 1987; Song and Cohen, 1988), planning (Allen and Koomen, 1983; Hogge, 1987), plan recognition (Kautz, 1987), diagnosis (Nökel, 1989), qualitative simulation (Weld and de Kleer, 1989), and knowledge-based systems (Koubarakis et al., 1989). In many other applications, what is desired is a general temporal reasoning system for storing, retrieving, and answering queries about both qualitative and quantitative information. The techniques developed here could be part of a special purpose reasoner in such a general system. The system would include other special purpose reasoners for quantitative information such as the distances between intervals or points (Dechter et al., 1989; Dean and McDermott, 1987), or combinations of qualitative and quantitative information (Allen and Kautz, 1985; Ladkin, 1989; Malik and Binford, 1983).

1.2. Contributions of the Thesis

We begin with a general discussion of the contributions and the approach taken followed by a detailed overview of the results in the thesis.

In general terms, the contributions of the thesis are: algorithms for finding a consistent scenario and for finding the feasible relationships, proofs of their correctness, analysis of their complexity, and evaluation of their behavior in practice. For the point algebra, we are able to give computationally efficient procedures for solving both tasks. Our algorithms are marked improvements over the previously known algorithms. For the full interval algebra, Vilain and Kautz (1986, 1989) show that both of these tasks are NP-Complete, 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:

- **Exact algorithms:** Solve the problem exactly but design algorithms that are efficient in practice even though their worst case is exponential.
- **Approximation algorithms:** Solve the problem approximately but design algorithms that are guaranteed polynomial and 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.

For finding a consistent scenario we develop an exponential algorithm and give evidence that the algorithm is efficient in practice for problems that arise in planning. For finding the feasible relationships we develop new approximation algorithms and give evidence that the algorithms produce good quality approximations for certain classes of problems. For both problems we identify "easy" special cases that are solvable in polynomial time.

Below we give a more detailed overview of our results. The discussion is divided according to the problem being solved.

For the task of finding a consistent scenario the main results are as follows. For the point algebra we develop an O(n^2) time algorithm that is an O(n) improvement over the previously known algorithm (Ladkin and Maddux, 1988b), where n is the number of points. For the interval algebra finding a consistent scenario has been shown to be (almost assuredly) intractable in the worst case. We show how the results for the point algebra aid in the design of a backtracking algorithm. The algorithm is shown analytically always to perform better than previous proposals and is shown experimentally to be useful in practice for planning problems and problems with similar characteristics.

For the task of finding the feasible relationships we consider two versions of the problem: an all-to-all version where we determine the feasible relationships between all pairs of intervals or points, and a one-to-all version where we determine the feasible relationships between one interval or point and every other interval or point.

For the all-to-all version of the task of finding the feasible relations the main results are as follows. For the point algebra, Vilain and Kautz (1986) claim that the well-known path consistency algorithm is exact for computing the feasible relationships between points. However, we present a counter-example to their theorem. We develop the first exact algorithm for finding all the feasible relationships that takes O(n^4) time, where n is the number of points, and show that the path consistency algorithm is exact for a subset of the point algebra.

For the interval algebra, finding the feasible relationships is again (almost assuredly) intractable. The intractability of finding exact solutions leads us to explore algorithms that find approximate solutions. Allen (1983) gives an O(n^3) approximation algorithm that turns out to be a variant of the path consistency algorithm. We develop a sequence of better (but more expensive) approximation algorithms. In particular, we develop an O(n^4) algorithm that computes a better approximation to the feasible relationships. 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 the approximate solution differed from the exact solution. 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. 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) show that a class of feasible relationships problems in the interval algebra can be phrased as feasible relationships problems in their point algebra. We show that our O(n^4) approximation algorithm is exact for the subset of the interval algebra that can be translated into the point algebra.

Finally, for the one-to-all version of the task of finding the feasible relations we show that a previously known algorithm (Edmonds and Karp, 1972) produces good quality approximations for certain classes of problems and takes O(n^2) time. We also show that the algorithm is exact for a useful subset of the interval algebra and of the point algebra.

1.3. Outline of the Thesis

An outline of the rest of the thesis is as follows. In Chapter 2 we give definitions and terminology used throughout the thesis. In Chapter 3 we examine algorithms for finding a consistent scenario, first giving an efficient algorithm for the point algebra and a subset of the interval algebra and then using those results to develop a backtracking algorithm for the interval algebra. In Chapter 4 we examine algorithms for the all-to-all version of the problem of finding the feasible relations. We give efficient algorithms for the point algebra. For the interval algebra we develop new approximation algorithms and identify easy (polynomial time) special cases where the algorithms are exact. In Chapter 5 we examine algorithms for the one-to-all version of the problem of finding the feasible relations. We give an efficient approximation algorithm and show that the algorithm is exact for a useful subset of the interval algebra and of the point algebra. In this chapter, we also give the results of some computational experiments designed to test the quality of the solutions produced by the all-to-all and one-to-all approximation algorithms. In Chapter 6, we survey selected applications of the interval algebra with the intent of showing where the results of this thesis will be of use. We also summarize the results of the thesis and try to give some general guidelines for applying the results. In Chapter 7 we discuss possible future work and give some concluding remarks.

2

Definitions and Terminology

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

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.

Definition 2.1. (Interval algebra, **IA**; Allen, 1983)
There are thirteen **basic** relations (including inverses) that can hold between two intervals (see Fig. 2.1). 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 set of all subsets of I, unary operator inverse, and binary operators intersection and composition (denoted "constraints" in Allen, 1983). Note that the relation I means there is no constraint between two intervals.

Definition 2.2. (Point algebra, **PA**; Vilain and Kautz, 1986)
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). 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 with additional operators complement and union and show that the algebras are relation algebras.

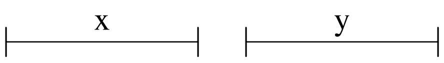
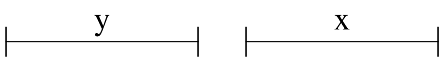
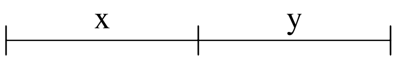
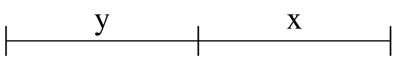
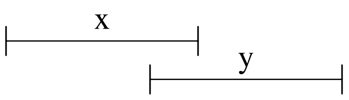
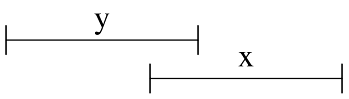
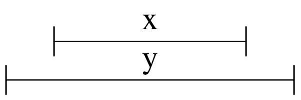
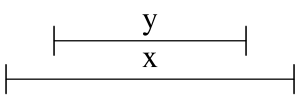
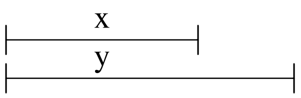
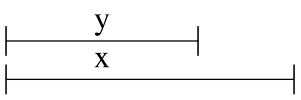
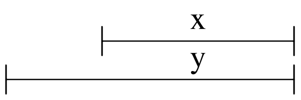
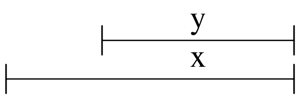
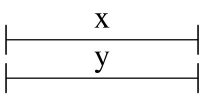
relation	symbol	meaning
x before y	b	
x after y	bi	
x meets y	m	
x met-by y	mi	
x overlaps y	o	
x overlapped-by y	oi	
x during y	d	
x contains y	di	
x starts y	s	
x started-by y	si	
x finishes y	f	
x finished-by y	fi	
x equal y	eq	

Fig. 2.1. Thirteen basic relations between intervals

2.2. Formalization of the Reasoning Tasks

We formalize our reasoning tasks using networks of binary relations (Montanari, 1974). Our development borrows from that found in Dechter et al. (1989) and Ladkin and Maddux (1988a, 1988b). This approach allows us to use some previously known algorithms and eases the development of new algorithms.

Definition 2.3. (Networks of binary relations; Montanari, 1974)
A **network of binary relations** 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} are disjunctions of the basic relations from one of the algebras. That is,

$$x_i R_{ij} x_j \;\equiv\; x_i R_1 x_j \;\vee\; \cdots \;\vee\; x_i R_m x_j$$

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.[1]

A **PA network** is a network of binary relations where 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.[2]

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

$$R_{ij}\colon\; X_i\, R_1\, X_j \vee \cdots \vee X_i\, R_m\, X_j$$

[1] See van Benthem, 1983 for some of the philosophical implications of modeling time by the real numbers.

[2] 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 Chapter 3.

A labeled graph is described by an $n \times n$ **adjacency matrix** C where entry C_{ij} is the label on edge (i, j). Note that $C_{ji} = C_{ij}^{-1}$.

An adjacency matrix B is a **consistent scenario** of an adjacency matrix C if

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

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

(c) there exists a consistent instantiation of B.

Given a consistent instantiation of a **PA** or **IA** network, the basic relations between variables satisfied by that consistent instantiation define a consistent scenario. 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.

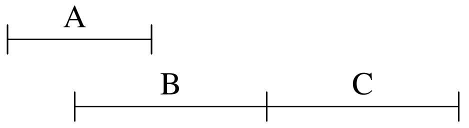
An element $R_k \in C_{ij}$ is **feasible** with respect to a network if and only if there exists a consistent instantiation of the network where R_k is satisfied. Given an **IA** network or a **PA** network, the **set of feasible relations** between two variables X_i and X_j in the network is the set consisting of *all and only* the $R_k \in C_{ij}$ that are feasible. More succinctly, the set of feasible relations is also called the **minimal label**.

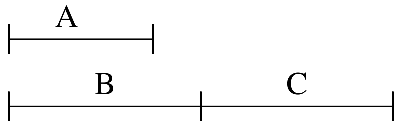
The reasoning tasks that we want to solve are finding a consistent scenario and finding the minimal labels.[3] Depending on the task being addressed, an algorithm is said to be **exact** for a class of input if it correctly finds a consistent scenario or correctly finds the minimal labels, for all instances 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,

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

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). As well, we do not always show edges that are labeled with I, the set of all basic relations. In this example, the relation between A and C is I, which shows that we have no explicit knowledge of the relation between A and C. There are two possible answers to the reasoning task of finding a consistent scenario. They are shown in the diagrams below. In the consistent scenario on the left, A overlaps B, in the one on the right, A starts B, and in both, B meets C and A is before C.





Thus, in every consistent instantiation of this network, "A before C" is satisfied and "B

[3] The terminology is borrowed from (Dechter et al., 1989) where it is used in a different context. Other terminology: a consistent scenario is a consistent singleton labeling (van Beek, 1989), a consistent instantiation is a satisfying assignment of values to the variables (Ladkin and Maddux, 1988b), and all pairs of feasible relationships is the deductive closure (Vilain et al., 1989) or, as it arises as a general constraint satisfaction problem, the minimal network (Montanari, 1974).

meets C" is satisfied. Further, there exists a consistent instantiation where "A overlaps B" is satisfied and one where "A starts B" is satisfied. An example of the latter is A $\leftarrow$ <1,2>, B $\leftarrow$ <1,3>, and C $\leftarrow$ <3,4>.[4]

For an example of our second reasoning task, we find the feasible relationships between every pair of intervals. The labels shown between A and B and between B and C are already the minimal labels. The one change is that the minimal label between A and C is just {b}, the "before" relation. We see that this is true in the consistent scenarios above. No other relation between A and C is feasible.

2.3. Additional Terminology

Mackworth (1977, 1987) defines three properties of networks that characterize local consistency of networks. A network is **node consistent** if and only 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 and only if, for every arc $(i,\ j)$,

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

In words, for every instantiation of X_i there exists an instantiation of X_j such that R_{ij} is satisfied. It is clear that **IA** and **PA** networks are also always arc consistent. A network is **path consistent** if and only if, for every triple $(i,\ k,\ j)$ of vertices,

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

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 can be used to show. Let A and C both be instantiated as <1,2>. No instantiation of B exists which satisfies the above formula where A, B, C are substituted for $x,\ y,\ z$, respectively.

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. Freuder (1982) defines **strongly k-consistent** 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.

[4] We are equating the names of the vertices with the variables they represent.

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, using only the relations in **PA**, into conjunctions of relations between the endpoints of the intervals (see Appendix A for an enumeration of **SA** and the translation into **PA**; also in Güther, 1984; 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

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

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

$$\begin{tikzcd} A^- \arrow[r, "\neq"] \arrow[d, "<"'] \arrow[dr, "<"] & B^- \arrow[d, "<"] \\ A^+ \arrow[ur, "\geq"] \arrow[r, "<"] & B^+ \end{tikzcd}$$

We define a new point algebra and new corresponding subset of the interval algebra that is of importance throughout the rest of the thesis. $\mathbf{PA_c}$ is the algebra with the same operators and underlying set as **PA** with the exception that $\neq$ 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 2.1 lists some of the properties of $\mathbf{PA_c}$ and $\mathbf{SA_c}$. Clause (i) verifies that $\mathbf{PA_c}$ and $\mathbf{SA_c}$ are indeed algebras. Clauses (ii)-(vi) prove useful later in the thesis.

Proposition 2.1. Let U be the underlying set of $\mathbf{SA_c}$ (respectively, $\mathbf{PA_c}$) and let I (respectively, ?) be the set of all basic relations of the algebra. The following properties of $\mathbf{SA_c}$ ($\mathbf{PA_c}$) are easily verified:

(i) U is closed under the operations inverse, intersection, and composition [i.e., $a^{-1} \in U$, $a \cap b \in U$, and $a \cdot b \in U$, for all a, b, $c \in U$].

(ii) Intersection is associative [i.e., $a \cap (b \cap c) = (a \cap b) \cap c$], intersection is commutative [i.e., $a \cap b = b \cap a$], intersection is idempotent [i.e., $a \cap a = a$], and I

(respectively, ?) is the identity for intersection [i.e., $a \cap I = I \cap a = a$].

(iii) Composition is associative, {eq} (respectively, {=}) is the identity for composition, and I (respectively, ?) is an annihilator for composition [i.e., $a \cdot I = I \cdot a = I$].

(iv) Composition distributes over intersection, provided the intersection does not give the empty set [i.e., $a \cdot (b \cap c) = a \cdot b \cap a \cdot c$ and $(b \cap c) \cdot a = b \cdot a \cap c \cdot a$, for all $a, b, c \in U$, provided $b \cap c \neq \varnothing$].

3

Finding a Consistent Scenario

In this chapter we examine the computational problem of finding consistent scenarios for the point algebra, **PA**, and for the interval algebra, **IA**. We first develop an efficient algorithm for **PA** networks. We then show how the results for the point algebra aid in the design of a backtracking algorithm for **IA** networks that is shown to be useful in practice. We also identify some easy special cases of **IA** networks.

3.1. The Point Algebra

3.1.1. Related Work

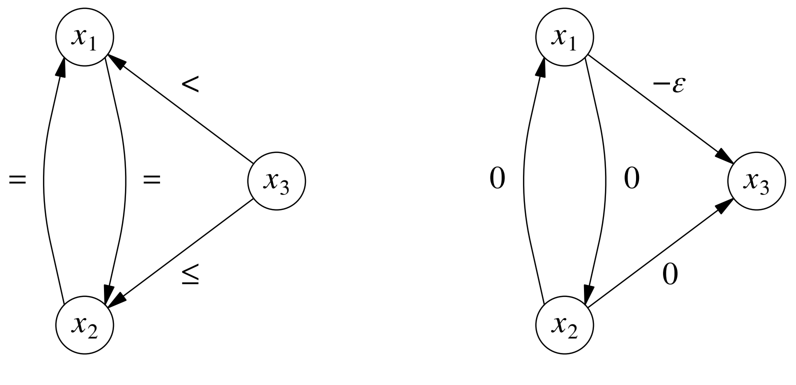
One method of finding a consistent scenario of a **PA** network is to first find a consistent instantiation of the network. The basic relations between variables satisfied by the consistent instantiation then give a consistent scenario.

Topological sort (see Knuth, 1973) 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 {<, ≤, =, >, ≥, ?}, 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, 1983). 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, 1982). 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,

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



PA network Shortest-path graph

Fig. 3.1. Shortest-path example

If all the l_{ij} are non-negative then we can use Dijkstra's (1959) algorithm to find a solution to this linear program.[5] Dijkstra's algorithm is $O(n^2)$. If some of the l_{ij} are negative then we can use the Floyd-Warshall algorithm to find a solution (see Aho et al., 1974). The Floyd-Warshall algorithm is $O(n^3)$. (See Deo and Pang, 1984 for an extensive bibliography of other shortest-path algorithms.)

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

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

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 value, ε, for turning a strict inequality into a weak inequality (see Chvátal, 1983, p. 451, for how to choose a value for ε such that solutions are preserved and no new solutions are introduced). A graphical representation can be constructed as follows. For each $x_i - x_j \le l_{ij}$ we draw a directed edge from vertex x_j to vertex x_i and label the edge with l_{ij}. Fig. 3.1 shows an example translation of a **PA** network into a shortest-path graph (the edges labeled with $+\infty$ are omitted).

[5] Dijkstra's algorithm can be viewed as the simplex algorithm greatly simplified because of the structure of the shortest path problem (Papadimitriou and Steiglitz, 1982).

In summary, if the allowed relations are restricted to {≤, =, ≥, ?}, then Dijkstra's algorithm can be used to find a consistent instantiation in O(n^2) time. If the allowed relations are restricted to {<, ≤, =, >, ≥, ?}, then the Floyd-Warshall algorithm can be used to find a consistent instantiation in O(n^3) time.

Finally, Ladkin and Maddux (1988b) 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 (Mackworth, 1977; Montanari, 1974) before finding a consistent scenario.

3.1.2. 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 (1988b, p.34) that topological sort alone will not work as the labels may be any one of the eight different **PA** elements, {∅, <, ≤, =, >, ≥, ≠, ?}, 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 ? (see Knuth, 1973). The "problem" labels are then {=, ∅, ≤, ≥, ≠}.

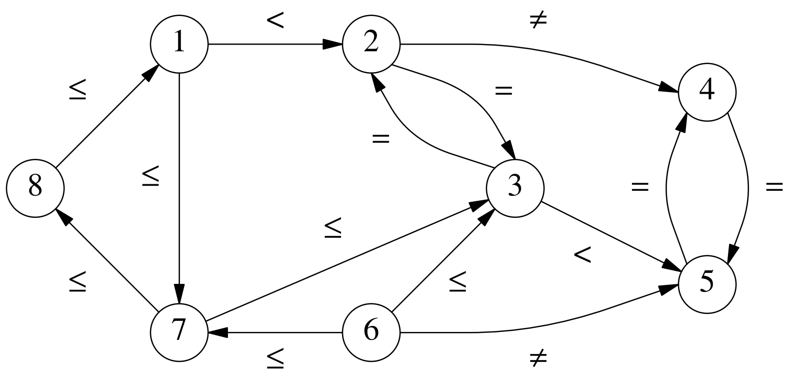
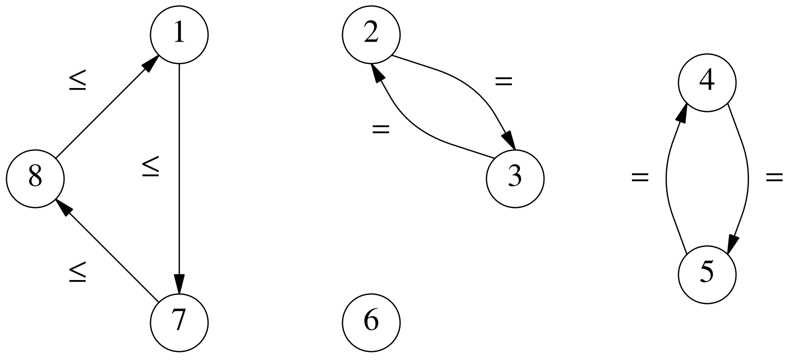


Fig. 3.2. Example PA network

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. Much of the discussion to follow relies on the assumption that looking at paths (the transitivity information) is sufficient for deciding the minimal label on an edge. The only exception to the truth of the assumption is that looking at paths sometimes assigns a label of ≤ instead of < (see Fig. 4.2 and discussion in Chapter 4) but we will note when this affects the discussion. The (somewhat large) network shown in Fig. 3.2 is used to illustrate the discussion. As usual, all edges (i, i) and all edges labeled ? are omitted.

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

$$v \le \cdots \le w \le \cdots \le v$$



$S_1 = \{1, 7, 8\}$ $S_3 = \{4, 5\}$
$S_2 = \{2, 3\}$ $S_4 = \{6\}$

Fig. 3.3. Strongly connected components

where one or more of the ≤ can be =. This is the same as saying v and w are in the same equivalence class if and only if there is a path from v to w and a path from w to v using only the edges labeled with ≤ 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 efficient algorithms are known (Tarjan, 1972).

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,

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

As an example, the strongly connected components of the network of Fig. 3.2 are shown in Fig. 3.3. There are four strongly connected components, S_1, S_2, S_3, and S_4. The condensed graph of the network of Fig. 3.2 is shown in Fig. 3.4. To illustrate, condensing the strongly connected component S_1 gives,

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

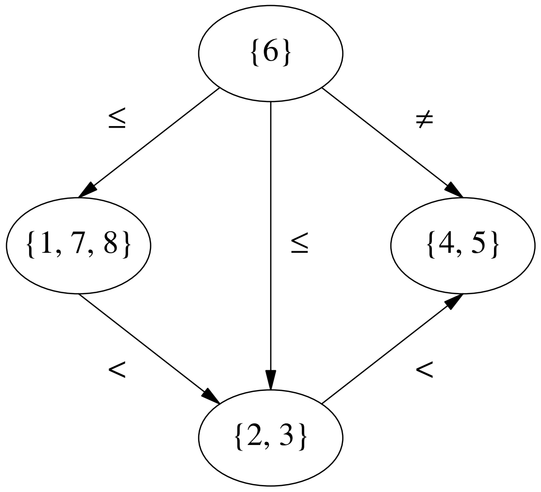


Fig. 3.4. Condensed PA network

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,

$$\begin{aligned}\hat{C}_{S_2S_3} &\leftarrow C_{24} \cap C_{25} \cap C_{34} \cap C_{35} \\ &\leftarrow \{<,>\} \cap \{<,=,>\} \cap \{<,=,>\} \cap \{<\} \\ &\leftarrow \{<\}\end{aligned}$$

The $\varnothing$ relation. To rule out the $\varnothing$ relation we must determine if the network is inconsistent. 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 ≤ can be =, or of the form,

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

where all but one of the < can be ≤ or =. It turns out that the first two cases are already detected when we identify all pairs of points that are forced to be 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. For example, suppose the label on the edge (1, 7) in the graph shown in Fig. 3.2 was < instead of the ≤ shown. Condensing the strongly connected component S_1 gives,

$$\hat{C}_{S_1S_1} \leftarrow C_{17} \cap C_{18} \cap C_{71} \cap C_{78} \cap C_{81} \cap C_{87}$$

$$\leftarrow \{<\} \cap \{>,=\} \cap \{>\} \cap \{<,=\} \cap \{<,=\} \cap \{>,=\}$$
$$\leftarrow \varnothing$$

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

The ≤ , ≥ relations. To remove the ≤ relation from the network, we simply change all ≤ labels to <. This is valid because, assuming that the ∅ 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 ≤ there exists a consistent scenario with < as the singleton label. But, changing a ≤ to a < can only force other labels to become <; it cannot force labels to become =. (Using the terminology of the algorithm in Fig. 3.5, 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.

Input: A **PA** network represented as a matrix C where entry C_{ij} is the label on edge (i, j).

Output: A consistent scenario of the network.

procedure CSPAN

Step 1. Identify the strongly connected components (SCCs) of the graph using only the edges labeled with <, ≤, and =.

Condense the graph: Let $\{S_1, S_2, \ldots, S_m\}$ be the SCCs found and let $\hat{C}$ be the adjacency matrix of the condensed graph. The vertices of the condensed graph are the S_i's. The labels on the edges between all pairs of vertices is given by,

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

If the empty label, ∅, results on any edge, then the network is inconsistent.

Step 2. Replace any remaining ≤ labels in $\hat{C}$ with < and perform a topological sort using only the edges in $\hat{C}$ labeled with <.

Fig. 3.5. Consistent scenario algorithm for PA networks

The ≠ 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 ≠ relations are now handled correctly (and implicitly) by topological sort. The output of topological sort is an assignment of numbers to the vertices (a mapping of the vertices to a time line) that is consistent with the information provided. As an example, consider the network shown in Fig. 3.4. 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. 3.2) is easily recovered from this information. The algorithm is summarized in Fig. 3.5.

Theorem 3.1. The algorithm in Fig. 3.5 correctly finds a consistent scenario of a **PA** network in $O(n^2)$ time, where n is the number of points.

***Proof*.** The proof of correctness is in Appendix C. For the time bound, finding the strongly connected components is $O(n^2)$ (Tarjan, 1972), condensing the graph looks at each edge only once, and topological sort is $O(n^2)$ (Knuth, 1973). □

It can be seen that the algorithm is asymptotically optimal in some sense, as we must at least examine every edge in the graph, of which there may be as many as $O(n^2)$. If we do not, we can not be sure that the label on that edge is not involved in a contradiction by, for example, being part of a loop that causes a vertex to be less than itself.

3.2. The Interval Algebra

3.2.1. Related Work

Vilain and Kautz (1986, 1989) 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. Let C be the adjacency matrix representation of an **IA** network. Recall that an adjacency matrix B is a consistent scenario of C if

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

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

(c) there exists a consistent instantiation of B.

To find a consistent scenario we simply search through the different possible B's that satisfy conditions (a) and (b)—it is a simple matter to enumerate them—until we find one that also satisfies condition (c). There are,

$$\mathop{\times}_{\substack{i = 1,\ldots,n \\ j = 1,\ldots,n}} | C_{ij} |$$

different possible B's. To reduce this search space, Allen (1983) proposes using simple backtracking search to enumerate the potential B's and an incremental version of the path consistency algorithm (see Section 4.1) to decide whether a partial solution found so far has a consistent instantiation. Valdés-Pérez (1987) gives a dependency-directed backtracking algorithm. As well, there has been much work on improving the performance of

```
procedure BACKTRACK ($i$)
begin
    $k \leftarrow 0$
    repeat
        $k \leftarrow k + 1$
        select $k$th candidate                                  (3.1)
        if (acceptable) then begin                              (3.2)
            record candidate
            if ($i < m$) then begin
                BACKTRACK ($i + 1$)
                if (not successful) then
                    cancel recording
            end
        end
    until (successful or no more candidates)
end
```

Fig. 3.6. Generic backtracking algorithm (Wirth, 1976)

backtracking that could be applied to this problem (see Haralick and Elliott, 1980; Nudel, 1983; Dechter and Pearl, 1988; Dechter and Meiri, 1989). Freuder (1978) and Seidel (1981) give algorithms that find all consistent instantiations (see Section 4.3.1).

Finally, it should be noted that Ladkin (1988) shows, by giving a translation, that the consistency and satisfiability of any arbitrary quantified formula involving the basic relations of the interval algebra can be decided by an algorithm that is exact for the restricted statements we can make in the interval algebra. Hence, algorithms that find consistent scenarios—or report that none exist—have applicability beyond **IA** networks.

3.2.2. An Improved Algorithm

Here we show how the results for the point algebra can be used to design an algorithm for finding a consistent scenario that is shown to be useful in practice.

The gist of the algorithm is the following. Rather than search directly for a consistent scenario of an **IA** network, as Allen and Valdés-Pérez do, we first search for something more general: a consistent **SA** subnetwork of the **IA** network. We then use algorithm CSPAN to find a consistent scenario of the **SA** subnetwork and, hence, to find a consistent scenario of the original **IA** network.

We use backtrack search to find a consistent **SA** subnetwork. Let C be the adjacency matrix representation of an **IA** network for which we wish to find a consistent scenario. We use backtrack search to find an adjacency matrix B such that,

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

(b) B_{ij} is in the underlying set of **SA**, for all i, j, and

(c) there exists a consistent instantiation of B.

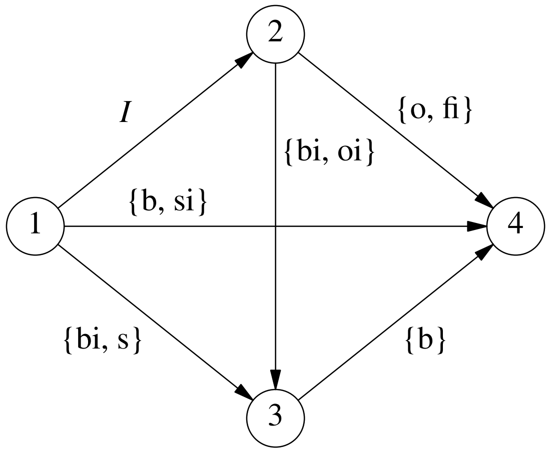
There are two important decisions to be made in designing a backtracking algorithm: (i) how to select a next candidate to extend a partial solution of size i to one of size $i+1$ (Eq. 3.1 of Fig. 3.6), and (ii) how to test whether a partial solution found so far is acceptable, i.e, whether it has a chance of being part of a solution to the entire problem (Eq. 3.2 of Fig. 3.6).

To select the next candidate, Allen and Valdés-Pérez choose the next alternative singleton labeling of an edge. i.e., they choose the next $| B_{ij} | = 1$. The key idea in our backtracking algorithm is that we choose the next candidate by decomposing the labels into the largest possible elements of **SA** and taking the next possible decomposition. This can considerably reduce the size of the domains we are searching through. 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 elements of **SA**: {b, m, o} and {bi, oi, si}. It is easy to see that this is guaranteed to be better since, for any choice of a singleton label, we can choose a label of larger (or equal) cardinality that is a superset of the singleton label. If the singleton label is consistent, so is the larger label. And, of course, there will be times when the larger label is consistent and the singleton label is not.

To test whether a partial solution found so far is acceptable and so might be part of a solution to the whole problem we translate the **SA** network into an equivalent **PA** network and use the $O(n^2)$ decision procedure for **PA** networks (Step 1 of Fig. 3.5). As we saw above, the benefits of the decision procedure go beyond a fast test for acceptability: next candidates can now be "larger" and thus more likely to be acceptable.

As an example, consider the network shown in Fig. 3.7. The backtrack search will look at the edges in the order (1,2), (1,3), (2,3), (1,4), (2,4), and (3,4). Fig. 3.8 shows a record of the search for both methods of selecting a next candidate. 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 elements of **SA** and searching through the decompositions, no backtracking is necessary (in general, the search is, of course, not always backtrack free).

The result of the backtracking algorithm is a consistent **SA** subnetwork of the **IA** network (or a report that the **IA** network is inconsistent). Finally, 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.

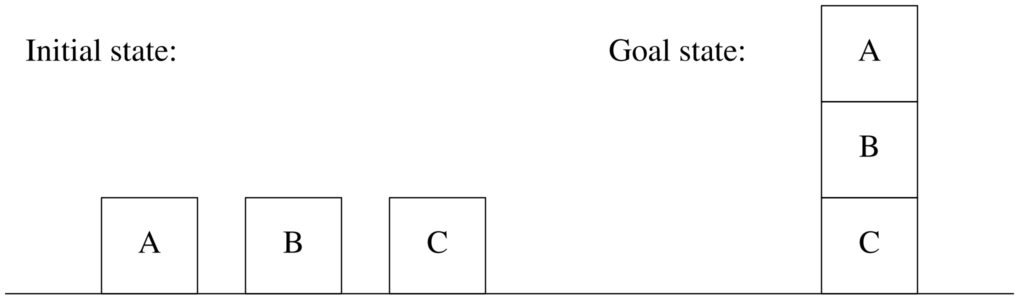


where I = {eq, b, bi, m, mi, o, oi, d, di, s, si, f, fi}

Fig. 3.7. Example IA network

Here is a somewhat larger example of planning in the blocks world (from Allen and Koomen, 1983). In general, a planning problem is specified by giving descriptions of the initial state and the goal state, and specifications of the actions the planning agent can perform, the effects of the actions, and the conditions under which the actions can be executed. A plan will be a sequence of actions that will take us from the initial state to the goal state. In planning, as formulated by Allen and Koomen (1983), actions and properties are associated with the intervals they hold over and the full interval algebra is used. Finding one consistent scenario corresponds to finding an ordering of the actions that will accomplish a goal.

Our specific planning problem is as follows. In the initial state, the three blocks are all on the table. The goal state is simply a tower of the blocks.



In this example, there is just one action called "Stack". The effect of the stack action is On(x, y): block x is on top of block y. For the action to be successfully executed, the

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}

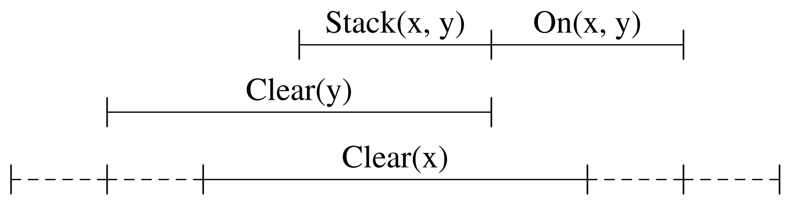
SA:

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

Fig. 3.8. Record of the backtracking search

conditions Clear(x) and Clear(y) must hold: neither block x or block y have a block on them. To complete the specification of the action, the relations between the effect, action, and conditions are as shown in the diagram below.

Stack:



We do not describe the planning process here, only the resulting temporal network. Given the initial state and goal state, we can immediately write down the following temporal information.

Initial Conditions			*Goal Conditions*		
Initial	{d}	Clear_1(A)	Goal	{d}	Clear_2(A)
Initial	{d}	Clear_1(B)	Goal	{d}	On(A,B)
Initial	{d}	Clear_1(C)	Goal	{d}	On(B,C)

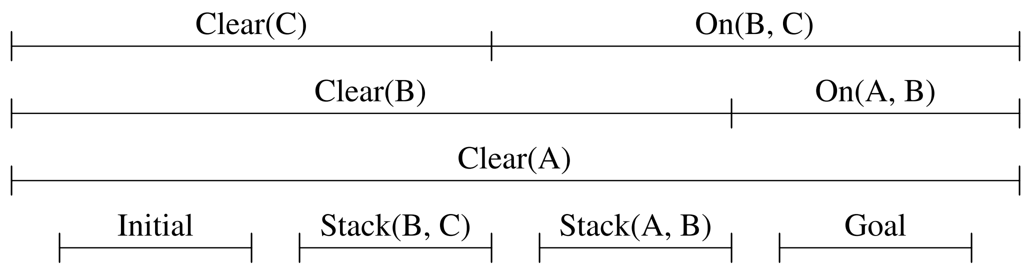
Planning introduces two stacking actions and the following temporal constraints.

Stacking Action			*Stacking Action*		
Stack(A,B)	{bi, mi}	Initial	Stack(B,C)	{bi, mi}	Initial
Stack(A,B)	{f}	Clear_2(B)	Stack(B,C)	{f}	Clear_2(C)
Stack(A,B)	{m}	On(A,B)	Stack(B,C)	{m}	On(B,C)
Stack(A,B)	{d}	Clear_3(A)	Stack(B,C)	{d}	Clear_3(B)

Proposition Constraints			*Domain Constraints*		
Clear_1(A)	{b,eq,bi}	Clear_2(A)	Clear_1(B)	{b,bi,m,mi}	On(A,B)
Clear_1(A)	{b,eq,bi}	Clear_3(A)	Clear_2(B)	{b,bi,m,mi}	On(A,B)
Clear_2(A)	{b,eq,bi}	Clear_3(A)	Clear_3(B)	{b,bi,m,mi}	On(A,B)
Clear_1(B)	{b,eq,bi}	Clear_2(B)	Clear_1(C)	{b,bi,m,mi}	On(B,C)
Clear_1(B)	{b,eq,bi}	Clear_3(B)	Clear_2(C)	{b,bi,m,mi}	On(B,C)
Clear_2(B)	{b,eq,bi}	Clear_3(B)			
Clear_1(C)	{b,eq,bi}	Clear_2(C)			

The subscripted propositions, such as Clear_1(A) and Clear_2(A), refer to different intervals. The domain constraints state that a block cannot at the same time be both clear and have another block on top of it. The proposition constraints state that the intervals of time over which the different Clear(x) propositions hold are either equal or have no intersection (see Allen and Koomen, 1983 for details). Note that there are far fewer than $O(n^2)$ explicit constraints, where n is the number of states, actions, and propositions in our network. Our method, because I is in **SA** and thus treated whole, only searches through decompositions of these 26 constraints.

One possible plan that will accomplish the goal of stacking the three blocks is the consistent scenario shown below.



A comparison of implementations of Allen's proposed backtracking algorithm (denoted Single) and our new algorithm (denoted CS_BackTrack) applied to the planning problem just described is shown in the following table.

Single	total	successful	fail
tries:	91	91	0
consistency checks:	314	91	223
total time (sec.):	3.1		
CS_BackTrack	total	successful	fail
tries:	91	91	0
consistency checks:	26	26	0
total time (sec.):	0.3		

Our algorithm was also tested on larger problems. The problems were random instances from a distribution designed to approximate planning applications (as estimated from the block-stacking example above). For a problem size of $n = 20$, the average time to find a solution was about seven seconds of CPU time (25 tests performed). For $n = 40$, it was 74 seconds (average over 21 tests). This seems surprisingly fast. However, it should be noted that four of the tests for $n = 40$ were not included as they were stopped before completion as a limit on the number of consistency checks was exceeded.

3.2.3. Easy Special Cases

The backtracking algorithm of the previous section finds a consistent scenario—or report that none exists—for general **IA** networks but the worst case time of the algorithm is exponential. In this section we discuss two easy (polynomial time) special cases of the general problem.

The first easy special case follows from the results for **PA** networks. 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. As an example, consider the (small) **SA** network below.

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

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

$$\begin{tikzcd}
A^- \arrow[r, "\neq"] \arrow[d, "<"'] \arrow[dr, "<"] & B^- \arrow[d, "<"] \\
A^+ \arrow[r, "<"] \arrow[ur, "\geq"'] & B^+
\end{tikzcd}$$

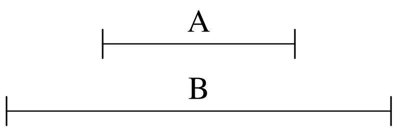
Equating the names of the vertices with the variables they represent, one consistent instantiation of this **PA** network is the assignment,

$$A^- \leftarrow 2 \qquad B^- \leftarrow 1$$
$$A^+ \leftarrow 3 \qquad B^+ \leftarrow 4$$

The corresponding consistent instantiation of the original **SA** network is simply,

$$A \leftarrow < 2, 3 > \qquad B \leftarrow < 1, 4 >$$

and the consistent scenario is given by,



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.

The second easy special case follows from results on a well-studied class of graphs called interval graphs (see Golumbic, 1980; Roberts, 1976).

Definition 3.1. (Interval graphs; Roberts, 1976)
A graph $G = (V, E)$ is an **interval graph** if and only if there is an assignment J of a real interval $J(u)$ to each $u \in V$ such that for all $u \neq v \in V$,

$$(u, v) \in E \quad \equiv \quad J(u) \cap J(v) \neq \emptyset$$

It can be seen that interval graphs are the special case of **IA** networks where all labels are either {eq,m,mi,o,oi,d,di,s,si,f,fi} (intersects) or {b,bi} (not intersects) and for which there exists a consistent instantiation. To be more precise, let C be the adjacency matrix representation of the **IA** network. The interval graph has an an edge $(u, v) \in E$ if C_{uv} = {eq,m,mi,o,oi,d,di,s,si,f,fi} and $(u, v) \notin E$ if C_{uv} = {b,bi}. Booth and Leuker (1976) give an algorithm that is linear in the number of vertices and edges for recognizing interval graphs. Roberts (1976) gives a polynomial time algorithm for finding an assignment J (a consistent instantiation).

As an aside, many applications of interval graphs have been identified (see Golumbic, 1980; Roberts, 1976). It can be argued, however, that **IA** networks are a better model for many of these applications. Interval graphs impose often overly strong restrictions on what can be stated. It cannot be stated, for example, that the relationship between two vertices is simply unknown. Further, if something stronger is known about the relationship, such as ‘‘A is before B’’, this too cannot be stated.

4

Finding the Feasible Relations: All-to-All Problems

In this chapter we examine the all-to-all version of the computational problem of finding the feasible relations for the point algebra, **PA**, and the interval algebra, **IA**. We first discuss the well-known path consistency algorithm. For the point algebra we show that, contrary to a previous claim, path consistency does not guarantee all relations are feasible and we develop an algorithm that is exact for **PA** networks. For the interval algebra we develop new approximation algorithms and identify some easy special cases of the problem.

4.1. The Path Consistency Algorithm

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 properties of **PA** and **IA** networks.

The path consistency algorithm works as follows (see Fig. 4.1; 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 Chapter 5 a discussion about defining an order over **IA**.

Input: A **PA** or **IA** network represented as 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
    $Q \leftarrow \{ (i, k, j) \mid 1 \le i < j \le n,\ 1 \le k \le n,\ k \ne i,\ k \ne j \}$

    while $Q$ is not empty do begin
        select and delete a path $(i, k, j)$ from $Q$          (4.1)
        $t \leftarrow C_{ij} \cap C_{ik} \cdot C_{kj}$          (4.2)
      if $(t \ne C_{ij})$ 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 $\{ (i, j, k), (k, i, j) \mid 1 \le k \le n,\ k \ne i,\ k \ne j \}$
```

Fig. 4.1. Path consistency algorithm (Mackworth, 1977)

Theorem 4.1 (Montanari, 1974; Mackworth and Freuder, 1985). The algorithm in Fig. 4.1 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. These operations have their usual first-order definitions: inverse: $xR^{-1}y \equiv yRx$, intersection: $x(R \cap S)y \equiv xRy \wedge xSy$, and composition: $xR \cdot Sz \equiv \exists y\ xRy \wedge ySz$. Below we discuss 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 **PA$_c$** the tables for the inverse of a label and the composition of two labels are easily specified explicitly as the underlying sets are small (see Appendix B). Similarly for **SA** and **SA$_c$**. For **IA**, however, the tables for inverse and composition are too large to be practical (2^{13} and $2^{13} \times 2^{13}$, respectively) but there are alternatives to explicitly specifying them. Hogge's (1987) method for inverse is to use a clever bit swapping technique and for composition is to use four smaller tables with an indexing scheme. Let $C_{ij} = \{R_1, \ldots, R_m\}$ be a label. Allen's (1983) method for inverse is to use the equivalence, $C_{ij}^{-1} \equiv \{R_1^{-1}, \ldots, R_m^{-1}\}$. This holds because inverse distributes over union. Allen's method for composition, which takes less space but more time than Hogge's, is to use the equivalence,

$$C_{ik} \cdot C_{kj} \equiv \bigcup (\{S\} \cdot \{T\}), \quad S \in C_{ik},\ T \in C_{kj} \tag{4.3}$$

which is the union of the pairwise composition of the basic relations. This holds because composition distributes over union. See Appendix B for the tables for the inverse and composition of the basic relations (denoted "transitivity table" in Allen, 1983).

Before giving an example, we first briefly relate the path consistency algorithm to other well-known algorithms for what has been called the algebraic path problem (Moller, 1985). In general terms, the algebraic path problem is to find the (infinite) set of all paths between each pair of vertices in a labeled graph. The graph is labeled with elements from a path algebra that has a multiplication and a join operator. Depending on the types of labels and the interpretation given to the operations on the labels of the paths, algorithms for this problem can, among other things, be used to find the shortest path in a graph, compute the transitive closure of a Boolean matrix, and solve systems of linear equations (see, e.g., Aho et al., 1974; Carré, 1979; Moller, 1985; Tarjan, 1981a, 1981b). It is generally assumed that for the multiplication and join operators of the path algebra, multiplication distributes over join. Mackworth's (1977) and Montanari's (1974) path consistency algorithms can be viewed as algorithms for path problems where multiplication does not distribute over join. For **IA** and **PA**, multiplication (composition) does not distribute over join (intersection). But as Proposition 2.1 shows, it is true for $\mathbf{SA_c}$ and $\mathbf{PA_c}$. Thus, for the special cases of $\mathbf{SA_c}$ and $\mathbf{PA_c}$ networks, we can take advantage of other efficient algorithms, such as that of Aho et al. (1974) or Tarjan (1981a, 1981b), for achieving path consistency.

As an example of the path consistency algorithm and of finding the feasible relations, consider the following description of events:

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.

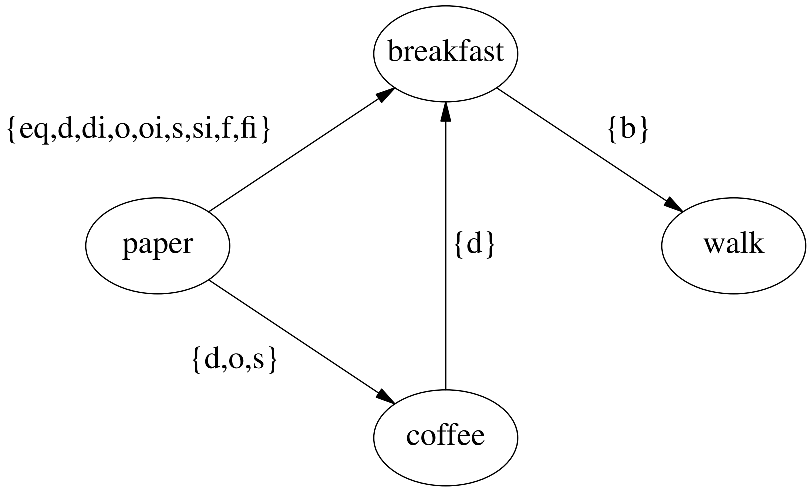
The first sentence tells us that the interval of time over which Fred read the paper intersects with the interval of time over which Fred ate breakfast. We represent this as "paper {eq,d,di,o,oi,s,si,f,fi} breakfast."

The second sentence fixes the relationship between the end points of the intervals over which Fred read his paper and over which Fred drank his coffee but it remains indefinite about the relationship between the start points of the two intervals. We represent this as "paper {d,o,s} coffee."[6] But we also know that drinking coffee is a part of

[6] Another possibility is the relation {b,m,d,o,s}, since the scenario where the coffee drinking

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." Putting this all together into an **IA** network gives,



where we have drawn the arc from "breakfast" to "walk" and so have labeled the edge with the inverse of the "bi" relation.

To illustrate the use of the path consistency algorithm for determining the feasible relations, we step through an iteration of the while loop. Suppose that the triple (paper, coffee, breakfast) is selected to be processed next (Eq. 4.1 of Fig. 4.1). Variable t is assigned,

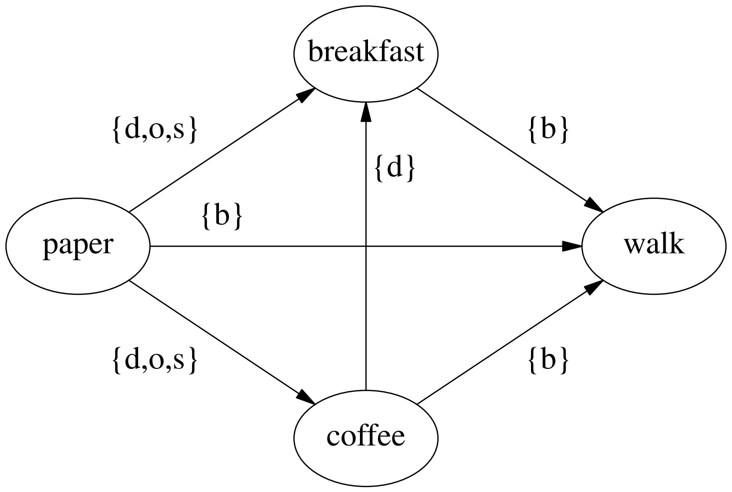
$$\leftarrow \quad C_{paper,breakfast} \cap C_{paper,coffee} \cdot C_{coffee,breakfast}$$
$$\leftarrow \quad \{eq,d,di,o,oi,s,si,f,fi\} \cap \{d,o,s\} \cdot \{d\}$$

Using Allen's method for composition and the composition table found in Appendix B, we get

$$\leftarrow \quad \{eq,d,di,o,oi,s,si,f,fi\} \cap (\{d\} \cdot \{d\} \cup \{o\} \cdot \{d\} \cup \{s\} \cdot \{d\})$$
$$\leftarrow \quad \{eq,d,di,o,oi,s,si,f,fi\} \cap (\{d\} \cup \{d,o,s\} \cup \{d\})$$
$$\leftarrow \quad \{d,o,s\}$$

Now, $t \neq C_{paper,breakfast}$ so the entries are updated and the related paths are added to the queue. The final result of the algorithm is shown below.

occurred entirely before reading the paper is not explicitly ruled out by the sentence.

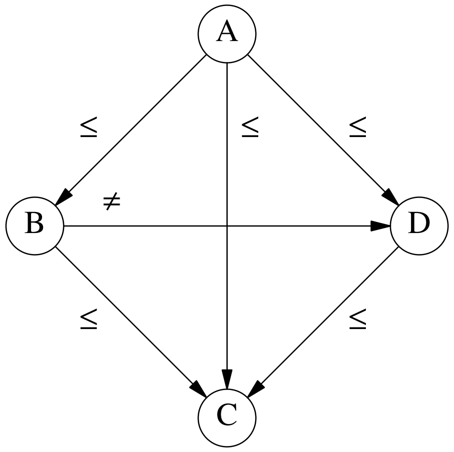


Determining the feasible relations can be viewed as determining the deductive consequences of our temporal knowledge. We are able to derive that Fred went for a walk after reading his paper and drinking his coffee and that Fred finished his paper before he finished his breakfast.

4.2. The Point Algebra

For the non-disjunctive subset of the point algebra, i.e., the relations {<, >, =}, algorithm CSPAN (Fig. 3.5) can be used to find the minimal labels. The minimal labels will simply be either the original network—corresponding to CSPAN successfully finding a consistent instantiation of the network—or the network with all labels being the empty set—corresponding to CSPAN reporting the temporal information is inconsistent.

For the full point algebra, Vilain and Kautz (1986, Theorem 4, p. 380) claim that the path consistency algorithm correctly finds the minimal labels for **PA** networks. However, their claim is false. The **PA** network below is a counter-example demonstrating that the path consistency algorithm does not correctly find the minimal labels for all **PA** networks.



Applying algorithm PC of Fig. 4.1 results in no changes; the network is already path consistent. However, the relation ≤ between A and C is not the minimal label as not every basic relation in the label is feasible. In particular, asserting A = C forces B and D to also be equal to A and C. But this is inconsistent with B ≠ D. Hence, the = relation is not feasible as there cannot exist a consistent instantiation of the network in which the = relation between A and C is satisfied. The label between A and C should be <. To reiterate, the counter-example shows that the path consistency algorithm is not exact for **PA** networks.

We next prove that the path consistency algorithm is exact for $\mathbf{PA_c}$ networks, 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 (intuitively, a set is convex if we can draw a line between every pair of points in the set and all the points along that line are also in the set).

Theorem 4.2 (Helly's theorem). 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 (ref. Chvátal, 1983).

Theorem 4.3. 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 ≠ relation.

***Proof*.** Theorem 4.3 is proved by showing that if all labels are from $\mathbf{PA_c}$ and the network is path consistent, then the network is k-consistent for all $k \leq n$. 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 $\mathbf{PA_c}$ 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 real numbers. The inductive assumption implies that variables $X_1, \ldots, X_{k-1}$ can be consistently instantiated. Let $x_1, \ldots, x_{k-1}$ be an instantiation such that

$$x_i \;\; R_{ij} \;\; x_j \qquad\qquad i, j = 1, \ldots, k-1$$

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

$$x_i \;\; R_{ik} \;\; X_k \qquad\qquad i = 1, \ldots, k-1 \tag{4.4}$$

is satisfied. We do so as follows. The $x_1, \ldots, x_{k-1}$ restrict the allowed instantiations of X_k. For each i in Eq. (4.4) we get bounds on instantiations of X_k. The key is that all these bounds define convex sets so by Helly's theorem it is sufficient to show that any two bounds have a point in common to show that they all have a point in common. But this follows directly from the definition of path consistency. Hence, all the bounds have a point in common and there exists at least one instantiation of X_k that satisfies Eq. (4.4) 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 there exists at least one instantiation of variable X_k such that

$$X_k \ \ R_{ki} \ \ x_i \qquad i = 1, \ldots, k-1$$

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

$$x_i \ \ R_{ij} \ \ x_j \qquad i, j = 1, \ldots, k$$

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

Thus, any algorithm that achieves path consistency is exact for $\mathbf{PA_c}$ networks (see Section 4.1 for a discussion of algorithms for ensuring path consistency). In particular, Aho et al. (1974) give an algorithm that achieves path consistency if certain properties hold. Proposition 2.1 shows the relevant properties and shows that $\mathbf{PA_c}$ has these properties with one provision. Composition does distribute over intersection *provided* the intersection of two labels is not equal to the empty set. Clearly, this is not always true. Does this restrict the applicability of the algorithm? Fortunately it does not. In the algorithm, if the intersection of two labels is equal to the empty set, this means the network is inconsistent, and the algorithm can halt and report the inconsistency.

In the remainder of this section we develop an algorithm that is exact for **PA** networks. 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 $\neq$.

In the proof of Theorem 4.3 for the exactness of path consistency for $\mathbf{PA_c}$ 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, and we can no longer show that a path consistent network is also k-consistent for all $k \leq n$. In fact, it is possible to have **PA** networks that are three-consistent but not four-consistent, i.e., networks where the intersection of any two bounds cannot be empty, but the intersection of three can be empty. Such is the case in the **PA** network shown in Fig. 4.2.

The network in Fig. 4.2 is path consistent. Let X_s, X_t, X_v, and X_w be the variables represented by the vertices s, t, v, and w, respectively. Further, let X_s, X_v, and X_t be instantiated as x_s, x_v, and x_t such that $x_s = x_v = x_t$. The instantiation is consistent. The bounds on the instantiation of X_w are $x_s \leq X_w$, $x_v \neq X_w$, and $x_t \geq X_w$. Using standard interval notation and substitution of equals the three bounds are $[x_s, +\infty)$, $(-\infty, x_s) \cup (x_s, +\infty)$, and $(-\infty, x_s]$. 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. While it is not necessary that a network be strongly n-consistent for the labels to be the minimal labels, the network in Fig. 4.2 is also an example of a path consistent network for which not all the labels are minimal.

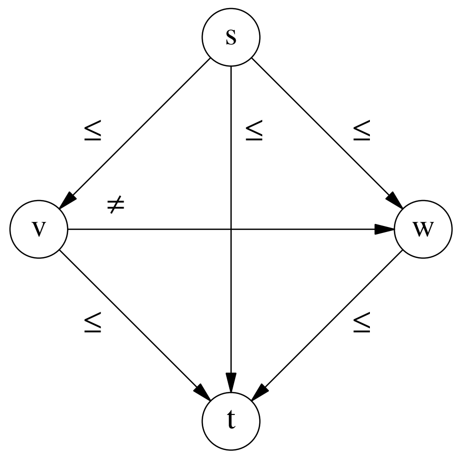


Fig. 4.2. "Forbidden" subgraph

Fig. 4.2 shows one counter-example of four vertices to the exactness of path consistency for **PA** networks. But are there other counter-examples of size $n \geq 4$? The following theorem answers this question and is the basis of an algorithm for finding the feasible relations for **PA** networks.

Theorem 4.4. 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. 4.2.

***Proof*.** The networks under consideration are k-consistent for $k = 1$, 2, and 3. If the labels of a network are not the minimal labels, then 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.

Thus, we know that (i) the intersection of all the bounds is empty, (ii) every two bounds have a point in common, by the assumption of path consistency, and (iii) at least one of the bounds must involve a $\neq$, 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 $\neq$ 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 $[c, +\infty)$, $(-\infty, c) \cup (c, +\infty)$, and $(-\infty, c]$. And it is easily shown by enumeration that the network in Fig. 4.2 is the only four-vertex path-consistent network that, up to isomorphism, gives rise to these bounds. $\square$

The counter-example then is unique, up to isomorphism, if the network is path consistent. This leads to the following algorithm. We solve the feasible relations problem by first applying the path consistency algorithm and then systematically searching for "forbidden" subgraphs and appropriately changing the labels. The algorithm (see Fig. 4.3) makes use of adjacency lists. For example, $\mathrm{adj}_{\le}(v)$ is the list of all vertices, w, for which there is an edge from v to w that is labeled with '$\le$'.

Changing the label on some edge (s, t) from '$\le$' 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 possible triangles that would be examined is given by $\{ (s, t, k),\ (k, s, t) \mid 1 \le k \le n,\ k \ne s,\ k \ne t \}$ (see RELATED_PATHS in Fig. 4.1). 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 '$\le$' to '$<$' would cause the path consistency algorithm to change the label on the edge (s, k) only in two cases:

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

In both, the label on (s, k) will become '$<$'. For (s, t) to change we must have the situation depicted in Fig. 4.2., for some v and w. But $v \le t$ and $w \le t$ together with $t \le k$ (or $t = k$) imply that $v \le k$ and $w \le k$ (we can assume the relations were propagated because we applied the path consistency algorithm before the 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 4.5. The algorithm in Fig. 4.3 correctly finds the minimal labels 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 '$\ne$' and n is the number of points.

***Proof*.** Let P, M, and S be the propositions that "the network is path consistent", "the labels between the vertices in the network are the minimal labels", and "the network contains a 'forbidden' subgraph", respectively. By Theorem 4.4 we have, $P \wedge \neg M \supset S$. Taking the contrapositive gives, $\neg S \supset \neg P \vee M$. 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 labels are the minimal labels.

For the time bound, the path consistency procedure is $O(n^3)$, where n is the number of points (Theorem 4.1). The FIND_SUBGRAPHS procedure can be seen to be $O(mn^2)$, where m is the number of edges labeled with '$\ne$'. Hence the overall algorithm is $O(\max(mn^2, n^3))$. □

A desirable feature of procedure FIND_SUBGRAPHS is that its cost is proportional to the number of edges labeled '$\neq$'. But in the worst case, there can be as many as O(n^2) edges labeled with '$\neq$' and thus algorithm FEASIBLE is also O(n^4). However, this worst case analysis is misleading as the worst cases for the algorithm are contrived and (presumably) would rarely occur. As preliminary experimental evidence, the algorithm was implemented in a straightforward way and tested on random problems up to size 100 (see section 5.3.1 for a description of the method for generating random problems; the description there is for **IA** networks but a similar method was employed for **PA** networks). It was found that about 90% of the time was spent in the path consistency algorithm and only about 2% in FIND_SUBGRAPHS. Hence, the O(n^3) path consistency procedure dominated the computation.

Input: A **PA** network represented as a matrix C where entry C_{ij} is the label on edge (i, j).

Output: The minimal labels for C_{ij}, $i, j = 1, \ldots, n$.

```
procedure FEASIBLE
begin
    PC
    FIND_SUBGRAPHS
end

procedure FIND_SUBGRAPHS
begin
    for each edge $(v, w)$ such that $w \in \mathrm{adj}_{\neq}(v)$ do begin
        $S \leftarrow (\mathrm{adj}_{\geq}(v) \cap \mathrm{adj}_{\geq}(w))$
        $T \leftarrow (\mathrm{adj}_{\leq}(v) \cap \mathrm{adj}_{\leq}(w))$
        for each $s \in S$ do
            for each $t \in T$ do begin
                $C_{st} \leftarrow$ '<'
                $C_{ts} \leftarrow$ '>'
            end
    end
end
```

Fig. 4.3. Feasible relations algorithm for PA networks

4.3. The Interval Algebra

In this section we discuss algorithms for finding the feasible relations for **IA** networks. Vilain and Kautz (1986, 1989) show that finding the feasible relations is NP-Complete for **IA** networks. Thus the worst cases of the algorithms that we devise will be exponential in the worst case. Unfortunately, no algorithm was found that works well in practice. We begin by reviewing two previous algorithms.

4.3.1. Exact Algorithms

Freuder (1978) and Seidel (1981) give algorithms for finding *all* consistent instantiations of a 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, Tsang (1987) points out that an alternative formulation of the feasible relations problem is 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. Note, however, that if the problem size is n in our formulation, it is now n^2 in this alternative formulation. Seidel's algorithm (1981) 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.

An exact algorithm for finding the feasible relations between just one pair of intervals is given in Chapter 5. That algorithm can thus, of course, also be used to find all pairs of minimal labels. Briefly, the algorithm tests whether a basic relation on an edge is feasible by using the backtracking algorithm CS_BackTrack (Section 3.2.2) to find a consistent instantiation or report that none exists. Initial experience, however, suggests this method is also practical only for small instances of the problem, or for instances where only a few of the relations between intervals fall outside of the special subset **SA**. In practice, if a consistent instantiation exists, algorithm CS_BackTrack often finds it quickly. The difficulty occurs when a basic relation is not feasible, i.e., no consistent instantiation exists. It often takes a long time to discover this. We conclude that in most cases a better approach than finding exact solutions is to, if possible, accept approximate solutions to the problem.

4.3.2. Approximation Algorithms

Allen (1983) shows that an algorithm that achieves path consistency can be used to find approximations to the minimal labels for **IA** networks (see Section 4.1). In this section we develop better but more expensive algorithms for determining approximations to the minimal labels between all intervals. The labels computed by the algorithm, as with the path consistency algorithm, are always a superset (not necessarily proper) of the minimal labels. The algorithms compute better approximations in that there are fewer infeasible relations in the computed labels. The results of some experiments designed to estimate the quality of the approximate solutions are summarized and discussed in Chapter 5.

Where does the path consistency algorithm fail? Determining this helps to 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 $R \in C_{ij}$ 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. Fig. 4.4 gives an example from Allen (1983) ascribed to Kautz. Applying the algorithm of Fig. 4.1 results in no changes to the labels; the network is path consistent. However, the network is inconsistent. The minimal labels are all $\varnothing$.

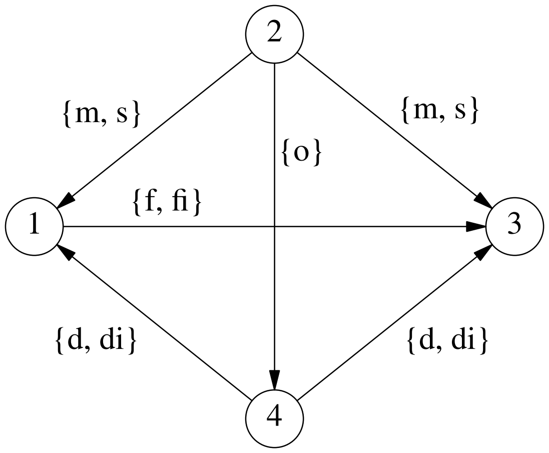


Fig. 4.4. Kautz's example

As mentioned, the path consistency algorithm, as an approximation, ensures that the $R \in C_{ij}$ are locally feasible—feasible not necessarily with respect to the entire graph, but with respect to every possible subgraph of three vertices (or 3-cliques in the graph). Note also that the path consistency algorithm only asks for the composition of labels on edges that share a vertex.

The simple idea for improving the approximation is to ensure that every $R \in C_{ij}$ 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. Eq. (4.2a) in Fig. 4.5 and the definition of composition over triangles (Eq. 4.3a) below ensure that C_{ij} gets updated to be the set of the $R \in C_{ij}$ that are feasible with respect to the subgraph of four vertices (i, k, l, j):

$$\Delta_{ikl} \cdot \Delta_{klj} \equiv \bigcup (\{P\} \cdot \{Q\} \cap \{S\} \cdot \{T\}), \qquad (4.3a)$$
$$O \in C_{kl},$$
$$S \in C_{ik},\ P \in (\{S\} \cdot \{O\} \cap C_{il}),$$
$$Q \in C_{lj},\ T \in (\{O\} \cdot \{Q\} \cap C_{kj})$$

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
begin
  $Q \leftarrow \{\, (i, k, l, j) \mid 1 \le i < j \le n,\ 1 \le k < l \le n,\ i, j, k, l$ distinct$\}$        (*)
  while $Q$ is not empty do begin
     select and delete a 4 – tuple $(i,\ k,\ l,\ j)$ from $Q$        (4.1a) (*)
     $t \leftarrow C_{ij} \cap \Delta_{ikl} \cdot \Delta_{klj}$        (4.2a) (*)
    if $(t \ne C_{ij})$ 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 $\{\, (k,\ i,\ j,\ l) \mid 1 \le k < l \le n,\ k, l, i, j$ distinct$\} \cup$        (*)
     $\{\, (i,\ j,\ l,\ k), (k,\ l,\ i,\ j) \mid 1 \le k, l \le n,\ k, l, i, j$ distinct$\}$        (*)
```

Fig. 4.5. New all-to-all consistency algorithm. Changes to the path consistency algorithm are marked (*).

The algorithm (shown in Fig. 4.5) 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.

To illustrate algorithm AAC, we apply it to the network shown in Fig. 4.4 and step through one iteration of the while loop. Suppose that the 4-tuple (1, 2, 4, 3) is selected to be processed next (Eq. 4.1a of Fig. 4.5). Variable t is assigned,

$$\leftarrow \quad C_{13} \cap \Delta_{124} \cdot \Delta_{243}$$

Using Eq. (4.3a) and the composition table found in Appendix B, we get

$$\leftarrow \quad \{\text{f, fi}\} \cap \Big(\bigcup_{O \in C_{24},} (\{P\} \cdot \{Q\} \cap \{S\} \cdot \{T\}),$$

$$
\begin{array}{ll}
 & S \in C_{12},\ P \in (\{S\} \cdot \{O\} \cap C_{14}), \\
 & Q \in C_{43},\ T \in (\{O\} \cdot \{Q\} \cap C_{23}))
\end{array}
$$

$$
\begin{array}{lll}
\leftarrow & \{\mathrm{f}, \mathrm{fi}\} \cap (\bigcup & (\{S\} \cdot \{T\} \cap \{P\} \cdot \{Q\}), \\
 & & O \in \{\mathrm{o}\}, \\
 & & S \in \{\mathrm{mi}, \mathrm{si}\},\ P \in (\{S\} \cdot \{O\} \cap \{\mathrm{d}, \mathrm{di}\}), \\
 & & Q \in \{\mathrm{d}, \mathrm{di}\},\ T \in (\{O\} \cdot \{Q\} \cap \{\mathrm{m}, \mathrm{s}\}))
\end{array}
$$

$$
\begin{array}{lllll}
\leftarrow & \{\mathrm{f}, \mathrm{fi}\} \cap (& (\{\mathrm{mi}\} \cdot \{\mathrm{s}\} & \cap\ \{\mathrm{d}\} \cdot \{\mathrm{d}\}) & \cup \\
 & & (\{\mathrm{mi}\} \cdot \{\mathrm{m}\} & \cap\ \{\mathrm{d}\} \cdot \{\mathrm{di}\}) & \cup \\
 & & (\{\mathrm{si}\} \cdot \{\mathrm{s}\} & \cap\ \{\mathrm{di}\} \cdot \{\mathrm{d}\}) & \cup \\
 & & (\{\mathrm{si}\} \cdot \{\mathrm{m}\} & \cap\ \{\mathrm{di}\} \cdot \{\mathrm{di}\})) &
\end{array}
$$

$$
\begin{array}{llll}
\leftarrow & \{\mathrm{f}, \mathrm{fi}\} \cap (& (\{\mathrm{d}, \mathrm{oi}, \mathrm{f}\} & \cap\ \{\mathrm{d}\}) \cup \\
 & & (\{\mathrm{eq}, \mathrm{s}, \mathrm{si}\} & \cap\ \{\mathrm{eq}, \mathrm{b}, \mathrm{bi}, \mathrm{m}, \mathrm{mi}, \mathrm{d}, \mathrm{di}, \mathrm{o}, \mathrm{oi}, \mathrm{s}, \mathrm{si}, \mathrm{f}, \mathrm{fi}\} \cup \\
 & & (\{\mathrm{eq}, \mathrm{s}, \mathrm{si}\} & \cap\ \{\mathrm{eq}, \mathrm{d}, \mathrm{di}, \mathrm{o}, \mathrm{oi}, \mathrm{s}, \mathrm{si}, \mathrm{f}, \mathrm{fi}\}) \cup \\
 & & (\{\mathrm{di}, \mathrm{o}, \mathrm{fi}\} & \cap\ \{\mathrm{di}\}))
\end{array}
$$

$$
\leftarrow \quad \varnothing
$$

Now, $t \neq C_{13}$ so the entries are updated and the related paths are added to the queue (in practice, the algorithm would halt and report the inconsistency).

Theorem 4.6. The all-to-all consistency algorithm of Fig. 4.5 ensures the labels are minimal with respect to all subgraphs of four vertices and requires O(n^4) time, where n is the number of intervals or points.

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

We say an **IA** network is **k-minimal** if it has the property that each label is minimal with respect to all subgraphs of k vertices. Thus, algorithm AAC ensures that a network is four-minimal. We know that strongly k-consistent implies k-minimal. But is the converse true? If the networks are **SA** or **PA** networks, the algorithm also ensures the networks are strongly four-consistent (see the inductive proof of Theorem 4.8 and Corollary 4.9). However, it is easy to find examples of **IA** networks that are four-minimal but not strongly four-consistent. Fig. 4.6 shows an example. The labels are all the minimal labels, hence the network is four-minimal. But the network is not strongly four-consistent. Consider the instantiation $X_1 \leftarrow <1, 2>$, $X_3 \leftarrow <3, 5>$, and $X_4 \leftarrow <4, 6>$. This instantiation satisfies the relations between the variables. Thus, if the network is four-consistent, there should also be an instantiation of X_2 such that all the relations between variables are satisfied. There is no such instantiation, however. This corrects a claim in (van Beek, 1989) where algorithm AAC is said to also achieve four-consistency for **IA** networks.

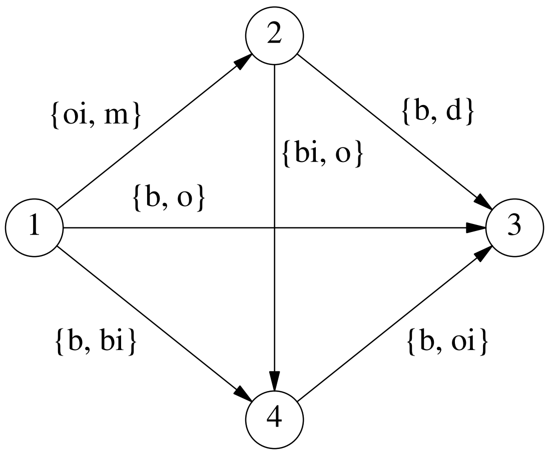


Fig. 4.6. Minimal but not four-consistent

Freuder (1978) gives an algorithm for achieving strong k-consistency, for any $k \leq n$, in O(n^k) time (Seidel, 1983), where n is the number of variables in the network. Thus, his algorithm can also be used to ensure the labels are k-minimal. Our algorithm has two advantages over Freuder's algorithm in this setting. First, formulating the problem so that Freuder's algorithm is applicable squares the problem size (see the previous section). Second, Freuder's algorithm must construct and manipulate non-binary constraints to enforce strong k-consistency, making implementation of the algorithm more difficult.

The idea for developing the initial better approximation algorithm can be generalized to develop successively more expensive algorithms that compute progressively better approximations. The algorithms would ensure that each label was k-minimal. For example, the next algorithm would define composition of labels on structures that share a triangle. This is only of theoretical interest since higher orders of consistency quickly become impractical for all but the smallest problems.

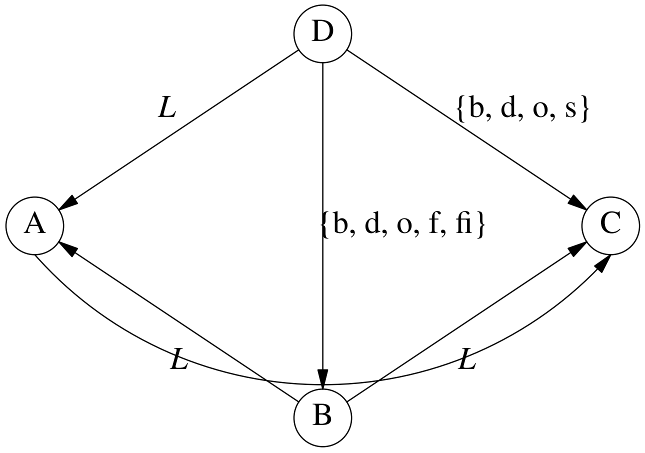
4.3.3. Easy Special Cases

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.

Easy Special Cases of the Path Consistency Algorithm

Previous work has identified classes of binary relations for which the path consistency algorithm gives exact answers (Dechter et al, 1989; Montanari, 1974; Valdés-Pérez, 1986). 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 fall into this class as Montanari assumes the domains of the relations are finite but in **IA** networks the domains are infinite. Valdés-Pérez (1986) shows that the path consistency algorithm is exact for **IA** networks that use only the basic relations of **IA**.

Recall that Fig. 4.2 shows a **PA** network for which, contrary to a claim by Vilain and Kautz, the path consistency algorithm does not find the minimal labels. The implication in Vilain and Kautz (1986) was that the minimal labels for an **SA** networks could be found by translating it into a **PA** network, finding the minimal labels, and then translating it back. Here we show that the counter-example of Fig. 4.2 can arise if we translate **SA** networks into **PA** networks. Hence, this method is not exact for **SA** networks. Below is the graphical representation of the example **SA** network.



{eq, b, di, o, s, si, fi}

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

Applying the algorithm of Fig. 4.1 results in no changes; the network is already path consistent. However, the relation $\le$ between A^- and B^+ is not the minimal label; the minimal label is $<$. 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 path consistency algorithm is not exact for **SA** networks, whether applied directly to the network or to a **PA** network translation of the network.

We next prove that the path consistency algorithm is exact for $\mathbf{SA_c}$ networks. The proof parallels the proof of the exactness of path consistency for $\mathbf{PA_c}$ networks (Theorem 4.3). In the proof of Theorem 4.3 the key step was showing that any two bounds have a point in common and then applying Helly's theorem to show that all the bounds together have a point in common. Here we need to show that any three bounds have a point in common before we can apply Helly's theorem.

Theorem 4.7. 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

$$< s_i, e_i > \ R_{ik} \ < X_k^s, X_k^e >, \qquad i = 1, \ldots, k-1 \tag{4.5}$$

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 $\{<, \le, =, \ge, >, ?\}$ between the endpoints of the intervals. For example, if the relation R_{1k} is the disjunction of the "before" and the "meets" relations,

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

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

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

For each i in Eq. (4.5) 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 Eq. (4.5) 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 there exists at least one instantiation of variable X_k such that

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

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. □

Thus, as is true for $\mathbf{PA_c}$ networks, any algorithm that achieves path consistency is exact for $\mathbf{SA_c}$ networks (see Section 4.1 for a discussion of algorithms for ensuring path consistency). In particular, Proposition 2.1 shows that Algorithm 5.5 of Aho et al. (1974) can be used.

Easy Special Cases of the New All-to-All Consistency Algorithm

In the remainder of this section we show that our new all-to-all consistency algorithm (AAC) is exact for **SA** networks (and, as a corollary, for **PA** networks). The strategy is to identify first why path consistency is not sufficient and where the proof of Theorem 4.7 fails for **SA** networks.

The inductive step of the proof of Theorem 4.7 required us to show that given an 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. Showing this relied on the fact that the relations between variables can be expressed as conjunctions of binary relations on the end points of the intervals and that these relations place bounds on the instantiations of the kth variable that define convex sets.

If the relations between variables are from **SA**, expressing some of the relations as conjunctions of binary relations on the end points (or, equivalently, as a **PA** network) requires us to use the $\neq$ binary relation and the bounds no longer define convex sets. As the example in the previous section shows, the translation into **PA** relations may give rise to the **PA** network shown in Fig. 4.2. Thus, path consistency cannot guarantee that the intersection of any three bounds have a point in common. However, algorithm AAC can guarantee that any three bounds have a point in common.

Theorem 4.8. The all-to-all consistency algorithm of Fig. 4.5 correctly finds the minimal labels between all pairs of intervals when applied to **SA** networks.

***Proof*.** The proof of Theorem 4.8 follows the inductive proof of Theorem 4.7 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 the bounds are *almost* convex and, by the discussion in the proof of Theorem 4.4, 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 proof of Theorem 4.4. These bounds cannot arise if the labels of the **SA** network are minimal with respect to all subgraphs of four vertices. Hence, the bounds have a point in common. □

Corollary 4.9. The all-to-all consistency algorithm of Fig. 4.5 correctly finds the minimal labels between all pairs of points when applied to **PA** networks.

5

Finding the Feasible Relations: One-to-All Problems

In this chapter we examine the one-to-all version of the computational problem of finding the feasible relations for the point algebra, **PA**, and the interval algebra, **IA**. We present an efficient approximation algorithm and show that the algorithm is exact for a useful subset of the interval algebra and of the point algebra. We also give algorithms for determining the feasible relations between just two points or intervals in **PA** and **IA** networks. Finally, for **IA** networks we experimentally test the quality of the approximations produced by the all-to-all and one-to-all approximation algorithms.

5.1. An Approximation Algorithm

The algorithms given in the previous chapter 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. Here we present an efficient approximation algorithm for the one-to-all version of the problem.

The algorithm (see Fig. 5.1) 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. 5.1, 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. The algorithm can also be viewed as a special case of the path consistency algorithm. 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 5.1. The one-to-all consistency algorithm of Fig. 5.1 requires O(n^2) time, where n is the number of intervals or points.

***Proof*.** Initially the 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). □

Input: A source vertex s and an adjacency matrix C.

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

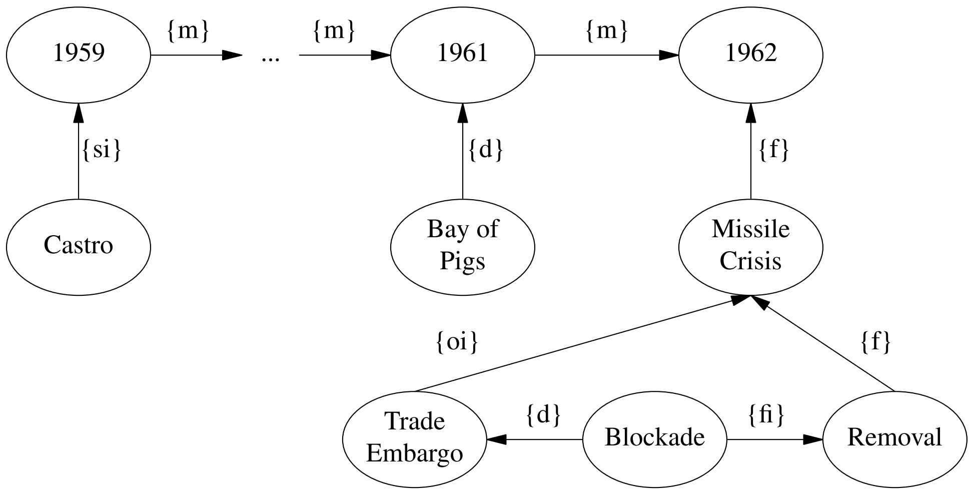
```
procedure OAC
begin
    $L \leftarrow V - \{ s \}$
    while $L$ is not empty do begin
        select a vertex $v$ from $L$ such that $C_{sv}$ is a minimum
        $L \leftarrow L - \{ v \}$
        for each $t$ in $V$ do begin
            $l \leftarrow C_{st} \cap C_{sv} \cdot C_{vt}$
         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

Fig. 5.1. One-to-all consistency algorithm

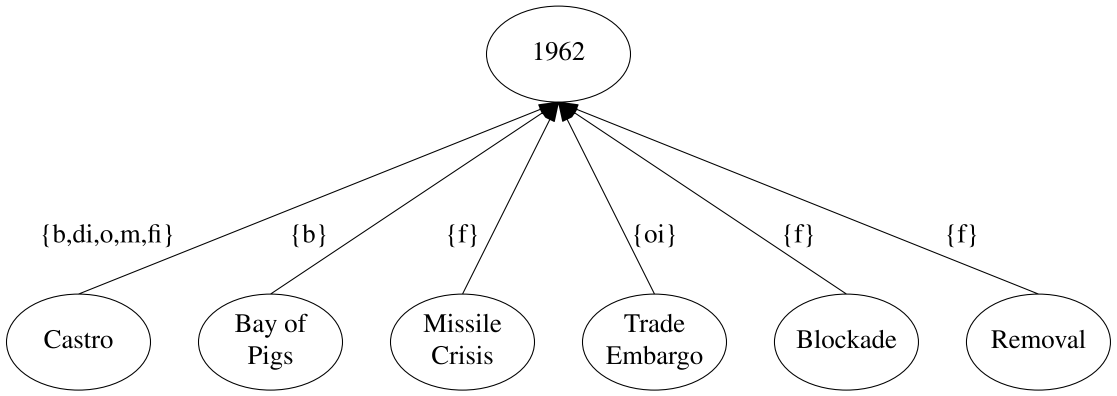
The one-to-all consistency algorithm requires the operation of finding the minimum of a set of labels. The final result of the algorithm is independent of how the minimum is chosen, but the choice does affect the number of iterations. In practice, the number of iterations of the algorithm can be halved compared to a random choice by choosing the minimum based on the following order on the set of all labels. Assign weights to the 13 basic interval relations according to how restrictive the relation is. Restrictiveness is measured by successively composing the basic relation with every possible label and summing the cardinalities of the results. With suitable scaling we get the following weights for the 13 basic relations: 1: eq; 2: fi, f, mi, m, si, s; 3: bi, b, di; and 8: d, oi, o.

The weight of a label is then the sum of the weights of its elements.

Here is an example of the one-to-all consistency algorithm and its use in knowledge-based systems. The temporal information we represent is some Cuban history. We adopt a suggestion by Vilain (1982) and use dates as time interval constants. This allows queries about the relationship between an event and a date as well as the relationship between two events.



Suppose we now ask, "What events occurred in the year 1962?" To answer this question, we use the one-to-all algorithm to determine the feasible relations between the interval 1962 and every other interval. The results are shown below.



The events that occurred are, "The Cuban missile crisis, the trade embargo, the naval blockade of Cuba, and the removal of Soviet missiles from Cuba." From the information represented, it may or may not be the case that Castro is still in power in 1962 (we only stated that he came into power in 1959).

5.2. The Point Algebra

Ghallab and Mounir Alaoui (1989) give an incremental method for adding and deleting temporal information represented in the point algebra. The data structure used is a rooted tree with an indexing scheme. Their method relies on being able to identify all points that are part of a ≤ cycle and therefore are equal to each other, but they do not describe how this would be done. They show experimentally that the method works well in practice.

It is shown in Corollary 5.3 that the one-to-all consistency algorithm (OAC) is exact for $\mathbf{PA_c}$ networks, provided the network is consistent (see the next section for a proof and further discussion). But OAC is not exact for **PA** networks. The network shown in Fig. 4.2 with vertex A as the source vertex is an example where OAC does not correctly find the minimal labels. For **PA** networks we give an algorithm for finding the minimal label between just one pair of points (Fig. 5.2). The algorithm is straightforward: for each possible basic relation r in the label on edge (s, t) we test whether r is feasible, i.e., whether a consistent instantiation exists with r satisfied. The algorithm is easily seen to take $O(n^2)$ time as the for loop is executed at most 3 times and the test for a consistent instantiation can be done in $O(n^2)$ time using step (1) of algorithm CSPAN (Fig. 3.5).

Input: A source vertex s, a target vertex t, and a **PA** network represented as a matrix C where entry C_{ij} is the label on edge (i, j).

Output: L, the minimal label for C_{st}.

```
procedure ML
begin
     $L \leftarrow \emptyset$
     for each $r \in C_{st}$ do
         $W_{ij} \leftarrow C_{ij}, \quad i, j = 1, \ldots, n$
         $W_{st} \leftarrow r$
         if ($\exists$ a consistent instantiation of $W$) then
             $L \leftarrow L \cup \{r\}$
end
```

Fig. 5.2. One-to-one minimal label algorithm for PA networks

5.3. The Interval Algebra

The one-to-all consistency algorithm (OAC) discussed at the beginning of the chapter can be used to find approximations to the minimal labels for **IA** networks. (The results of some computational experiments designed to estimate the quality of the approximate solutions of OAC and of the all-to-all algorithms of Chapter 4 are summarized and discussed in the next section). The algorithm is, of course, not exact for general **IA** networks. An exact algorithm for finding the minimal label between just one pair of intervals is easily constructed using the same idea as in the algorithm for **PA** networks shown in Fig. 5.2. Whereas in the **PA** version the test for a consistent instantiation is done using Step (1) of algorithm CSPAN, the **IA** version would use algorithm CS_BackTrack (Section 3.2.2) to test for a consistent instantiation. The algorithm just described, denoted

ML_BackTrack, was implemented and used to find exact solutions for the computational experiments described in the next section. Our experience suggests ML_BackTrack is practical for instances of the problem where only a few of the relations between intervals fall outside of the special subset **SA**, but takes much time when most of the relations are outside of **SA**. In such cases, a better approach is to, if possible, accept approximate solutions to the problems.

In the remainder of this section we examine how far we must restrict the expressive power of the representation language to guarantee that the one-to-all consistency algorithm (OAC) is exact. 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 5.2. The one-to-all consistency algorithm of Fig. 5.1 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 prove that, for those labels computed by OAC, the results are equivalent to those of PC. Then, since PC is exact by Theorem 4.7, 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,

$$C_{sj} \subseteq C_{sk} \cdot C_{kj}, \quad j, k = 1, \ldots, n \tag{5.1}$$

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

$$C_{st} \subseteq C_{sv} \cdot C_{vt}, \quad v = 1, \ldots, n \tag{5.2}$$

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 Eq. (5.2). 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 (iv) of Proposition 2.1) 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 (clause (iii) of Proposition 2.1),

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

Applying Eq. (5.1) repeatedly,

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

$$= C_{st}$$

A contradiction. □

From clause (iv) of Proposition 2.1 stating a distributivity property of $\mathbf{PA_c}$, the point algebra that excludes the ≠ relation, we have also proved that OAC is exact for $\mathbf{PA_c}$ networks.

Corollary 5.3. The one-to-all consistency algorithm of Fig. 5.1 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 2.1 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 Chapter 3, we gave 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 is consistent (see Chapter 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).

5.3.1. 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 the path consistency algorithm 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.

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, OAC, PC, and AAC, chose a particular edge, determined the minimal label on that edge using the exact backtracking algorithm, ML_BackTrack, 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. 5.3 summarizes the results for

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

two distributions. Distribution one was chosen to approximate instances that may arise in a planning application (as estimated from a block-stacking example in Allen and Koomen, 1983). The important parameter in the planning application is that the relations between most of the actions are originally unconstrained (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. 5.3). 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

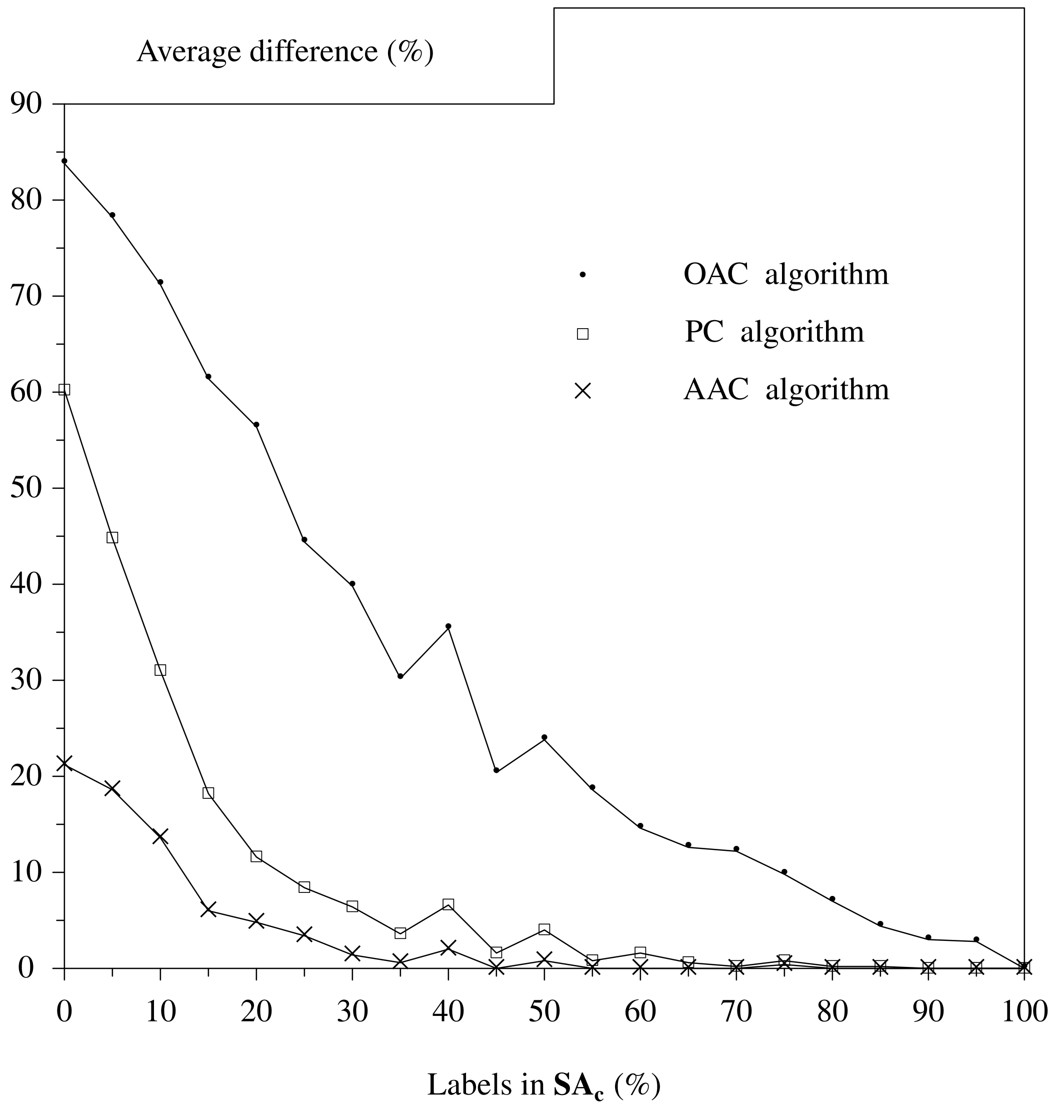


Fig. 5.4. Percentage differences between the approximation algorithms and an exact algorithm for various percentage of labels in $\mathbf{SA_c}$. 250 tests performed for each subinterval; problem size is 25.

well the OAC, PC, and AAC algorithms approximate the exact solution. Recall that Theorem 4.7 (Theorem 5.2) 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. 5.4). Thus we have an effective test for predicting whether it would be useful to apply a more expensive algorithm.

6

Applications and a Practitioner's Guide

In this chapter we survey three example applications of the point and interval algebras chosen from the literature to show where the results of this thesis could be useful. Following that we summarize our results and give some more general remarks about how the results presented could be used in practice.

6.1. Example Applications

Knowledge-Based Systems. The interval algebra is used as part of a language called Telos for representing knowledge about information systems (Koubarakis et al., 1989; Topaloglou and Koubarakis, 1989). We focus here on the temporal component of Telos. The temporal component allows the representation of (and queries about) the history of the domain and the system's beliefs about that history. Other features of Telos are that it provides time interval constants referring to one year intervals (e.g., 1985), one month intervals (e.g., 1985/12), one day intervals (e.g., 1986/12/25), other finer levels of granularity, and semi-infinite intervals. Here is a small example using the syntax of the Telos knowledge representation language (Koubarakis et al., 1989).

```
TELL TOKEN martian
        IN paper ( at 1986/10..* )
        WITH
           author
           first_author      : Stanley ( at 1986/10..* );
                             : LaSalle ( at 1987/1..* );
                             : Wong ( before 1987/5 )
           title             : ‘The MARTIAN System’;
END
```

The TELL command adds new information to the system. This example states that (i) 'martian' is an instance of the class 'paper' and it has been a paper since the beginning of 1986/10, (ii) the first author Stanley started writing the paper at the beginning of the same month, (iii) LaSalle became an author of the paper since 1987/1, and (vi) Wong was an author before 1987/5. The intended meaning of the semi-infinite interval in (ii) is that Stanley will remain an author of the paper into the indefinite future. Subsequent TELL commands can, of course, update the temporal information in the system if this fact should become no longer true. We can now ask questions of the system.

ASK : LaSalle $\in$ martian.author [during 1988]

ASK : LaSalle $\in$ martian.author [during 1988]
BELIEVED at 1985/1/1

The first ASK command queries whether the system currently believes that LaSalle was an author of the Martian paper during 1988 (Answer: Yes). The second ASK command queries whether the system believed on January 1, 1985 that LaSalle was an author of the Martian paper during 1988 (Answer: No). These examples illustrate that the system can answer queries with respect to its current state and with respect to previous states.

The authors of Telos decided to use only the thirteen basic relations, foregoing representational completeness in favor of guaranteed exact answers in quick time. (To be precise, the system maintains in a $\mathbf{PA_c}$ network the relations between the end points of every interval.) Moreover, the Telos system uses the path consistency algorithm to derive new consequences of the temporal information asserted by the TELL command. The basic interval relations are 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 the Telos 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. Finally, the decision procedure for **PA** networks (Step (1) of algorithm CSPAN) may be of use in the TELL operator to detect if new temporal information is inconsistent with the temporal information already stored.

Natural Language Processing. Almeida (1987) and Song and Cohen (1988) use the interval algebra in their solutions 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 indefinite. Almeida, in his solution, restricts his representation language to {b}, {m}, {eq}, {b, m}, {eq, d, s, f}, {eq, s}, {eq, f}, and the inverses of those relations. Song and Cohen restrict their representation language to the thirteen basic relations plus {b, o, m} (called precedes) and {eq, d, s, f} (called includes). Song and Cohen's algorithm determines the temporal relations between events mentioned in a narrative by analyzing each new utterance in turn, and determining the relation of the event in the current utterance to an event in a previous utterance that is currently in focus. The more vaguely specified relations precedes and includes are used in the absence of more specific temporal adverbials or connectives. It turns out that both Almeida's and Song and Cohen's representation languages are also subsets of $\mathbf{SA_c}$. Thus, once we have extracted the possibly indefinite relations between some of the events mentioned in the narrative, we can determine *exactly* the feasible relations between all pairs of the events using the path consistency algorithm or between one event and every other event using the one-to-all algorithm. As well, the consistent scenario algorithm may be of use in more recent work by Song (1990). Song (1990) applies the earlier work in discourse processing to the recognition of a speaker's plan underlying a discourse and the plan's temporal constraints. The decision portion of the consistent scenario algorithm can be used as a heuristic to prune away from further consideration any proposed plans that are temporally inconsistent.

In Chapter 4 we gave an example of representing the temporal information in a short narrative and of drawing inferences using the path consistency algorithm. Drawing these inferences are part of understanding a story and allow us to answer queries about the relations between the events in the story that are only implicit. We repeat the narrative below.

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.

With reference to the example, we can answer queries such as "Did Fred finish his coffee before going for a walk?" (Answer: Yes.), "What is the relationship between Fred reading the paper and going for a walk?" (Answer: Fred read his paper before going for a walk.), and "Is it possible that Fred finished his paper before starting his breakfast?" (Answer: No.).

Planning. The interval algebra is used in planning (Allen and Koomen, 1983; Pelavin and Allen, 1986; Hogge, 1987, and recall the block-stacking example in Chapter 3). 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. In this framework a plan can be represented as an **IA** network.

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).

In step (ii), the planner, before starting to search for a plan, 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. This may be useful if it allows us to detect early that a plan is temporally inconsistent before much work has been expended expanding the plan. This is speculation, however, and actual experience with a planner is needed to determine whether the planner is made more or less efficient as a result.

In step (iii), a temporal ordering of the operators must be selected. This corresponds to finding a consistent scenario of an **IA** network. Thus, the backtracking algorithm for finding a consistent scenario (CS_BackTrack) is useful here. In Chapter 3 we gave an extensive planning example and showed experimentally that the backtracking algorithm worked well in practice on planning problems.

6.2. A Practitioner's Guide

The following observations may be relevant in deciding where the different results are applicable.

6.2.1. Finding a Consistent Scenario

A guide to applying the results for finding consistent scenarios is straightforward. The results themselves are summarized in the table below. The algorithms are the consistent scenario algorithm for **PA** networks (CSPAN, Fig. 3.5) and the consistent scenario backtracking algorithm (CS_BackTrack, 3.2.2).

Algorithm	**Exact**	**Cost**
CSPAN	$\mathbf{PA_c}$, **PA** $\mathbf{SA_c}$, **SA** (translate)	$O(n^2)$
CS_BackTrack	$\mathbf{SA_c}$, **SA**, **IA**	$O(n^2\, rk^r)$[8]

The practitioner must first choose between the point- and interval-based representation languages. If our representation language is **PA** or **SA**, or subsets of them, we can find a consistent scenario quickly using algorithm CSPAN. If our representation language is **IA** the best we can offer is algorithm CS_BackTrack, which is exponential in the worst case. One bright spot is that the algorithm works well in practice for problems that arise in planning. The algorithm will 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 we only rarely want to represent relations that are not in **SA**. 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 CS_BackTrack 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 CS_BackTrack algorithm can simply be used as needed).

6.2.2. Finding the Feasible Relations

A guide to applying the results for finding the feasible relations is less straightforward. The results themselves are summarized in the table below. The algorithms are minimal label (ML, Fig. 5.2), one-to-all consistency (OAC, Fig. 5.1), path consistency (PC, Fig. 4.1), feasible relations (FEASIBLE, Fig. 4.3), all-to-all consistency (AAC, Fig. 4.5), and minimal label backtracking (ML_BackTrack, Section 5.3). The following methodology is suggested for choosing the appropriate algorithm for finding the feasible relations. First, the practitioner must choose between the point- and interval-based representation

8 Here r is the number of edges not labeled with I, the set of all basic relations, and k is the cardinality of I.

languages.

Algorithm	Exact	Approximate	Cost
ML	**PA$_c$**, **PA**		O(n^2)
OAC	**PA$_c$** **SA$_c$**	**PA** **SA**, **IA**	O(n^2)
PC	**PA$_c$** **SA$_c$**	**PA** **SA**, **IA**	O(n^3)
FEASIBLE	**PA$_c$**, **PA** **SA$_c$**, **SA** (translate)		O(max(mn^2, n^3))
AAC	**PA$_c$**, **PA** **SA$_c$**, **SA**	 **IA**	O(n^4)
ML_BackTrack	**SA$_c$**, **SA**, **IA**		O(n^2 rk^r)

Point-based. If a point-based language is used (**PA$_c$**, **PA**) the choice of algorithm is easily seen in the table above.

PA$_c$: For **PA$_c$** we would use either the OAC or the PC algorithm, depending on whether we wanted to determine the feasible relations between just a few points or the feasible relations between all pairs of points.

PA: For **PA** we would use either the ML or the FEASIBLE algorithm, depending again on whether we wanted to determine the feasible relations between just a few points or the feasible relations between all pairs of points. In some computational experiments it was found that the cost of the FEASIBLE algorithm is negligibly greater than that of the PC algorithm. This provides evidence for the argument that, if we want all pairs of feasible relations, there is little incentive to restrict the representation language from **PA** to **PA$_c$** or to accept approximate solutions in order to use the PC algorithm.

Interval-based. If an interval-based language is used (**SA$_c$**, **SA**, **IA**) the choice of algorithm of course again depends on how expressive the representation language must be. But here we must sometimes also decide whether an exact solution is necessary or whether an approximate one is acceptable. In the discussion above about finding consistent scenarios, we noted that there the practitioner could, if desired, not commit at the outset to a restricted representation language. This was true because there it is easy to determine algorithmically the ‘‘best’’ algorithm for solving the problem. Such is not the case here. For example, if it is determined that an **IA** network is not one of the special cases that can be solved efficiently, do we then proceed to solve the problem exactly, perhaps taking a long time in doing so, or do we use an approximation algorithm? Clearly, this decision belongs to the practitioner. Further discussion is divided according to the choice of representation language.

SA$_c$: As for **PA$_c$**, for **SA$_c$** we would use either the OAC or the PC algorithm for finding the feasible relations between a few intervals or between all pairs of intervals, respectively.

SA: For **SA**, if we want exact solutions the choice is between the FEASIBLE and the AAC algorithm. FEASIBLE is, in practice, much faster than AAC but more difficult to implement as an **SA** network must first be translated into a **PA** network, solved, then translated back again. If we are willing to accept approximate solutions, the approximate solutions produced by both PC and OAC are almost always the exact solutions. This should be balanced against the fact that we can get exact solutions at little extra cost by translating the **SA** network into a **PA** network and using procedure FEASIBLE.

IA: It is generally impractical to compute exact solutions if our representation language is **IA**. The decision to be made then becomes: are we able to restrict our representation language sufficiently so that exact answers can be tractably computed or, alternatively, do we accept approximate solutions? In computational experiments it was found that the approximate solutions are almost always the same as the exact solutions if many (greater than one-half) of the relations we want to represent are in $\mathbf{SA_c}$ or **SA**. Hence, in these cases the OAC and PC algorithms are useful. However, when most (less than one-fifth) of the relations we want to represent are outside of $\mathbf{SA_c}$ or **SA** the OAC and PC algorithms can produce poor approximations. Given an arbitrary **IA** network, the predictive test proposed in Chapter 5 is a useful tool for deciding whether to apply the PC algorithm or a more expensive approximation algorithm such as AAC. This should be balanced against the increased cost of AAC. The AAC algorithm is particularly appropriate for small networks, but these are precisely the cases where the greatest improvement was found (again, see Chapter 5). Further, large networks can sometimes be partitioned into several small networks and the algorithm is then applicable. Allen (1983) proposes ‘‘reference intervals’’ for this purpose and Koomen (1989) continues this work. Briefly, reference intervals are a way of grouping intervals into clusters so that we reason about the relations between intervals within clusters only.

7

Future Work and Conclusion

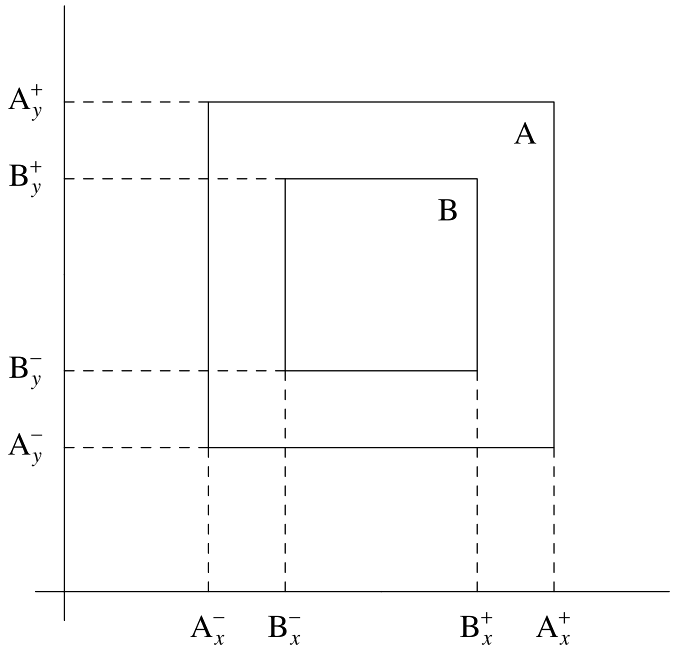
In this chapter we discuss some potential directions for future work. One direction is applying and extending the results of the thesis to spatial reasoning. Another is extending the kinds of temporal information we can represent to include quantitative information. We end with some conclusions.

7.1. Future Work

Spatial Reasoning. Until now, we have interpreted points and intervals as temporal objects. However, all of our results apply equally well if we interpret points and intervals as spatial objects. Thus, it should be investigated how useful the algebras and our results are for spatial reasoning. An example where **PA** and **IA** may be useful is in spatial layout (Eastman, 1970, 1973; Earl and March, 1979; Baykan and Fox, 1987). Spatial objects such as desks, lamps, and chairs would be modeled by points or intervals. The relations between points or intervals would represent constraints on the placement of the spatial objects such as, the lamp must be beside the chair, or, two objects cannot be in the same place. A consistent scenario would then correspond to a layout that satisfies all the constraints.

Just as the algebra of one-dimensional intervals generalizes the algebra of zero-dimensional points, an obvious next step is to continue this generalization and develop algebras for representing and reasoning about the qualitative relationships between two-dimensional and three-dimensional convex regions.

We may wish to restrict the allowed spatial objects rather than allow arbitrarily complex convex regions. Malik and Binford (1983), in their work on representing quantitative spatial information, suggest representing spatial information about each dimension separately, taking the projection along the x-axis, y-axis, and z-axis, respectively. This is equivalent to restricting the allowed spatial objects to be rectangles (2-D) and boxes (3-D). For qualitative spatial information, we would then use **IA** to represent each dimension separately. As an example, the two-dimensional relation, A inside B, is represented as the conjunction of interval relations, B_x during A_x and B_y during A_y.



We may also wish to restrict the allowed spatial relations along each dimension rather than allow the full interval algebra. For example, many of the common spatial relations—such as, inside, to the left of, to the right of, below, above, occludes, in front of, and behind—can be represented using just $\mathbf{SA_c}$.

A General Temporal Reasoning System. In this thesis we examined frameworks for representing and reasoning about qualitative temporal information such as that in statements that (a, b) specify the relative ordering of end points of intervals, (c) specify only that two intervals of time intersect, and (d) specify the relative ordering of entire intervals.

a. The Cuban Missile crisis took place during Kennedy's presidency.

b. Fred put the paper down and drank the last of his coffee.

c. Fred was reading the paper while eating his breakfast.

d. I'll come by either before my class or after it.

For some applications we may also want to represent and reason about quantitative temporal information such as that in statements that (e) specify the endpoints of intervals of time, (f) describe times in terms of their distances from other times, (g) specify the duration of an event, and (h) describe disjoint uncertainty about the duration of an interval.

e. The concert begins at 2:00.

f. I'll phone you five minutes before I leave.

g. The meeting lasted for an hour.

h. Fred went to work either by car (30-40 minutes) or by bus (60-75 minutes).

Ideally, what is desired is a general temporal reasoning system for storing, retrieving, and answering queries about both qualitative and quantitative information and that is useful across applications. Our results can be viewed as a contribution to a special purpose reasoner in such a general system (see Miller and Schubert, 1988). The general system would include other special purpose reasoners for other kinds of temporal information such as quantitative information about the distances between intervals or points (Dechter et al., 1989; Dean and McDermott, 1987), or combinations of qualitative and quantitative information (Allen and Kautz, 1985; Ladkin, 1989; Schmiedel, 1988a, 1988b). Building such a general system would be a useful project.

Some interesting problems arise by allowing both qualitative and quantitative information as both kinds of information constrain the other. For a simple example, let A^- and A^+ be the end points of interval A. Suppose we know that A before or overlap B, $B^+ - A^+ \geq 0$, and $A^+ - B^- > 0$. From this we can derive the stronger information that A overlaps B, $B^+ - A^+ > 0$, and $A^+ - B^- > 0$. Work has been done on these problems (Ladkin, 1989; Schmiedel, 1988a) but open problems remain.

7.2. Conclusion

Allen (1983) and Vilain and Kautz (1986) give frameworks for representing and reasoning about qualitative temporal information. We looked at two reasoning tasks that arise in these frameworks: Given (possibly indefinite) 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 relationships between some or 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 asymptotically optimal and 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 analytically to always be better than previous proposals and is shown experimentally to be useful for planning problems.

For finding the feasible relationships, we give a counter-example to a result in the literature and give exact algorithms for **PA** and **SA** networks. We also show that for both the all-to-all and one-to-all versions of the problems, previously known algorithms are exact for $\mathbf{PA_c}$ and $\mathbf{SA_c}$ networks, where $\mathbf{PA_c}$ and $\mathbf{SA_c}$ are subsets of **PA** and **SA**, respectively. Finally, the intractability of finding exact solutions for **IA** networks led us to develop new approximation algorithms. We presented the results of some computational experiments that attempted to characterize the quality of the approximations. The experiments led us to propose a test for predicting when the approximation algorithms produce solutions that are close to the exact solutions.

Appendix A

In this appendix 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. A^- and A^+ represent the start and end points of interval A, respectively, and $A^- < A^+$ and $B^- < B^+$ are true for every translation.

SA and $\mathbf{SA_c}$ contain only a small but important and useful subset of the 2^{13} elements that **IA** contains. But what *cannot* be expressed in **SA** that can be expressed in **IA** 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 also requires disjunction.

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

$$(A^- > B^- \wedge A^- > B^+ \wedge A^+ > B^- \wedge A^+ > B^{+)}$$

The nearest approximation using only conjunction is the following,

$$A^- \neq B^- \wedge A^- \neq B^+ \wedge A^+ \neq B^- \wedge A^+ \neq B^+$$

So, the nearest approximation to "A {b, bi} B" using an element of **SA** is "A {b, bi, d, di, o, oi} B". But as can be seen, this allows, for example, A to overlap B, which we did not intend.

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

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

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

Appendix B

Interval algebra operations (Allen, 1983).

Composition of two basic relations (the "equals" relation is omitted; let I be the set of all basic relations, {eq, b, bi, m, mi, o, oi, d, di, s, si, f, fi}, L be {eq, d, di, o, oi, s, si, f, fi}, M be {b, o, m}, and N be {bi, oi, mi}):

·	b	bi	d	di	o	oi	m	mi	s	si	f	fi
b	b	I	M d s	b	b	M d s	b	M d s	b	b	M d s	b
bi	I	bi	N d f	bi	N d f	bi	N d f	bi	N d f	bi	bi	bi
d	b	bi	d	I	M d s	N d f	b	bi	d	N d f	d	M d s
di	M di fi	N di si	L	di	di o fi	di oi si	di o fi	di oi si	di o fi	di	di oi si	di
o	b	N di si	d o s	M di fi	M	L	b	di oi si	o	di o fi	d o s	M
oi	M di fi	bi	d oi f	N di si	L	N	di o fi	bi	d oi f	N	oi	di oi si
m	b	N di si	d o s	b	b	d o s	b	eq f fi	m	m	d o s	b
mi	M di fi	bi	d oi f	bi	d oi f	bi	eq s si	bi	d oi f	bi	mi	mi
s	b	bi	d	M di fi	M	d oi f	b	mi	s	eq s si	d	M
si	M di fi	bi	d oi f	di	di o fi	oi	di o fi	mi	eq s si	si	oi	di
f	b	bi	d	N di si	d o s	N	m	bi	d	N	f	eq f fi
fi	b	N di si	d o s	di	o	di oi si	m	di oi si	o	di	eq f fi	fi

Inverse of a relation:

C	eq	b	bi	d	di	o	oi	m	mi	s	si	f	fi
C^{-1}	eq	bi	b	di	d	oi	o	mi	m	si	s	fi	f

Point algebra operations (Vilain and Kautz, 1986).

Composition of two relations:

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

Intersection of two relations:

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

Inverse of a relation:

C	$=$	$<$	$\leq$	$>$	$\geq$	$\neq$
C^{-1}	$=$	$>$	$\geq$	$<$	$\leq$	$\neq$

Appendix C

In this appendix we prove Theorem 3.1 and Theorem 4.6, and we give a proof that the consistent scenario problem and the minimal labeling problem are NP-Complete.

> **Theorem 3.1.** The algorithm in Fig. 3.5 correctly finds a consistent scenario of a **PA** network in $O(n^2)$ time, where n is the number of points.

***Proof*.** We prove here statements (a) and (b) below. The rest of the algorithm is justified by the discussion in the text of Chapter 3.

a. The vertices v and w are forced to be equal precisely when there is a cycle of the form

$$v \le \cdots \le w \le \cdots \le v \tag{C.1}$$

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

b. The network is inconsistent precisely when there is a cycle of the form

$$v = \cdots = w \ne v, \tag{C.2}$$

or of the form,

$$v \le \cdots \le w \le \cdots \le v \ne w, \tag{C.3}$$

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

$$v < \cdots < w < \cdots < v. \tag{C.4}$$

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

We first give some notation. Let $P = (v, x_1), (x_1, x_2), \ldots, (x_r, w)$ be a simple path (no vertex is repeated in the sequence) from vertex v to vertex w, $v \ne w$, in the graph of the network. The label $l(P)$ of a path P is defined as the composition of the labels of the edges of P taken in order, $ip(v, w)$ is defined as the intersection of the labels of all the simple paths from v to w, and $ml(v, w)$ is defined as the minimal label on the edge (v, w).

The key to the proof is that by Theorems 4.3 and 4.4, the path consistency algorithm correctly determines the minimal label on an edge if the minimal label is one $\{\emptyset, <, =, >, \ne, ?\}$. This is equivalent to saying that, for each $r \in \{\emptyset, <, =, >, \ne, ?\}$,

$$ip(v, w) = r \equiv ml(v, w) = r.$$

Thus, we need to look at only the simple paths between vertices to prove statements (a) and (b).

If $ml(v, w) =$ '$=$', then $ip(v, w) =$ '$=$', and there must exist simple paths P_i and P_j between v and 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
$l(P_i)$	$\le$	$\ge$	$\le$	$=$
$l(P_j)$	$=$	$=$	$\ge$	

By examination of the composition table (Appendix B) it can be seen that these four cases arise only when there is a cycle of the form in Eq. (C.1).

If $ml(v, w) = \emptyset$, then $ip(v, w) = \emptyset$, and there must exist simple paths P_i, P_j, and P_k between v and 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 (Appendix B) it can be seen that case (1) arises only when there is a cycle of the form in Eq. (C.3), cases (2)-(6) arise only when there is a cycle of the form in Eq. (C.4), and case (7) arises only when there is a cycle of the form in Eq. (C.2). $\Box$

Theorem 4.6. The all-to-all consistency algorithm of Fig. 4.5 ensures the labels are minimal with respect to all subgraphs of four vertices and requires $O(n^4)$ time, where n is the number of intervals or points.

***Proof*.** A label on an edge (i, j) is minimal with respect to all subgraphs of four vertices if, for every pair of vertices k and l,

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

Let C be the adjacency matrix representation of an **IA** or **PA** network. Somewhat more informally, we must prove that for every edge (i, j) and every subgraph of four vertices (i, k, l, j), all of the $R \in C_{ij}$ are feasible with respect to the subgraphs.

We show that, given a particular edge (i, j) and subgraph (i, k, l, j), the algorithm correctly determines only the $R \in C_{ij}$ that are feasible with respect to that subgraph. Then, since the algorithm iterates until this property holds for all such edges and subgraphs, the theorem is proved.

Given an edge (i, j) and a subgraph (i, k, l, j), Eq. (4.2a) in Fig. 4.5 determines,

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

Using Eq. (4.3a) we have,

$$\begin{aligned}
\leftarrow\; & C_{ij} \cap (\bigcup (\{P\} \cdot \{Q\} \cap \{S\} \cdot \{T\}), \\
& \qquad O \in C_{kl}, \\
& \qquad S \in C_{ik},\; P \in (C_{il} \cap \{S\} \cdot \{O\}), \\
& \qquad Q \in C_{lj},\; T \in (C_{kj} \cap \{O\} \cdot \{Q\})
\end{aligned}$$

$$\begin{aligned}
\leftarrow\; & \bigcup (C_{ij} \cap \{P\} \cdot \{Q\} \cap \{S\} \cdot \{T\}), \\
& \qquad S \in C_{ik},\; T \in C_{kj},\; P \in C_{il},\; Q \in C_{lj},\; O \in C_{kl}, \\
& \qquad (\{P\} \cap \{S\} \cdot \{O\}) \neq \emptyset,\; (\{T\} \cap \{O\} \cdot \{Q\}) \neq \emptyset\}
\end{aligned}$$

$$\begin{aligned}
\leftarrow\; & \{R \mid R \in C_{ij},\; S \in C_{ik},\; T \in C_{kj},\; P \in C_{il},\; Q \in C_{lj},\; O \in C_{kl}, \\
& \qquad (\{R\} \cap \{P\} \cdot \{Q\}) \neq \emptyset,\; (\{R\} \cap \{S\} \cdot \{T\}) \neq \emptyset, \\
& \qquad (\{P\} \cap \{S\} \cdot \{O\}) \neq \emptyset,\; (\{T\} \cap \{O\} \cdot \{Q\}) \neq \emptyset\}
\end{aligned}$$

Recall that an adjacency matrix B is a scenario of a network C if $B_{ij} \subseteq C_{ij}$, and $| B_{ij} | = 1$, for all i, j. The scenario is also a consistent scenario if there exists a consistent instantiation of B. The effect of Eq. (4.2a) then is to, for every $R \in C_{ij}$, look for a scenario of this subgraph that involves R. The scenario is checked to ensure it is path consistent. But, by Theorem 4.7 and the fact that the basic relations are in $\mathbf{SA_c}$, this is sufficient to test whether a consistent instantiation exists. Hence, only the $R \in C_{ij}$ that are feasible with respect to this subgraph remain. □

We end with a proof that the consistent scenario problem and the minimal labeling problem are NP-Complete. We do this by first proving that the following decision problem is NP-Complete.

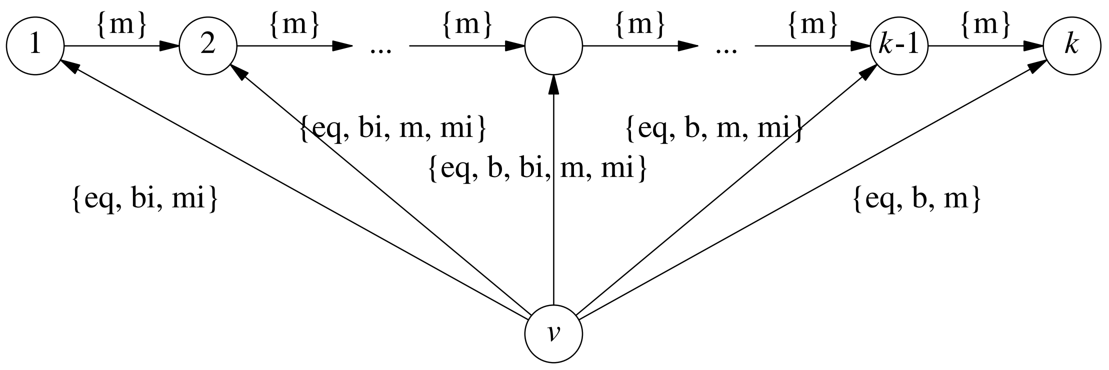
Interval Algebra Consistency (IAC)
Given a directed graph $G = (V, E)$ with labels on the edges from the set of elements of **IA**, does there exist a consistent scenario of the graph?

Kautz (Vilain et al., 1989) gives a proof that IAC is NP-Complete by showing 3-Satisfiability is polynomially transformable to IAC. (An earlier sketch of the proof can be found in Vilain and Kautz, 1986). We show Graph Coloring is polynomially transformable to IAC.

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

Theorem A3. Interval Algebra Consistency is NP-Complete.

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



The idea is that in any consistent scenario, every vertex $v \in V$ is forced to be equal to only one of the k special vertices. This is the mapping χ of vertices to colors. We also must ensure that any two vertices adjacent in G do not map to the same value. Hence, for each $(v, w) \in E$ we create an edge in E_L and label it as follows.

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

Thus, an efficient algorithm for IAC would imply an efficient algorithm for Graph Coloring. Hence, the IAC problem is NP-Complete. But the IAC is a special case of the consistent scenario and minimal labeling problems. That is, finding a consistent scenario or finding a minimal labeling also answers the decision problem: does a consistent scenario exist. Hence, the consistent scenario and minimal labeling problems are also NP-Complete.

References

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

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 H. Kautz. 1985. A Model of Naive Temporal Reasoning. In *Formal Theories of the Commonsense World*, J. Hobbs and R. Moore (eds.), Ablex, 251-268.

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

Almeida, M. J. 1987. Reasoning about the Temporal Structure of Narrative. Ph.D. thesis available as State University of New York at Buffalo Technical Report 87-10, Buffalo, N.Y.

Baykan, C. A., and M. S. Fox. 1987. An Investigation of Opportunistic Constraint Satisfaction in Space Planning. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1035-1038.

Booth, K. S., and G. S. Leuker. 1976. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-tree Algorithms. *J. Comput. Syst. Sci.* **13**, 335-379.

Carré, B. 1979. *Graphs and Networks*. Clarendon Press.

Chvátal, V. 1983. *Linear Programming*. W. H. Freeman.

Dean, T., and D. V. McDermott. 1987. Temporal Data Base Management. *Artificial Intelligence* **32**, 1-55.

Dechter, R., and I. Meiri. 1989. Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Mich., 271-277.

Dechter, R., and J. Pearl. 1988. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence* **34**, 1-38.

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.

Deo, N., and C. Pang. 1984. Shortest-path algorithms: Taxonomy and Annotation. *Networks* **14**, 275-323.

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

Earl, C. F., and L. J. March. 1979. Architectural Applications of Graph Theory. In *Applications of Graph Theory*, R. J. Wilson and L. W. Beineke (eds.), Academic Press.

Eastman, C. M. 1970. Representations for Space Planning. *Comm. ACM* **3**, 242-250.

Eastman, C. M. 1973. Automated Space Planning. *Artificial Intelligence* **4**, 41-64.

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.

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

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

Güther, S. 1984. Zur Repräsentation Temporaler Beziehungen in SRL. KIT Report 21, Fachbereich Informatik, Technische Universität, Berlin. Cited in: A. Schmiedel, 1988a.

Haralick, R. M., and G. L. Elliott. 1980. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence* **14**, 263-313.

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

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

Kautz, H. A. 1987. A Formal Theory of Plan Recognition. Ph.D. thesis available as University of Rochester Technical Report 215, Rochester, N.Y.

Knuth, D. E. 1973. *The Art of Computer Programming. Volume 1 / Fundamental Algorithms*. Addison-Wesley, 258-265.

Koomen, J. A. 1989. Localizing Temporal Constraint Propagation. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ont., 198-202.

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

Ladkin, P. B. 1989. Metric Constraint Satisfaction with Intervals. Technical Report TR-89-038, International Computer Science Institute, Berkeley, Calif.

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. 1987. Constraint Satisfaction. In *Encyclopedia of Artificial Intelligence*, S. C. Shapiro (ed.), John Wiley & Sons.

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.

Malik, J., and T. O. Binford. 1983. Reasoning in Time and Space. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, W. Germany, 343-345.

Miller, S. A., and L. K. Schubert. 1988. Time Revisited. *Proceedings of the Seventh Canadian Conference on Artificial Intelligence*, Edmonton, Alta., 39-45.

Moller, F. 1985. A Survey of Systolic Systems for Solving the Algebraic Path Problem. Department of Computer Science Research Report CS-85-22, University of Waterloo.

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

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

Nökel, K. 1989. Temporal Matching: Recognizing Dynamic Situations from Discrete Measurements. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Mich., 1255-1260.

Nudel, B. 1983. Consistent-Labeling Problems and their Algorithms: Expected-Complexities and Theory-Based Heuristics. *Artificial Intelligence* **21**, 135-178.

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

Pelavin, R., and J. F. Allen. 1986. A Formal Logic of Plans in Temporally Rich Domains. *Proceedings of the IEEE* **74**, 1364-1382.

Roberts, F. S. 1976. *Discrete Mathematical Models, with Applications to Social, Biological, and Environmental Problems.* Prentice-Hall.

Schmiedel, A. 1988a. Temporal Constraint Networks. KIT Report 69, Fachbereich Informatik, Technische Universität, Berlin.

Schmiedel, A. 1988b. A Temporal Constraint Handler for the BACK System. KIT Report 70, Fachbereich Informatik, Technische Universität, Berlin.

Seidel, R. 1981. A New Method for Solving Constraint Satisfaction Problems. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, B.C., 338-342.

Seidel, R. 1983. On the Complexity of Achieving k-Consistency. Department of Computer Science Technical Report 83-4, University of British Columbia. Cited in: A. K. Mackworth, 1987.

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

Song, F. 1990. A Processing Model for Temporal Analysis with Applications in Plan Recognition. Ph.D. thesis, University of Waterloo, Waterloo, Ont. In preparation.

Tarjan, R. E. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* **1**, 146-160.

Tarjan, R. E. 1981a. A Unified Approach to Path Problems. *J. ACM* **28**, 577-593.

Tarjan, R. E. 1981b. Fast Algorithms for Solving Path Problems. *J. ACM* **28**, 594-614.

Topaloglou, T., and M. Koubarakis. 1989. Implementation of Telos: Problems and Solutions. Knowledge Representation and Reasoning Technical Report KRR-TR-89-8, Department of Computer Science, University of Toronto.

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

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

Valdés-Pérez, R. E. 1987. The Satisfiability of Temporal Constraint Networks. *Proceedings of the Sixth National Conference on Artificial Intelligence*, 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,* Boston, Mass., 728-734.

van Beek, P., and R. Cohen. 1990. Exact and Approximate Reasoning about Temporal Relations. *Computational Intelligence.* In press.

van Benthem, J. F. A. K. 1983. *The Logic of Time*. Reidel.

Vilain, M. 1982. A System for Reasoning About Time. *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, Penn., 197-201.

Vilain, M., and H. Kautz. 1986. Constraint Propagation Algorithms for Temporal Reasoning. *Proceedings of the Fifth National Conference on Artificial Intelligence*, 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.

Wirth, N. 1976. *Algorithms + Data Structures = Programs*. Prentice-Hall.

Weld, D. S., and J. de Kleer. 1989. Introduction to Chapter 4, History-Based Simulation and Temporal Reasoning. In *Readings in Qualitative Reasoning about Physical Systems*, Morgan-Kaufman, 351-352.