Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

TEMPORAL REASONING: A CONSTRAINT-BASED APPROACH

I. Meiri

January 1992
CSD-920019

Temporal Reasoning:
A Constraint-Based Approach

Itay Meiri

January 1992

Technical Report R-173
Cognitive Systems Laboratory
Department of Computer Science
University of California
Los Angeles, CA 90024

This report reproduces a dissertation submitted to UCLA in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science. This work was supported in part by grants from the Air Force Office of Scientific Research, AFOSR 900136, and the National Science Foundation, IRI 8815522.

Abstract

This thesis introduces a new approach to temporal reasoning which increases both the expressive power and the flexibility of handling temporal information. It also offers a unifying conceptualization of existing approaches and invites the importation of solution techniques from several disciplines.

Temporal reasoning is viewed as a constraint satisfaction problem in which variables are forced to comply with a set of constraints. The variables are temporal objects such as intervals (representing time periods during which events occur or propositions hold) and time points (representing beginnings and ends of intervals), and the constraints specify the relative location of these objects along the time line. Unlike existing approaches, our proposal permits the processing of both qualitative and quantitative constraints.

First, a model called *temporal constraint networks* is introduced, which facilitates the processing of quantitative information such as duration and timing of events. In this model, variables represent time points, and the constraints refer to absolute timing or time differences between events. Second, a framework called *general temporal networks* is developed, combining quantitative information about duration and timing with qualitative relations about precedence and occurrences of events. This framework offers the unique flexibility of treating both points and intervals as the primitive objects in the language, thereby generalizing and unifying the temporal-constraint-networks model and common approaches to temporal reasoning such as Allen's interval algebra and Vilain and Kautz's point algebra.

We present constraint-based algorithms for performing the following reasoning tasks: finding all feasible times that a given event can occur, finding all possible relationships between two given events, and generating one or more scenarios consistent with the information provided. Several new classes of tractable temporal problems are identified and characterized, involving special formats of qualitative and quantitative expressions. Such problems can be solved in polynomial time using local relaxation algorithms.

To my parents Haim and Shoshana Meiri

Acknowledgments

First and foremost, I would like to thank my advisors Rina Dechter and Judea Pearl for teaching me all aspects of good research. Rina introduced me to constraint processing, interested me in temporal constraints, and has been a constant source of fresh ideas. Judea inspired me (and everybody else around) in producing quality research. He always showed interest in my work and spent many hours teaching me how to present my ideas more clearly. Rina and Judea gave me the freedom to pursue my own areas of interest while helping me in every possible way—both academically and non-academically. For their excellent guidance and for their encouragement throughout my years at UCLA, I will forever be grateful.

I also wish to acknowledge the other members of my committee—Stott Parker, James MacQueen, Edward Keenan, and, especially, Rich Korf—for their valuable comments. Thanks also go to Michelle Bonnice for editing on the final draft. I have benefited from discussions with researchers outside UCLA; in particular, I wish to thank Henry Kautz, Peter Ladkin, and Peter Van Beek.

Special thanks go to my colleagues at the Cognitive Systems Laboratory, who have made my experience at UCLA enjoyable. In particular, I thank Rachel Ben-Eliyahu for being good company and the first to support me in every major decision I have made and for always offering help and good advice; Dan Geiger for encouraging me during my first years at UCLA and for long discussions on the nature of science; and my officemate, Moisés Goldszmidt, for putting up with me all these years, for sharing the ups and downs of graduate school, and for many productive and unproductive discussions.

Most importantly, I would like to thank my family: Dafna for being my roommate and family away from home; Orna for her unconditional support and care; and, especially, my parents Haim and Shoshana, who always valued the education of their children above everything, for having confidence in my abilities and for giving me the encouragement and support necessary for me to pursue my own goals. I would not be where I am today without them.

Table of Contents

List of Figures

List of Tables

For every thing its time
And a time for every purpose
Under the heaven.

—King Solomon: Ecclesiastes, *iii*, 1

CHAPTER 1

Introduction

Representing and reasoning about time plays an important role in artificial intelligence (AI). Almost any area within AI—common-sense reasoning [31, 57], natural language understanding [3, 33], plan recognition[60], scheduling[55], planning [47, 70, 14, 16], and qualitative reasoning [74], to name a few—involves some sort of reasoning about time. Philosophers, psychologists, linguists, and, of course, computer scientists have struggled for many years with the difficulties inherent in representing time. Several theories and formalisms have been proposed, yet many questions remain unsolved: Is time discrete or continuous? Should we use a linear or a branching time model? Are the primitive temporal objects points or intervals?[1] All these dilemmas seem to indicate that as long as the nature of time is not fully understood, we cannot devise a general system for reasoning about time.

Nevertheless, some compromise must be made if, for practical reasons, one decides to reason about time. Instead of pursuing a general theory of time, this thesis offers a formalism for temporal reasoning that can be incorporated into current problem-solving programs. The idea, which can be traced back to Kahn and Gorry [33] and Allen [2], is to build a system that will handle only the temporal reasoning tasks of a problem solver. It will have no understanding, whatsoever, of the domain in question; this knowledge will be handled exclusively by the problem solver. The temporal reasoning system will be responsible only for the temporal aspects involved in the problem solving. It will be provided with a set of temporal statements expressed in some predetermined language and, based on this input, will deduce new temporal statements and answer a variety of queries concerning the input statements. Designing such a system requires a simplistic, in a sense, representation of time, thus avoiding most of the dilemmas mentioned above.

[1]See Shoham's book [57] and Dean's article [15] for an AI perspective and VanBenthem's book [69] for a philosophical perspective on these issues.

We envision a temporal reasoning system to consist of a temporal knowledge base, a routine to check its consistency, a query-answering mechanism, and an inference mechanism capable of discovering new information. The primitive entities in the knowledge base are *propositions*, such as "I was driving a car" or "the book was lying on the table," with which we mentally associate temporal intervals; each interval represents the time period during which the corresponding proposition holds. For example, the first proposition may be embedded in the sentence "I was driving a car last night" and the second in "The book was on the table when I left the room." The temporal information might be relative (e.g., "P occurred before Q") or metric (e.g., "P had started at least 3 hours before Q was terminated"). To express less specific information, disjunctive sentences may also be needed (e.g., "You can come in before or after lunch hour"). We also allow references to absolute time (e.g., 4:00 p.m.) and to the duration of propositions (e.g., "P lasted at least two hours"). Given temporal information of this kind, we wish to derive answers to queries such as "Is it possible that a proposition P holds at time t?", "What are the possible times at which a proposition P holds?", and "What are the possible temporal relationships between two propositions P and Q?".

As an example, consider a typical temporal reasoning problem. We are given the following information.

Example 1.1 *John and Fred work for a company that has both local and main offices in Los Angeles. They usually work at the local office, in which case it takes John less than 20 minutes and Fred 15–20 minutes to get to work. Twice a week John works at the main office, in which case his commute to work takes at least 60 minutes. Today John left home between 7:05–7:10 a.m., and Fred arrived at work between 7:50–7:55 a.m. We also know that Fred and John met at a traffic light on their way to work.*

We wish to represent and reason about such knowledge. We wish to answer queries such as "Is the information in this story consistent?", "Who was the first to arrive at work?", and "What are the possible times at which John arrived at work?".

Several formalisms for expressing and reasoning about temporal knowledge of this kind have been proposed, most notably Allen's interval algebra [2], Vilain and Kautz's point algebra [72], linear inequalities [46, 64], and Dean and

McDermott's time map [17, 15]. Only two formalisms—Allen's interval algebra [2] and Vilain and Kautz's point algebra [72]—were mathematically formulated and were described in sufficient detail to allow rigorous mathematical analysis of the computational complexity involved. These two formalisms view temporal reasoning as a *constraint satisfaction problem (CSP).*

We first describe the basics of constraint processing and then show how temporal reasoning can be expressed as a CSP.

1.1 Constraint Satisfaction Problems

Constraint specification is a convenient form for expressing knowledge in a declarative way; more precisely, in terms of a set of constraints on some entities. It allows the user to focus on local relationships among entities in the domain, without specifying the actual representation of the constraints and the method for their satisfaction. Let us briefly review the basic concepts from the CSP literature.[2]

For simplicity we shall consider the case of *binary* constraints. A binary CSP involves a set of variables $\{X_1, \ldots, X_n\}$ representing entities in the domain at hand and a set of constraints on pairs of variables. The *domain* of a variable X_i, denoted by D_i, defines the set of values X_i may assume. A *binary constraint* C_{ij} on variables X_i and X_j, is a relation, namely a subset of the Cartesian product of their domains (i.e., $C_{ij} \subseteq D_i \times D_j$) that specifies the permitted pairs of values for X_i and X_j. A constraint can be specified extensionally, as a list of all allowed (or disallowed) pairs, or intensionally, for example $X_i < X_j$ or $X_i + X_j = 3$. Clearly, whenever the domains are infinite, only the latter representation can be used. A tuple that satisfies all the constraints is called a *solution*. The problem is *consistent* if it has a solution.

To demonstrate these concepts, let us examine two classical CSPs.

Example 1.2 The **n-queens** problem. The task is to place n queens on an $n \times n$ chess board such that no two queens can attack each other (i.e., they are not on the same row, column, or diagonal). We create n variables $X_1, \ldots, X_n$, where variable X_i represents the queen in row i (note that each queen must be

[2]For a more detailed description of CSPs, see the survey articles by Mackworth [44] and Dechter [19].

placed in a different row). The value assigned to a variable X_i stands for the column number of the queen in row i. Thus, the domain D_i is finite, consisting of the values $\{1, \ldots, n\}$. A constraint C_{ij}, between variables X_i and X_j, forbids illegal board configurations, in which the queens in rows i and j attack each other. For example, the constraint between the queens in rows 1 and 2 (specified by a list of permitted pairs) is

$$C_{12} = \{(u, v) | 1 \leq u, v \leq n, |u - v| > 1\}.$$

Each pair $(u, v) \in C_{12}$ refers to a board configuration in which queen 1 and queen 2 are located at squares $(1, u)$ and $(2, v)$, respectively. It can be easily seen that for $n = 2$ or $n = 3$ the problem is inconsistent. For $n = 4$ the problem becomes consistent: the tuple $X = (2, 4, 1, 3)$ is one solution. □

Example 1.3 The **k-colorability** problem. The task is to color the vertices of a given graph $G = (V, E)$ using k colors, such that no two adjacent vertices are assigned the same color. Let $|V| = n$. We create n variables $X_1, \ldots, X_n$, where variable X_i represents node i. Each domain consists of the values $\{1, \ldots, k\}$, representing the k colors. With each edge $(i, j) \in E$ we associate a constraint C_{ij} that forbids an assignment of identical colors to nodes i and j. Such a constraint can be represented, intensionally, as the $\neq$ relation or, extensionally, by a list of allowed (or disallowed) tuples. If we choose an allowed tuples representation, then all constraints are specified as

$$C_{ij} = \{(u, v) | 1 \leq u, v \leq k, u \neq v\}.$$

□

Extensive research has been carried out on solving constraint satisfaction problems [50, 43, 28, 25, 26, 32, 52, 21, 22, 18]. Although the bulk of these works has focused on discrete and finite domains, many of the solution techniques can be applied successfully to the temporal domain, as will be shown in the sequel.

1.2 Temporal Reasoning as a Constraint Satisfaction Problem

How can temporal reasoning be viewed as a CSP? First, we need to identify the entities in our temporal domain. We shall consider two types of temporal objects: points and intervals. Intervals correspond to time periods during which events occur or propositions hold, and points represent beginning and ending points of some events, as well as neutral points of time. These objects will be the variables in our CSP. Temporal statements will be treated as constraints on the location of these objects along the time line. There are two types of constraints: qualitative and quantitative. *Qualitative* constraints specify the relative position of paired objects, and *quantitative* constraints place absolute bounds or restrict the temporal distance between points.

Illustration Let us formulate Example 1.1 as a CSP. We have two meaningful events: "John was going to work" and "Fred was going to work." These events are associated with intervals $J = [P_1, P_2]$ and $F = [P_3, P_4]$, respectively. The extreme points of these intervals, $P_1, \ldots, P_4$, represent the times at which Fred and John left home and arrived at work. We also introduce a neutral point P_0 to represent the "beginning of time" in our story. One possible choice for P_0 is 7:00 a.m. The variables in our CSP will be the intervals J and F and the points $P_0, \ldots, P_4$. The fact that John and Fred met at a traffic light is translated into a qualitative constraint that forces intervals J and F to overlap. The information on Fred's commuting time translates into a quantitative constraint that restricts the length of interval F, namely the distance between P_3 and P_4. $\Box$

Allen's interval algebra [2] and Vilain and Kautz's point algebra [72] can be regarded as qualitative constraint-based approaches to temporal reasoning, as their representation languages allow only qualitative statements. In Allen's algebra, the temporal objects are intervals, and constraints are specified by the relative location of paired intervals. For example, we can specify that intervals I and J intersect, that they are disjoint, or that I occurred before J. In Vilain and Kautz's point algebra, on the other hand, the temporal objects are points, and constraints are specified by the relative location of paired points. For example, P occurred at the same time as Q or P and Q did not occur at the same time, where P and Q are time points.

1.3 Contributions and Outline of the Thesis

The existing formalisms for temporal reasoning—Allen's interval algebra and Vilain and Kautz's point algebra—facilitate reasoning about qualitative relations, but they cannot handle many forms of quantitative knowledge. This thesis introduces a new approach to temporal reasoning which increases both the expressive power and the flexibility of handling temporal information. The approach comprises two new formalisms. The first, called *temporal constraint satisfaction problem (TCSP)*, provides a framework for dealing with quantitative information, such as duration and timing of events. The second, *general temporal networks*, subsumes the interval algebra, the point algebra, and the TCSP model, thus providing a general unified framework for temporal reasoning that is capable of handling both qualitative and quantitative information.

For each of these formalisms, we present algorithms for performing these reasoning tasks: finding all feasible times that a given event can occur, finding all possible relationships between two given events, and generating one or more scenarios consistent with the information provided.

The proposed formalisms use the constraint satisfaction paradigm, thus encouraging the transference of algorithms and theoretical results developed for general CSPs to the temporal domain. Specifically,

1. Reasoning tasks can be solved by decomposition into *singleton labelings*, each solvable in polynomial time. This decomposition scheme can be improved by traditional constraint satisfaction techniques such as variants of backtrack search.

2. The input can be effectively encoded in a *minimal network* representation, which provides answers to many queries.

3. Local consistency algorithms, such as *arc consistency* and *path consistency*, can be used in preprocessing the input network to improve search efficiency or in computing an approximation to the minimal network.

4. Although most temporal reasoning tasks are, in general, intractable, by using constraint satisfaction techniques, we were able to identify several classes of tractable problems that are solvable in polynomial time.

Chapter 2 reviews the qualitative approaches to temporal reasoning—Allen's interval algebra and Vilain and Kautz's point algebra.

Chapter 3 presents the TCSP model (also called *metric networks* in the sequel), which provides a framework for processing quantitative, metric constraints. In this framework, variables represent time points, and temporal information is represented by a set of unary and binary constraints. Unary constraints place absolute bounds on points, whereas binary constraints restrict the *temporal distance* between points. Each constraint is specified by a set of permitted intervals.

We distinguish between *simple temporal problems (STPs)* and general temporal problems, the former admitting at most one interval constraint on any pair of time points. We show that the STP, which subsumes the major part of Vilain and Kautz's point algebra, can be solved in polynomial time. For general TCSPs, we present a decomposition scheme that performs the three reasoning tasks considered and introduce a variety of techniques for improving its efficiency. We also study the applicability of path consistency algorithms in preprocessing of temporal problems, demonstrate the termination of these algorithms, and bound their complexities.

None of the formalisms described so far can handle all forms of temporal knowledge. The qualitative approaches, Allen's interval algebra and Vilain and Kautz's point algebra, have difficulties in representing and reasoning about metric, numerical information, while the quantitative approach discussed in Chapter 3 exhibits limited expressiveness when it comes to qualitative information. Chapter 4 offers a solution to this problem: a general, constraint-based computational model for temporal reasoning, called general temporal networks, that is capable of handling both qualitative and quantitative information. In this model, variables represent both points and intervals (as opposed to the previous formalisms, where one has to commit to a single type of object), and constraints may be either metric (between points) or qualitative, disjunctive relations (between temporal objects). The unique feature of this framework is that it allows the representation and processing of all types of qualitative constraints considered in the literature to date, as well as the new, metric constraints of the TCSP formalism.

To solve reasoning tasks in this model, we use constraint satisfaction techniques, such as decomposition into singleton labelings and path consistency. We also identify two new classes of tractable problems involving both qualitative and quantitative constraints. The first comprises *augmented qualitative networks*—composed of qualitative constraints between points and quantitative domain constraints—which can be solved using arc and path consistency. The second class comprises networks for which path consistency algorithms are exact.

Chapter 5 provides summary and concluding remarks, as well as directions for future research.

CHAPTER 2

Qualitative Networks

2.1 Allen's Interval Algebra

Allen [2] formulated temporal knowledge in terms of qualitative statements regarding the relative locations of paired intervals. Consider a pair of intervals I and J. There are seven possible relations that can hold between them: *before, meets, overlaps, starts, during, finishes*, and *equal*, as depicted in Figure 2.1. Each one of these relations is associated with an inverse relation; for example, the inverse of the relation *before* is the relation *after*, because I *before* J is equivalent to J *after* I. The inverse relations are shown in Table 2.1. Overall, we have 13 possible *basic relations* that can exist between a pair of intervals; they will be represented by the set $\{b, m, o, s, d, f, bi, mi, oi, si, di, fi, =\}$.

A subset of basic relations corresponds to an ambiguous, disjunctive relationship between intervals. If the relative location of intervals I and J is specified by a relation set $\{r_1, \ldots, r_k\}$ (written as $I\ \{r_1, \ldots, r_k\}\ J$), then the following disjunction holds:

$$(I\ r_1\ J) \vee \cdots \vee (I\ r_k\ J).$$

Relation	**Symbol**	**Inverse**
I before J	b	bi
I meets J	m	mi
I overlaps J	o	oi
I starts J	s	si
I during J	d	di
I finishes J	f	fi
I equal J	$=$	$=$

Table 2.1: The basic relations between a pair of intervals and their inverses.

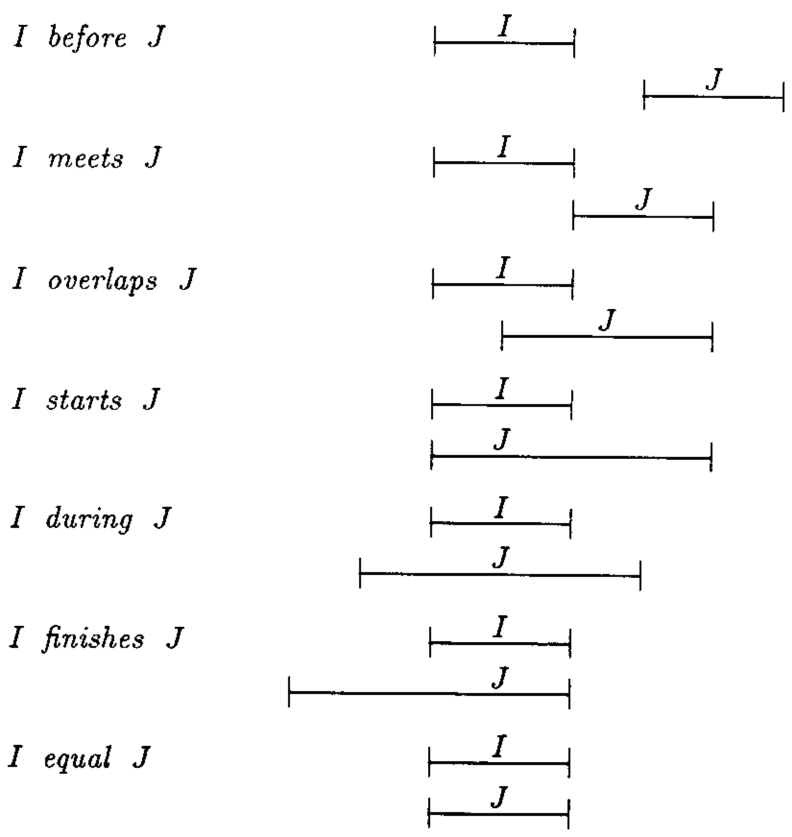

Figure 2.1: The basic relations between a pair of intervals.

For example, the relationship $I\ \{s, si, d, di, f, fi, o, oi, =\}\ J$ expresses the fact that intervals I and J are not disjoint. It excludes the basic relations, *before, after, meets*, and *met by*, between I and J (see Figure 2.2), thus forbidding basic relations whereby I and J are disjoint.

Allen suggested using an *interval algebra (IA)* to represent and reason about temporal knowledge. The elements of the algebra are all 2^{13} subsets of the basic relations $\{b, m, s, d, f, o, bi, mi, si, di, fi, oi, =\}$. On these elements, two binary operations, *intersection* and *composition* (to be described later in this section), are defined. To represent a given body of knowledge in the IA formalism, we simply translate the temporal statements into IA relationships between event intervals.

Example 2.1 (Allen [2]) We are given the following information:

John was not in the room when I touched the switch to turn on the

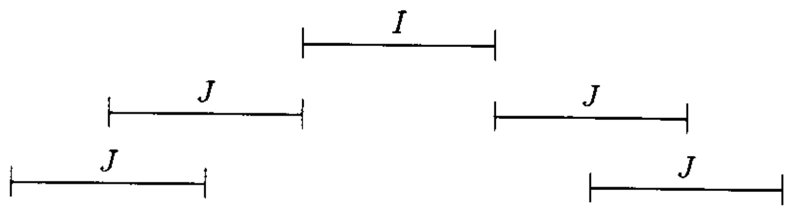

Figure 2.2: The disjointness relationship.

light, but John was in the room later when the light went out.

Let S be the time of touching the switch, L be the time the light was on, and R be the time that John was in the room. The above information is translated into a set of IA relations between S, L, and R:

1. S *overlaps* or *meets* L:
$$S\ \{o, m\}\ L.$$
2. S is *before, meets*, is *met by*, or *after* R:
$$S\ \{b, m, mi, a\}\ R.$$
3. L *overlaps, starts*, or is *during* R:
$$S\ \{o, s, d\}\ R.$$

□

Having represented the given knowledge within the IA framework, we are now interested in solving several reasoning tasks:

1. Determine whether the given information is *consistent*, namely, whether it is possible to arrange the intervals along the time line according to the given information.

2. If the information is consistent, find one or some arrangements of the intervals along the time line, each corresponding to a possible *scenario*, that is consistent with the given information.

3. Determine for a given basic relation r and a given pair of intervals I and J, whether there exists a consistent scenario in which the relationship between I and J is r.

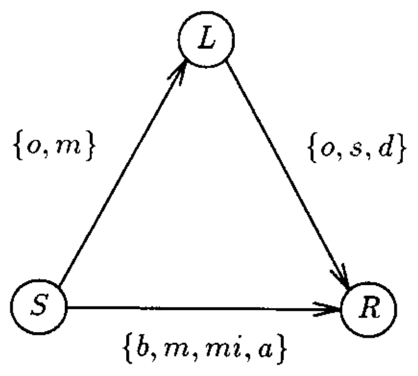

Figure 2.3: The constraint graph of Example 2.1.

4. Find all the possible relations between a given pair of intervals I and J.

Temporal knowledge specified in Allen's IA can be expressed naturally as a CSP. An *interval algebra network (IA network)* involves a set of variables $\{X_1, \ldots, X_n\}$, where each variable represents a temporal interval. The domain of each variable is the set of ordered pairs of real numbers (i.e., $D_i = \{(a,b)|a,b \in \Re, a < b\}$), representing the beginning and ending points of the corresponding interval. Constraints are given as IA elements.

An IA network is associated with a *constraint graph*, where node i represents variable X_i and an edge between nodes i and j represents a *direct constraint* C_{ij} between variables X_i and X_j. Each edge is labeled by the corresponding IA relation set. The lack of an edge between nodes i and j stands for a *universal constraint*, denoted by ?, which allows all basic relations. The constraint graph representing Example 2.1 is depicted in Figure 2.3.

As in general CSPs, a tuple that satisfies all the constraints is called a *solution*. In IA networks, each solution corresponds to a feasible scenario. We say that the network is *consistent* if it has a solution. One solution to the network of Figure 2.3 is $\{S = (1,2), L = (2,3), R = (2,4)\}$, which represents the scenario shown in Figure 2.4.

We define a partial order $\subseteq$ among IA constraints. A constraint C' is *tighter* than constraint C'' (or conversely C'' is more *relaxed* than C'), denoted by $C' \subseteq C''$, if $C' \subseteq C''$ when viewing IA relations as sets. This partial order can be extended to networks having the same variable set. A network N' is tighter than network N'', if the partial order $\subseteq$ is satisfied for all the corresponding

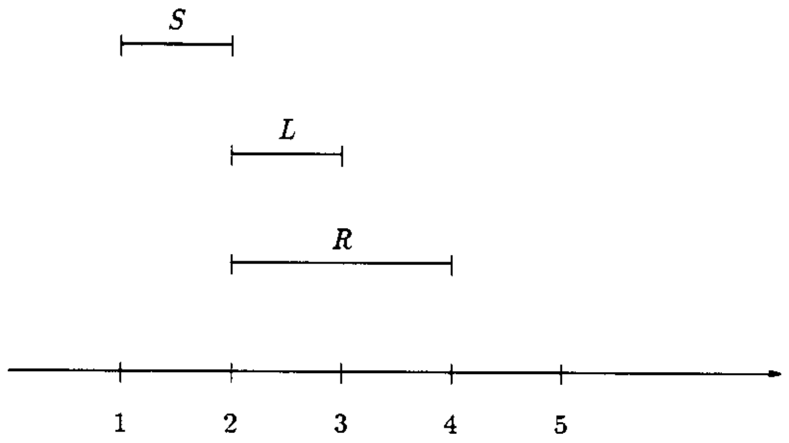

Figure 2.4: A consistent scenario of Example 2.1.

constraints.

Two networks are *equivalent* if they possess the same solution set. A network may have many equivalent representations; in particular, there is a *unique* equivalent network M, which is minimal with respect to $\subseteq$, called the *minimal network*.

We may define the minimal network in an equivalent, somewhat more natural way. Consider an arbitrary constraint C_{ij}, specified by an IA relation set $\{r_i, \ldots, r_k\}$. Now consider the set of all solutions, that is, the set of all possible scenarios. We shall say that a basic relation r_l $(1 \leq l \leq k)$ is a *consistent label* of an edge $i \longrightarrow j$ if there exists a scenario in which the relationship between intervals X_i and X_j is r_l. The *minimal label* (or *minimal constraint*) of an edge $i \longrightarrow j$ is the set of all its consistent labels. The minimal network is the (unique) network labeled by the minimal labels of all edges. The minimal network of Example 2.1 is shown in Figure 2.5. It can easily be verified that all its labels are consistent.

The minimal network provides a more explicit representation of the given knowledge and, therefore, it is useful in answering many types of queries.

Using the CSP terminology, the interesting reasoning tasks for IA networks are deciding consistency, finding one or more solutions, computing the minimal labels, and computing the full minimal network. It turns out that all these tasks are intractable; even the simplest task of deciding consistency is NP-complete [72]. Thus, it is unlikely that we can find polynomial-time algorithms for their solution. As a result, we settle for exponential, exhaustive search algorithms such

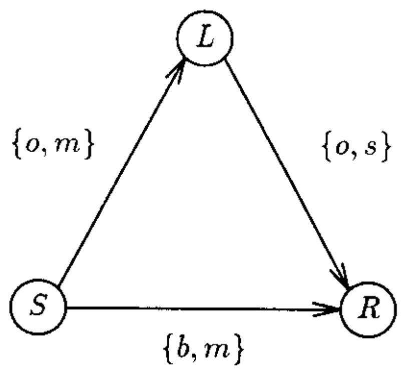

Figure 2.5: The minimal network of Example 2.1.

as backtracking (for example, the algorithm given in [65]).

Another approach is to use approximation algorithms such as *path consistency* (to be discussed in the next section), which run in polynomial time. These algorithms improve the representation of the network by removing basic relations, which are not contained in minimal labels, from some edges. In some cases, as we shall see later, they may compute the minimal labels of some edges or even the full minimal network.

2.2 Path Consistency in Interval Algebra Networks

The notion of *local consistency* [50, 43, 25] plays an important role in constraint processing. The idea is to consider small subnetworks and to make them locally consistent by ignoring the rest of the network. We say that a network is *k-consistent* if any consistent assignment of values to any subset S of $k-1$ variables (i.e., an assignment that satisfies all the constraints applicable to S) is extensible to any variable $X \notin S$ (i.e., the extended assignment satisfies all the constraints applicable to $S \cup X$) [25]. Enforcing k-consistency improves the representation of the network, and for small k's it is usually worthwhile, since its complexity is bounded by $O(n^k)$.

An important special case is *3-consistency*, also known as *path consistency* [50, 43]. In this chapter, we shall use an equivalent, more convenient definition of path consistency. This definition uses two binary operations on constraints: *intersection* and *composition*.

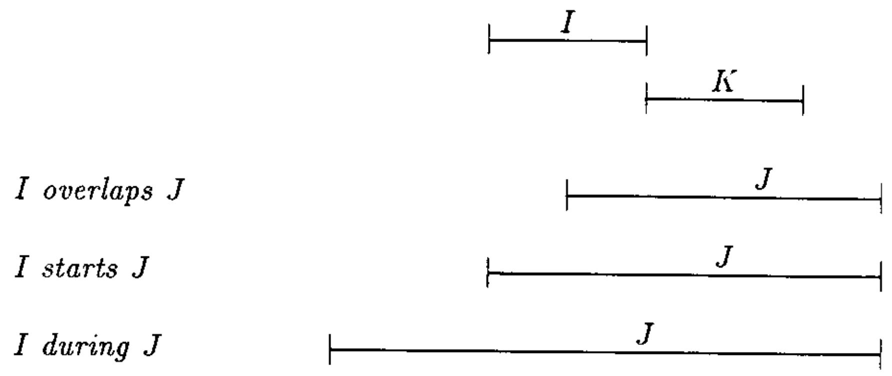

Figure 2.6: Composition of basic relations.

The *intersection* of two IA relations R' and R'', denoted by $R' \oplus R''$, is the set-theoretic intersection $R' \cap R''$.

The *composition* of two IA relations, R' between intervals I and K and R'' between intervals K and J, is a new relation between intervals I and J, induced by R' and R''. The composition of two basic relations r' and r'' is defined by a *transitivity table*[2], a portion of which is shown in Table 2.2.[1] For example, the basic relations I *meets* K and K is *during* J induce a new (composite) relation on I and J : I *overlaps*, is *during*, or *starts* J, depending on the location of J's starting point (see Figure 2.6). It follows that the entry for $m \otimes d$ in the transitivity table is the set $\{o, d, s\}$. The composition of two *composite* relations R' and R'', denoted by $R' \otimes R''$, is the composition of the constituent basic relations, namely

$$R' \otimes R'' = \{r' \otimes r'' | r' \in R', r'' \in R''\}.$$

Definition 2.2 We say that a given network is *path consistent* if for every three variables i, j, k,

$$C_{ij} \subseteq C_{ik} \otimes C_{kj}.$$

Any network can be converted into an equivalent path-consistent form by repeatedly applying the relaxation operation

$$C_{ij} \leftarrow C_{ij} \oplus C_{ik} \otimes C_{kj} \qquad (2.1)$$

[1]The full table can be found in Allen's paper [2].

	b	**s**	**d**	**o**	**m**
b	b	b	b o m d s	b	b
s	b	s	d	b o m	b
d	b	d	d	b o m d s	b
o	b	o	o d s	b o m	b
m	b	m	o d s	b	b

Table 2.2: Composition in the interval algebra.

until either a fixed point is reached or some constraint becomes empty, indicating inconsistency. Using this method, path consistency can be achieved in $O(n^3)$ time [2, 72].

Illustration Consider the network of Figure 2.3. Let us enforce path consistency by repeatedly applying Equation (2.1).

- *Step 1*: Apply
$$C_{SR} \leftarrow C_{SR} \oplus C_{SL} \otimes C_{LR}.$$
Composing the constraints C_{SL} and C_{LR} induces a new constraint between S and R:
$$C'_{SR} = \{o, m\} \otimes \{o, s, d\} = \{b, o, m, d, s\}.$$
Then, intersecting C'_{SR} with the original constraint $C_{SR} = \{b, m, mi, a\}$ yields a new constraint between S and R:
$$C_{SR} \leftarrow \{b, m\}.$$

- *Step 2*: Apply
$$C_{LR} \leftarrow C_{LR} \oplus C_{LS} \otimes C_{SR}.$$
This results in a new constraint between L and R:
$$C_{LR} \leftarrow \{o, s\}.$$

- *Step 3*: Further applications of Equation (2.1) will not change the network; thus, we have reached a fixed point.

relation	**symbol**
$\emptyset$	$\emptyset$
$\{<\}$	$<$
$\{=\}$	$=$
$\{>\}$	$>$
$\{<,=\}$	$\leq$
$\{>,=\}$	$\geq$
$\{<,>\}$	$\neq$
$\{<,=,>\}$	?

Table 2.3: The point algebra elements.

The resulting network is the minimal network representation (Figure 2.5). It can be verified that this network is indeed path consistent.[2] $\square$

In some cases, path-consistency enforcing algorithms (or path consistency, for short) are *exact*, that is, they compute the minimal network (e.g., Example 2.1). In general, however, path consistency is not guaranteed to compute the minimal network or even to detect inconsistency (a counterexample is given in [72]). Much work has been done on characterizing special cases for which local consistency algorithms are exact. In the next section, we present some classes of IA networks for which 3- or 4-consistency indeed computes the minimal network.

2.3 Vilain and Kautz's Point Algebra

Because of the computational limitations of the IA, Vilain and Kautz [72] suggested an alternative model, a *point algebra (PA)*, in which the information is expressed by means of constraints on points. There are three possible basic relations that can hold between a pair of points P and Q: $P < Q$, $P = Q$, and $P > Q$. The elements of the PA are, therefore, all 2^3 subsets of the basic relations $\{<, =, >\}$. Sometimes it will be more convenient to use the shorthand notation shown in Table 2.3.

A *point algebra network (PA network)* involves a set of variables $\{X_1, \ldots, X_n\}$, where each variable represents a time point. The domain of each variable is the

[2] It can easily be shown that the minimal network is always path consistent.

	$<$	$=$	$>$
$<$	$<$	$<$	?
$=$	$<$	$=$	$>$
$>$	?	$=$	$>$

Table 2.4: Composition in the point algebra.

set of real numbers $\Re$, standing for the set of times the variable may assume. The constraints are given as PA elements.

Most of the IA terminology (consistency, minimal network, path consistency, etc.) can be used in the same manner for the PA. The only difference is in the definition of the composition operation, which is now defined by the transitivity table of Figure 2.4 [72].

To represent knowledge in the PA, we translate the input statements into PA relationships between points. The points may be the starting and ending points of event intervals or some neutral points of time.

Example 2.3 Suppose we are given the following relationship between intervals I and J:

$$I \; \{s, d, f, =\} \; J, \tag{2.2}$$

namely, I *starts*, is *during*, *finishes*, or is *equal* to J. Let A and B be the starting and ending points, respectively, of interval I (i.e., $I = [A, B]$), and C and D be the starting and ending points, respectively, of interval J (i.e., $J = [C, D]$). The relationship in Equation (2.2) can be expressed by the PA relations

$$A < B, \; C < D, \; A \leq D, \; A \geq C, \; B \leq D, \; B \geq C. \tag{2.3}$$

Note that $\leq$ and $\geq$ are shorthand notation for the PA elements $\{<, =\}$ and $\{>, =\}$, respectively. It can be verified that Equation (2.3) is equivalent to Equation (2.2). □

In general, any PA network can be expressed as an IA network. The opposite is not true—the PA can handle only a subset of the IA networks; there are problems that can be expressed by binary relations between intervals but not by binary relations between points.

Example 2.4 (Vilain and Kautz [72]) Consider the IA relation

$$I \ \{b, a\} \ J, \tag{2.4}$$

namely I is *before* or *after* J, where intervals I and J are given by $I = [A, B]$ and $J = [C, D]$. This relationship cannot be encoded by binary PA relations on points; it requires the 4-ary constraint

$$(B < C) \vee (D < A).$$

Since the PA is limited to binary constraints, the IA network of Equation (2.4) cannot be represented as a PA network. □

The limited expressiveness of the PA is compensated for by its tractability. It turns out that most reasoning tasks for problems expressed in the PA can be solved in polynomial time, using 3- and 4-consistency.

The consistency of a PA network can be decided using path consistency. A given PA network is consistent if and only if after executing path consistency the resulting network is nonempty (Ladkin and Maddux [39]; see also Chapter 4 below). Thus, deciding the consistency of a PA network is $O(n^3)$. A faster, $O(n^2)$ algorithm for deciding consistency and for finding a consistent scenario is given in [68].

Path consistency can also be used in computing the minimal network. Consider a subset of PA networks, called *convex PA (CPA) networks*. In these networks, the constraints are taken from the set $\{<, \leq, =, \geq, >\}$; namely, the $\neq$ relation is excluded. It can be shown ([66]; see also Chapter 3 below), that path consistency is exact for CPA networks. Thus, computing the minimal network of a CPA network is $O(n^3)$. Path consistency is not exact, however, in the full PA (where the $\neq$ relation is allowed). In order to compute the minimal network for a PA network, we need to enforce 4-consistency [66], which requires time $O(n^4)$.

These results were used to identify tractable classes of IA networks. As we have seen before, some IA relations can be expressed by a conjunction of binary PA relations. Let IA_{PA} be the set of all these relations.[3] Similarly, let IA_{CPA} be the set of all IA relations that can be expressed by a conjunction of binary CPA relations. An IA network whose constraints are IA_{PA} or IA_{CPA} relations

[3]A complete list of these relations is given in [67] and [39].

will be called an *IA_{PA} network* or an *IA_{CPA} network*, respectively. According to the previous results, it is clear that these classes of networks are tractable. To solve such a network, we simply translate it into an equivalent point network (in $O(n^2)$ time), solve the resulting PA (or CPA) network, and then, if necessary (for example when the task is to compute the minimal network), translate the resulting network back into interval representation (again in time $O(n^2)$).

An alternative, sometimes more efficient way is to execute the appropriate local consistency algorithm, 3- or 4-consistency, on the interval network itself. In particular, 3-consistency computes the minimal network when applied to an IA_{CPA} network, and 4-consistency computes the minimal representation of an IA_{PA} network [67].

Although it may seem that IA_{PA} and IA_{CPA} networks are only of theoretical importance, it turns out that many AI applications that employ Allen's formalism in representing temporal knowledge, use essentially only IA_{PA} relations. Examples can be found in natural language understanding applications [5, 59], medical expert systems [30], medical diagnosis systems [34], and in qualitative reasoning [51].

Current research on qualitative temporal networks focuses mainly on identifying additional tractable subsets of Allen's IA. For example, using results from graph theory, another tractable class was recently reported in [29]. It is desirable to find tractable classes with powerful expressiveness; in particular, classes that subsume the PA. A negative result on the prospects of finding such classes is given in Chapter 4. We define a class of networks, called *interval-point algebra (IPA) networks*, where the constraints are relations between points and intervals. We will show (Theorem 4.3) that even this restricted subset, which subsumes the PA, is intractable.

Finally, it should be noted that in using the IA or the PA, one is committed to a single type of objects: intervals or points. In many cases, however, users may need the flexibility of expressing knowledge using both types of objects. Vilain [71] presented an hybrid system that consists of both points and intervals. Allen and Hayes [4] have also noted that a theory that can accommodate both types of objects is needed. This issue will be treated more formally in Chapter 4, where we develop a general *qualitative algebra* that allows references to both points and intervals, including relations between points and intervals; this algebra can be seen as a generalization of both the IA and the PA.

CHAPTER 3

Metric Networks

One of the requirements of a temporal reasoning system is the ability to deal with metric information. For instance, in Example 1.1 we need to express information on duration of events ("Fred's commuting time") or timing of events ("The time John left home"). Unfortunately, the IA and the PA do not offer a convenient mechanism for dealing with such knowledge.

Some suggestions about representing quantitative knowledge have been made. Dean and McDermott [17] introduced a *time management system* that can handle some quantitative information. This system was not formulated mathematically, however. Malik and Binford [46] and Valdés-Pérez [64] suggested that quantitative knowledge be represented as a set of constraints on the *temporal distance* between time points. If X_i and X_j are two time points, a constraint on their temporal distance would be of the form

$$X_j - X_i \leq c, \tag{3.1}$$

which gives rise to a set of linear inequalities. Malik and Binford suggested using the simplex algorithm to test the satisfiability of these inequalities.

In this chapter we shall follow Malik and Binford and Valdés-Pérez and present a formal, constraint-based model that facilitates the processing of such temporal-distance constraints. We consider time points as the variables we wish to constrain, where a time point may be a beginning or an ending point of some event, as well as a neutral point of time. We will use a form of temporal-distance constraints which is more general than Equation (3.1), because it allows disjunctive statements. To solve reasoning tasks in this model, we shall use constraint satisfaction techniques instead of using the (exponential) simplex algorithm. We discuss three reasoning tasks: finding all feasible times that a given event can occur, finding all possible relationships between two given events, and generating one or more scenarios consistent with the information provided.

Consider the following example.

Example 3.1 *John goes to work either by car (30–40 minutes) or by bus (at least 60 minutes). Fred goes to work either by car (20–30 minutes) or in a carpool (40–50 minutes). Today John left home between 7:10 and 7:20 a.m., and Fred arrived at work between 8:00 and 8:10 a.m. We also know that John arrived at work about 10–20 minutes after Fred left home.*

We wish to answer queries such as "Is the information in the story consistent?", "Is it possible that John took the bus and Fred used the carpool?", and "What are the possible times at which Fred left home?".

Let P_1 be the proposition "John was going to work" and P_2 the proposition "Fred was going to work." P_1 and P_2 are associated with intervals $[X_1, X_2]$ and $[X_3, X_4]$, respectively, where X_1 represents the time John left home, while X_4 represents the time Fred arrived at work. Several quantitative constraints are given in the story. From the fact that it takes John either 30–40 minutes or more than 60 minutes to get to work, the temporal distance between X_1 and X_2 is constrained by

$$30 \leq X_2 - X_1 \leq 40 \text{ or } X_2 - X_1 \geq 60.$$

Similar constraints apply to $X_4 - X_3$ and $X_2 - X_3$. Choosing $X_0 =$ 7:00 a.m., the fact that John left home between 7:10 and 7:20 a.m. imposes the constraint

$$10 \leq X_1 - X_0 \leq 20.$$

The constraint on $X_4 - X_0$ assumes a similar form.

The rest of this chapter is organized as follows. In Section 3.1, we present the TCSP model—a formal computational model that facilitates the representation and processing of quantitative, metric constraints. Section 3.2 deals with a restricted, simpler TCSP (called STP), solvable in polynomial time. Sections 3.3–3.5 offer some techniques for solving the general TCSP: decomposition into several STPs, approximation schemes, and network-based approaches. Section 3.6 relates the TCSP model to Allen's IA and to Vilain and Kautz's PA.

3.1 The Temporal Constraint Satisfaction Problem Model

The definitions needed for describing a TCSP follow closely those developed for the general CSP (Montanari [50]). A TCSP involves a set of variables

$\{X_1, \ldots, X_n\}$ having continuous domains; each variable represents a time point. Each constraint is represented by a set of intervals[1]

$$\{I_1, \ldots, I_k\} = \{[a_1, b_1], \ldots, [a_k, b_k]\}.$$

A unary constraint T_i restricts the domain of variable X_i to the given set of intervals; namely, it represents the disjunction

$$(a_1 \leq X_i \leq b_1) \vee \cdots \vee (a_k \leq X_i \leq b_k).$$

A binary constraint T_{ij} constrains the permissible values for the distance $X_j - X_i$; it represents the disjunction

$$(a_1 \leq X_j - X_i \leq b_1) \vee \cdots \vee (a_k \leq X_j - X_i \leq b_k).$$

We assume that constraints are always given in a *canonical form* in which all intervals are pairwise disjoint.

A *network of binary constraints* (a *binary TCSP*) consists of a set of variables $\{X_1, \ldots, X_n\}$ and a set of unary and binary constraints. Such a network can be represented by a *directed constraint graph*, where nodes represent variables and an edge $i \rightarrow j$ indicates that a constraint T_{ij} is specified; it is labeled by the interval set. Each input constraint T_{ij} implies an equivalent constraint T_{ji}; however, usually only one of these will be shown in the constraint graph. A special time point, X_0, is introduced to represent the "beginning of the world." All times are relative to X_0, thus we may treat each unary constraint T_i as a binary constraint T_{0i} (having the same interval representation). For simplicity we assume $X_0 = 0$. The constraint graph of Example 3.1 is given in Figure 3.1.

A tuple $X = (x_1, \ldots, x_n)$ is called a *solution* if the assignment $\{X_1 = x_1, \ldots, X_n = x_n\}$ satisfies all the constraints. A value v is a *feasible value* for variable X_i if there exists a solution in which $X_i = v$. The set of all feasible values of a variable is called the *minimal domain*. The network is *consistent* if at least one solution exists.

We define three binary operations on constraints—union, intersection, and composition—respecting their usual set-theoretic definitions.

[1]For simplicity we assume closed intervals; however, the same treatment applies to open and semi-open intervals.

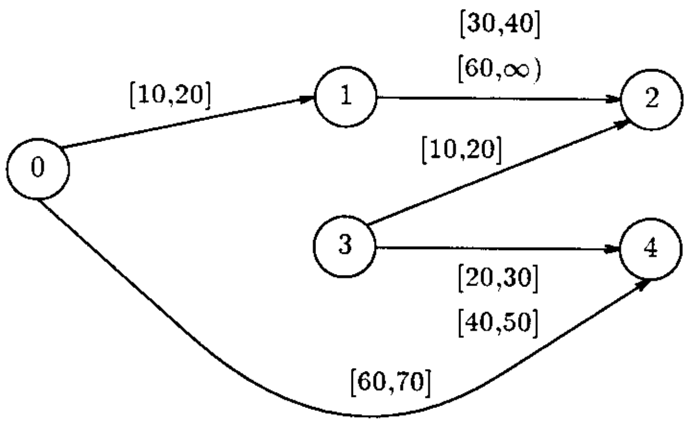

Figure 3.1: A constraint graph representing Example 3.1.

Definition 3.2 Let $T = \{I_1, \ldots, I_l\}$ and $S = \{J_1, \ldots, J_m\}$ be constraints, that is, sets of intervals of a real variable t (t corresponds to $X_j - X_i$ in the case these are binary constraints).

1. The *union* of T and S, denoted by $T \cup S$, admits only values that are allowed by either T or S, namely,

$$T \cup S = \{I_1, \ldots, I_l, J_1, \ldots, J_m\}.$$

2. The *intersection* of T and S, denoted by $T \oplus S$, admits only values that are allowed by both T and S, namely,

$$T \oplus S = \{K_1, \ldots, K_n\},$$

where $K_k = I_i \cap J_j$ for some i and j. Note that $n \leq l + m$.

3. The *composition* of T and S, denoted by $T \otimes S$, admits only values r for which there exist $t \in T$ and $s \in S$, such that $t + s = r$, namely,

$$T \otimes S = \{K_1, \ldots, K_n\},$$

where $K_k = [a + c, b + d]$ for some $I_i = [a, b]$ and $J_j = [c, d]$. Note that $n \leq l \times m$.

A pictorial illustration of the intersection and composition operations is given in Figure 3.2. Note that for some of these operations the resulting interval representation is not in canonical form. For instance, the composition operation

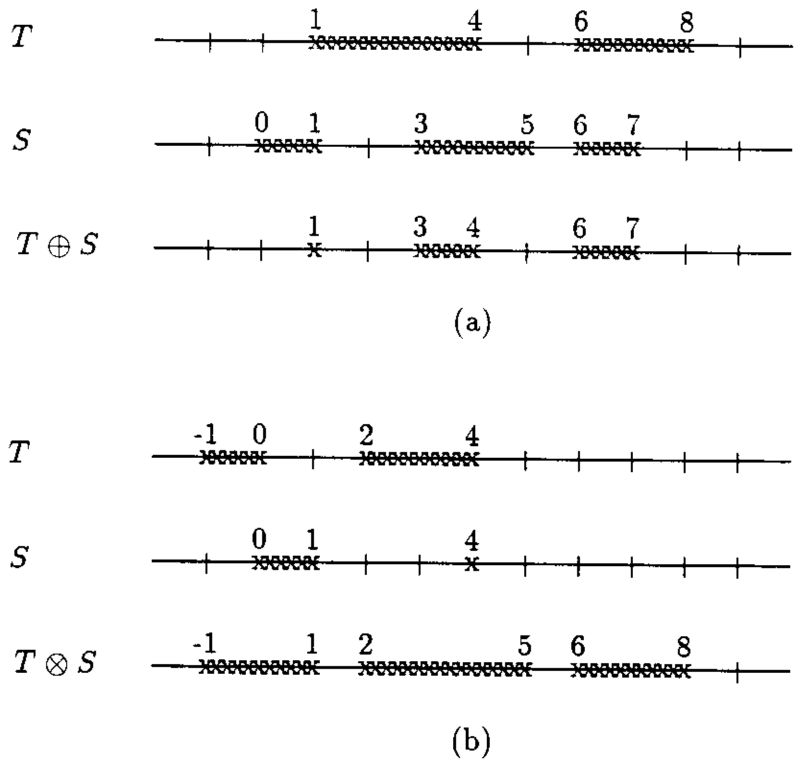

Figure 3.2: Operations on constraints: (a) intersection, (b) composition.

results in four intervals; however, due to overlap, only three of them appear in the canonical form. These three operations parallel the usual operations of union, intersection, and composition in general constraint networks [50]. In particular, when T and S represent binary constraints on the differences $X_j - X_i$ and $X_k - X_j$, respectively, $T \otimes S$ admits only pairs of values (x_i, x_k) for which there exists a value x_j such that (x_i, x_j) is permitted by T and (x_j, x_k) is permitted by S.

These operations are extended to operations on networks in the usual way. Given networks T and S on the same set of variables, we define

$$(T \cup S)_{ij} = T_{ij} \cup S_{ij},$$

and

$$(T \oplus S)_{ij} = T_{ij} \oplus S_{ij},$$

where i and j range over all pairs of variables.

A partial order among constraints can be defined as follows. A binary constraint T is *tighter* than S, denoted by $T \subseteq S$, if every pair of values allowed by T is also allowed by S; namely, for every interval $I \in T$ there exists an interval $J \in S$ such that $I \subseteq J$. The tightest constraint is the *empty constraint*, $\emptyset$ (if the network contains an empty constraint, then it is trivially inconsistent). The most relaxed constraint is the *universal constraint*, $(-\infty, \infty)$. Edges corresponding to universal constraints are usually omitted from the constraint graph.

A partial order among binary constraint networks having the same set of variables can be defined as follows. A network T is tighter than network S, denoted $T \subseteq S$, if the partial order $\subseteq$ is satisfied for all the corresponding constraints; namely, for all pairs i, j, $T_{ij} \subseteq S_{ij}$. Two networks are *equivalent* if they represent the same solution set. A network may have many equivalent representations; in particular, there is one equivalent network that is minimal with respect to $\subseteq$, called the *minimal network* (note that the minimal network is unique because equivalent networks are closed under intersection). The arc constraints specified by the minimal network are called the *minimal constraints*.

A network is *decomposable*[2] [50] if every locally consistent assignment[3] to any set of variables S can be extended to a solution. The importance of decomposability lies in facilitating the construction of a solution by a *backtrack-free search* [26].

Given a constraint network, the first interesting problem is to determine its consistency. If the network is consistent, we may wish to find some specific solutions, each representing a possible scenario, or to answer queries concerning the set of all solutions. The interesting queries are:

1. What are the possible times at which X_i could occur? (asking for the minimal domain of X_i).

2. What are all the possible relationships between X_i and X_j? (asking for the minimal constraint between X_i and X_j).

Computing the full minimal network would provide answers to all such queries. The rest of this chapter presents several techniques for solving these tasks.

[2]In [50] decomposability is defined for minimal networks only.

[3]An assignment of values to a set of variables S is *locally consistent* if it satisfies the constraints applicable to S, that is, those involving only variables in S (including the unary constraints).

3.2 The Simple Temporal Problem

A TCSP in which all constraints specify a single interval is called a *simple temporal problem (STP)*. In such a network, each edge $i \rightarrow j$ is labeled by an interval $[a_{ij}, b_{ij}]$ that represents the constraint

$$a_{ij} \leq X_j - X_i \leq b_{ij}. \tag{3.2}$$

Alternatively, the constraint can be expressed as a pair of inequalities:

$$X_j - X_i \leq b_{ij}, \tag{3.3}$$

and

$$X_i - X_j \leq -a_{ij}. \tag{3.4}$$

Thus, solving an STP amounts to solving a set of linear inequalities on the X_i's.

The problem of solving a system of linear inequalities is well known in the operations research literature. It can be solved by the (exponential) simplex method [11] or by Khachiyan's algorithm [36], which is rather complicated in practice. Fortunately, the special class of linear inequalities characterizing the STP admits a simpler solution; the inequalities are given a convenient graph representation, to which a shortest paths algorithm can be applied [8, 58, 41, 42]. In the AI literature, a data structure, similar to this graph representation, called a *time map*, was introduced by Dean and McDermott [17] to facilitate planning, but it was not formulated mathematically.

Formally, we associate an STP with a directed edge-weighted graph $G_d = (V, E_d)$, called a ***distance graph*** (to be distinguished from the constraint graph). It has the same node set as G, and each edge $i \rightarrow j \in E_d$ is labeled by a weight a_{ij} representing the linear inequality $X_j - X_i \leq a_{ij}$. In Example 3.1, if we assume that John used a car and Fred used a carpool, we get an STP having

$$T_{12} = \{[30, 40]\} \text{ and } T_{34} = \{[40, 50]\},$$

and the distance graph depicted in Figure 3.3.

Each path from i to j in G_d, $i_0 = i, i_1, \ldots, i_k = j$, induces the following constraint on the distance $X_j - X_i$:

$$X_j - X_i \leq \sum_{j=1}^{k} a_{i_{j-1}, i_j}. \tag{3.5}$$

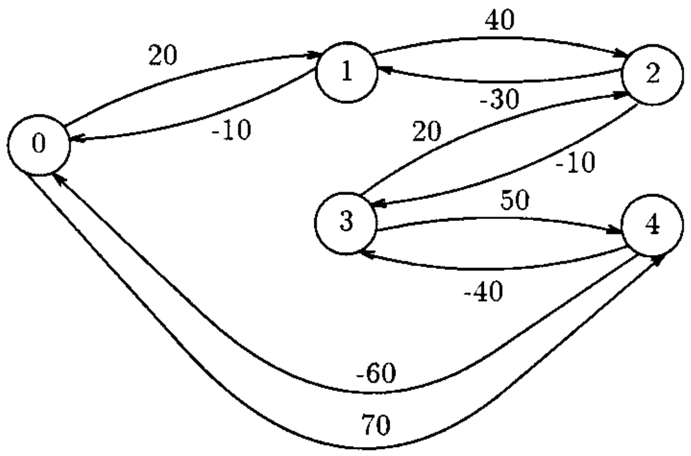

Figure 3.3: A distance graph representing a portion of Example 3.1.

If there is more than one path from i to j, then it can be verified easily that the intersection of all the induced path constraints yields

$$X_j - X_i \leq d_{ij}, \tag{3.6}$$

where d_{ij} is the length of the shortest path from i to j. Based on this observation, the following condition for the consistency of an STP can be established.

Theorem 3.3 (Shostak [58], Liao and Wong [42], Leiserson and Saxe [41]) *A given STP T is consistent if and only if its distance graph G_d has no negative cycles.*

Proof Suppose there is a negative cycle C consisting of nodes $i_1, \ldots, i_k = i_1$. Summing the inequalities along C yields

$$X_{i_1} - X_{i_1} < 0,$$

which cannot be satisfied.

Conversely, if there is no negative cycle in G_d, then the shortest path between each pair of nodes is well defined. For any pair of nodes i and j, the shortest paths satisfy $d_{0j} \leq d_{0i} + a_{ij}$; thus,

$$d_{0j} - d_{0i} \leq a_{ij}.$$

Hence, the tuple $(d_{01}, \ldots, d_{0n})$ is a solution of the given STP. □

Corollary 3.4 *Let G_d be the distance graph of a consistent STP. Two consistent scenarios are given by*

$$\begin{aligned} S_1 &= (d_{01}, \ldots, d_{0n}), \\ S_2 &= (-d_{10}, \ldots, -d_{n0}), \end{aligned}$$

which assign to each variable its latest and earliest possible times, respectively.

> **Proof** The proof of Theorem 3.3 shows that S_1 is a solution. To show that S_2 is a solution, note that for all i and j,
>
> $$d_{i0} \leq a_{ij} + d_{j0},$$
>
> or
>
> $$(-d_{j0}) - (-d_{i0}) \leq a_{ij},$$
>
> yielding S_2 as a solution. □

From the above discussion, it follows that a given STP can be effectively specified by a complete directed graph, called a *d-graph*, where each edge $i \rightarrow j$ is labeled by the shortest path length d_{ij} in G_d; the d-graph corresponds to a more explicit representation of our STP (see Equations (3.5) and (3.6)).

Theorem 3.5 (decomposability) *Any consistent STP is decomposable relative to the constraints in its d-graph.*

> **Proof** It suffices to show that any instantiation of a subset S of k variables $(1 \leq k < n)$ that satisfies all the shortest path constraints applicable to S is extensible to any other variable. This will be shown by induction on $|S| = k$.
>
> For $k = 1$, S consists of a single variable X_i instantiated to x_i. We will show that for any other variable X_j we can find an assignment $X_j = v$ that satisfies the shortest path constraints between them. The value v must satisfy
>
> $$-d_{ji} \leq v - x_i \leq d_{ij}. \qquad (3.7)$$
>
> Since all cycles in the distance graph are nonnegative, we have
>
> $$d_{ji} + d_{ij} \geq 0,$$

and hence there exists a value v satisfying Equation (3.7).

Assume that the theorem holds for $|S| = k-1$; we must show that it holds for $|S| = k$. Without loss of generality, let $S = \{X_1, \ldots, X_k\}$, and let $\{X_i = x_i | 1 \leq i \leq k\}$ be an assignment that satisfies the shortest path constraints among the variables in S. Let $X_{k+1} \notin S$. We need to find a value $X_{k+1} = v$ that satisfies the shortest path constraints between X_{k+1} and all variables in S. In other words, v must satisfy

$$v - x_i \leq d_{i,k+1},$$

$$x_i - v \leq d_{k+1,i},$$

for $= 1, \ldots, k$, or

$$v \leq \min\{x_i + d_{i,k+1} | 1 \leq i \leq k\},$$

$$v \geq \max\{x_i - d_{k+1,i} | 1 \leq i \leq k\}.$$

Suppose the minimum is attained at i_0 and the maximum at j_0. Thus, v must satisfy

$$x_{j_0} - d_{k+1,j_0} \leq v \leq x_{i_0} + d_{i_0,k+1}. \tag{3.8}$$

Since x_{i_0} and x_{j_0} satisfy the constraint between them, we have

$$x_{j_0} - x_{i_0} \leq d_{i_0,j_0}.$$

This, together with $d_{i_0,j_0} \leq d_{i_0,k+1} + d_{k+1,j_0}$, yields

$$x_{j_0} - d_{k+1,j_0} \leq x_{i_0} + d_{i_0,k+1}.$$

Therefore, there exists a value v that satisfies the condition of Equation (3.8). □

The importance of Theorem 3.5 lies in providing an efficient algorithm for assembling a solution to a given STP: we simply assign to each variable any value that satisfies the d-graph constraints relative to previous assignments (starting with $X_0 = 0$). Decomposability guarantees that such a value can always be found, regardless of the order of assignment. A second by-product of decomposability is that the domains characterized by the d-graph are minimal.

Corollary 3.6 *Let G_d be the distance graph of a consistent STP. The set of feasible values for variable X_i is $[-d_{i0}, d_{0i}]$.*

Proof According to Theorem 3.5, the assignment $X_0 = 0$ can be extended by assigning any value v satisfying $v \in [-d_{i0}, d_{0i}]$ to X_i. This assignment, in turn, can be extended to a full solution. Thus, v is a feasible value. □

We have noted that the d-graph represents a tighter, yet equivalent network of the original STP. From Theorem 3.5 we can now conclude that this new network is the minimal network.

Corollary 3.7 *Given a consistent STP T, the equivalent STP M, defined by*

$$\forall i, j, M_{ij} = \{[-d_{ji}, d_{ij}]\},$$

is the minimal network representation of T.

Proof We will show that M is the minimal network by showing that it cannot be tightened any more; in other words, starting with the assignment $X_0 = 0$, for any $d \in [-d_{ji}, d_{ij}]$ there exists a solution $X = (x_0, \ldots, x_n)$ in which $x_j - x_i = d$. There are two cases depending on d.

Case 1:

$$d \leq d_{0j} - d_{0i}. \tag{3.9}$$

According to Corollary 3.6, $X_i = d_{0i}$ is a feasible value. Clearly,

$$d_{0i} + d \geq d_{0i} - d_{ji},$$

and since

$$d_{ji} \leq d_{j0} + d_{0i},$$

we get

$$d_{0i} + d \geq -d_{j0}.$$

Together with Equation (3.9) we have

$$-d_{j0} \leq d_{0i} + d \leq d_{0j}.$$

Therefore, the assignment $X_j = d_{0i} + d$ satisfies the unary domain constraints on variable X_j, and

$$\{X_0 = 0, X_i = d_{0i}, X_j = d_{0i} + d\}$$

satisfies the constraints applicable to $\{X_0, X_i, X_j\}$. By Theorem 3.5 this partial assignment can be extended to a solution.
Case 2:

$$d \geq d_{0j} - d_{0i}. \tag{3.10}$$

According to Corollary 3.6, $X_j = d_{0j}$ is a feasible value. Clearly,

$$d_{0j} - d \geq d_{0j} - d_{ij},$$

and since

$$d_{ij} \leq d_{i0} + d_{0j},$$

we get

$$d_{0j} - d \geq -d_{i0}.$$

Together with Equation (3.10) we have

$$-d_{i0} \leq d_{0j} - d \leq d_{0i}.$$

Therefore, the assignment $X_i = d_{0j} - d$ satisfies the unary domain constraints on variable X_i, and

$$\{X_0 = 0, X_i = d_{0j} - d, X_j = d_{0j}\}$$

satisfies the constraints applicable to $\{X_0, X_i, X_j\}$. By Theorem 3.5 this partial assignment can be extended to a solution. $\square$

Illustration Consider the distance graph of Figure 3.3. Since there are no negative cycles, the corresponding STP is consistent. The shortest path distances, d_{ij}, are shown in Table 3.1. The minimal domains are $10 \leq X_1 \leq 20$, $40 \leq X_2 \leq 50$, $20 \leq X_3 \leq 30$, and $60 \leq X_4 \leq 70$. In particular, one special solution is the tuple $(d_{01}, \ldots, d_{04})$, namely the assignment

$$\{X_1 = 20, X_2 = 50, X_3 = 30, X_4 = 70\},$$

which selects for each variable its latest possible time. According to this solution, John left home at 7:10 a.m. and arrived at work at 7:50 a.m., while Fred left home at 7:30 a.m. and arrived at work at 8:10 a.m. The minimal network is given in Table 3.2. Notice that the minimal network is symmetric in the sense that if $T_{ij} = \{[a, b]\}$ then $T_{ji} = \{[-b, -a]\}$. An alternate scenario, in which John

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

Table 3.1: Lengths of shortest paths in the distance graph of Figure 3.3.

	0	1	2	3	4
0	[0]	[10,20]	[40,50]	[20,30]	[60,70]
1	[-20,-10]	[0]	[30,40]	[10,20]	[50,60]
2	[-50,-40]	[-40,-30]	[0]	[-20,-10]	[20,30]
3	[-30,-20]	[-20,-10]	[10,20]	[0]	[40,50]
4	[-70,-60]	[-60,-50]	[-30,-20]	[-50,-40]	[0]

Table 3.2: The minimal network corresponding to Figure 3.3.

used a bus and Fred used a carpool (i.e., $T_{12} = \{[60, \infty)\}$ and $T_{34} = \{[40, 50]\}$), results in a negative cycle and is therefore inconsistent. □

The d-graph of an STP can be constructed by applying Floyd-Warshall's All-Pairs-Shortest-Paths algorithm [53] to the distance graph (see Figure 3.4). The algorithm runs in time $O(n^3)$ and detects negative cycles simply by examining the sign of the diagonal elements d_{ii}. It constitutes, therefore, a polynomial time algorithm for determining the consistency of an STP and for computing both the minimal domains and the minimal network. Once the d-graph is available, assembling a solution requires only $O(n^2)$ time, because each successive assignment needs to be checked against previous assignments and is guaranteed to remain unaltered. Thus, finding a solution can be achieved in $O(n^3)$ time.

Algorithm All-Pairs-Shortest-Paths

```
1. for $i := 1$ to $n$ do $d_{ii} \leftarrow 0$
2. for $i, j := 1$ to $n$ do $d_{ij} \leftarrow a_{ij}$
3. for $k := 1$ to $n$ do
4.     for $i, j := 1$ to $n$ do
5.         $d_{ij} \leftarrow \min\{d_{ij}, d_{ik} + d_{kj}\}$
```

Figure 3.4: Floyd-Warshall's algorithm.

3.3 The General Temporal Constraint Satisfaction Problem

Having solved the STP, we now return to the general problem in which edges may be labeled by several intervals. Davis [13] showed that determining consistency for a general TCSP is NP-hard.

Theorem 3.8 (Davis [13]) (*i*) *Deciding consistency for a TCSP is NP-hard.* (*ii*) *Deciding consistency for a TCSP with no more than two intervals per edge is NP-hard.*

Proof (*i*) Reduction from 3-coloring. Let $G = (V, E)$ be a graph to be colored. We construct a TCSP T in the following way. For each node V_i, we introduce a variable X_i and a unary constraint on X_i

$$X_i \in \{[1], [2], [3]\}, \tag{3.11}$$

where [1], [2], and [3] stand for the three admissible colors. With each edge $(i, j) \in E$ we associate a binary constraint

$$X_j - X_i \in \{[-2], [-1], [1], [2]\}. \tag{3.12}$$

Equation (3.12) restricts X_i and X_j to different colors. Hence, T is consistent if and only if G is 3-colorable.

(*ii*) Again, reduction from 3-coloring. We construct a TCSP T as follows. For each node V_i, we introduce two variables X_i' and X_i'' having domains

$$X_i' \in \{[1],[2,3]\},$$

$$X_i'' \in \{[1,2],[3]\},$$

and restrict X_i' and X_i'' to being equal:

$$X_i' = X_i''.$$

This forces X_i' and X_i'' to assume integer values, as in Equation (3.11). To restrict the colors of nodes V_i and V_j to different colors, the following binary constraints are introduced:

$$X_j' - X_i' \in \{[-2],[-1,2]\},$$

$$X_j'' - X_i' \in \{[-2,-1],[1,2]\},$$

$$X_j' - X_i'' \in \{[-2,1],[2]\}.$$

T is consistent if and only if the graph is 3-colorable. □

A straightforward way of solving the general TCSP is to decompose it into several STPs, solve each one of them, and then combine the results. Given a binary TCSP T, we define a *labeling* of T as a selection of one interval from each constraint. Each labeling defines an STP graph whose edges are labeled by the selected intervals. We can solve any of the TCSP tasks by considering all its STPs. Specifically, the original network is consistent if and only if there is a labeling whose associated STP is consistent. Any solution of T is also a solution of one of its STPs and vice versa. Also, the minimal network of T can be computed from the minimal networks associated with its individual STPs, as stated in the following theorem.

Theorem 3.9 *The minimal network M of a given TCSP T satisfies*

$$M = \bigcup_l M_l,$$

where M_l is the minimal network of the STP defined by labeling l, and the union is over all the possible labelings.

	0	1	2	3	4
0	[0]	[10,20]	[40,60] [70]	[20,50]	[60,70]
1	[-20,-10]	[0]	[30,40] [60]	[10,30] [40]	[40,60]
2	[-70] [-60,-40]	[-60] [-40,-30]	[0]	[-20,-10]	[0,30]
3	[-50,-20]	[-40] [-30,-10]	[10,20]	[0]	[20,30] [40,50]
4	[-70,-60]	[-60,-40]	[-30,0]	[-50,-40] [-30,-20]	[0]

Table 3.3: The minimal network of Example 3.1.

Proof We first note that the solution set of T is identical to the union of the solution sets of its labelings. Hence, $\bigcup M_l$ is equivalent to T. M is by definition the tightest of all networks equivalent to T, and therefore $M \subseteq \bigcup M_l$. Now suppose that M is strictly tighter than $\bigcup M_l$. Then, there exist a pair of variables i and j, a labeling s, and a value d, such that $d \in (M_s)_{ij}$ but $d \notin M_{ij}$. Let x and y be values of the variables i and j, respectively, such that $y - x = d$. According to the minimality of M_s, this partial assignment can be extended to a solution of s, which is also a solution of T; hence $d \in M_{ij}$, yielding a contradiction. Therefore, $\bigcup M_l \subseteq M$. □

Illustration The minimal network of Example 3.1 is shown in Table 3.3. In this case, only three of the four possible labelings contribute to the minimal network. □

The complexity of solving a general TCSP by generating all the labelings and solving them independently is $O(n^3k^e)$, where k is the maximum number of intervals labeling an edge and e is the number of edges.

This brute-force enumeration process can be pruned significantly by running a backtracking search on a meta-CSP in which the variables are the TCSP's edges and the domains are the possible intervals. The backtracking algorithm assigns an interval to an edge, as long as the condition of Theorem 3.3 is satisfied; if no such assignment is possible, it backtracks.

Formally, let T be a given TCSP, and let $G = (V, E)$ be its associated constraint graph. Let CSP(T) be a discrete CSP with variables $X_1, \ldots, X_m$, where $m = |E|$ and variable X_i corresponds to edge $e_i \in E$. The domain of X_i consists of the intervals $I_1, \ldots, I_k$ that label e_i in G. The constraints are not given implicitly (as a list of allowed or disallowed combinations); instead, any assignment $\{X_{i_1} = I_{i_1}, \ldots X_{i_s} = I_{i_s}\}$ is consistent if and only if the corresponding STP is consistent. Clearly, each solution of CSP(T) corresponds to a consistent labeling of G, and thus any algorithm that finds all the solutions of CSP(T) can be used to solve T. A backtrack algorithm that computes the minimal network of a TCSP is shown in Figure 3.5. It is defined by two recursive procedures: Forward and Go-back. The first extends a current partial assignment if possible, and the second handles dead-end situations. The procedures maintain a list of candidate intervals C_i for each variable X_i.

Backtrack is initiated by calling Forward with $i = 0$, namely, the instantiated list is empty. The procedure Solve-STP$(I_1, \ldots, I_m)$ returns the minimal network of the STP defined by $\{I_1, \ldots, I_m\}$. The procedure Consistent-STP$(I_1, \ldots, I_i, I_j)$ determines whether the partial STP defined by $\{I_1, \ldots, I_i, I_j\}$ is consistent; it can be done either by using an all-pairs-shortest-paths algorithm or an improved algorithm to be described in Section 3.4. At the beginning of the algorithm $M = \emptyset$ and, upon termination, M contains the minimal network (if $M = \emptyset$ then the network is inconsistent). If our task is to find a single solution, then once we find a consistent labeling we may construct a solution using the technique described in the previous section.

Although the worst-case complexity of this approach is also $O(n^3 k^e)$, it enables us to utilize enhancement techniques that, in practice, reduce the complexity of backtrack substantially below its worst case value. Such techniques include backjumping [28], variable ordering [26, 56, 20], value ordering [32, 21], and learning schemes [18]. Moreover, with some investment of storage space, the work done on any partial instantiation can be utilized toward its extension (without redoing the problem afresh), and this reduces the time complexity to $O(n^2 k^e)$.

In the following sections we will present alternative approaches for solving the general TCSP. In particular, Section 3.4 discusses path consistency algorithms that can be used as either an approximation or a preprocessing step before applying backtracking. Section 3.5 shows how the topology of the constraint graph

```
Forward($I_1, \ldots, I_i$)
1.  if $i = m$ then begin
2.      $M \leftarrow M \cup$ Solve-STP($I_1, \ldots, I_m$)
3.      Go-Back($I_1, \ldots, I_m$)
4.  end
5.  $C_{i+1} \leftarrow \emptyset$
6.  for every $I_j$ in $D_{i+1}$ do
7.      if Consistent-STP($I_1, \ldots, I_i, I_j$) then
8.          $C_{i+1} \leftarrow C_{i+1} \cup \{I_j\}$
9.  if $C_{i+1} \neq \emptyset$ then begin
10.     $I_{i+1} \leftarrow$ first element in $C_{i+1}$
11.     remove $I_{i+1}$ from $C_{i+1}$
12.     Forward($I_1, \ldots, I_i, I_{i+1}$)
13. end
14. else Go-Back($I_1, \ldots, I_i$)

Go-back($I_1, \ldots, I_i$)
1. if $i = 0$ then exit
2. if $C_i \neq \emptyset$ then begin
3.     $I_i \leftarrow$ first element in $C_i$
4.     remove $I_i$ from $C_i$
5.     Forward($I_1, \ldots, I_i$)
6. end
7. else Go-back($I_1, \ldots, I_{i-1}$)
```

Figure 3.5: A backtracking algorithm.

can be exploited to yield more efficient algorithms.

3.4 Path Consistency Algorithms

Imposing local consistency among subsets of variables may serve as a preprocessing step to improve backtrack. Local consistency algorithms, especially path consistency, might also serve as a good approximation scheme which often yields the minimal network. In this section we study the applicability of path consistency and its weaker version, directional path consistency, in the TCSP framework.

Floyd-Warshall's algorithm, used for solving the STP, can be considered a *relaxation* algorithm—in every step of the process the label of an edge is updated by an amount that depends only on the current labels of adjacent edges. In fact, there is a rich family of similar algorithms [1, 9, 40, 62, 61, 54], all based on the same principle. Montanari [50] was the first to use such an algorithm, called *path consistency*, in the context of constraint satisfaction problems. This was further explored and analyzed by Mackworth [43] and Mackworth and Freuder [45].

Pursuing its traditional role [50, 43], path consistency in the context of a TCSP is defined as follows.

Definition 3.10 A path through nodes $i_0, i_1, \ldots, i_m$ is *path consistent* if and only if, for any pair of values v_0 and v_m such that $v_m - v_0 \in T_{i_0,i_m}$, there exists a sequence of values $v_1, \ldots, v_{m-1}$ such that

$$v_1 - v_0 \in T_{i_0,i_1}, v_2 - v_1 \in T_{i_1,i_2}, \ldots, v_m - v_{m-1} \in T_{i_{m-1},i_m}.$$

A *network* is path consistent if and only if every path is consistent.

Using the operations $\oplus$ and $\otimes$ (denoting intersection and composition), Montanari's path consistency algorithm (equivalent to Mackworth's PC-1 [43]) is shown in Figure 3.6. The algorithm imposes local consistency among triplets of variables until a fixed point is reached or until some constraint becomes empty, indicating an inconsistent network. Clearly, the algorithm computes a network that is equivalent to the original one. For discrete-domain CSPs, Montanari showed that the algorithm terminates and that the resulting network is indeed path consistent. In our case, given that TCSPs are continuous-domain, one cannot guarantee that the algorithm terminates. It is clear, however, that running

Algorithm PC-1

```
1. repeat
2.     $S \leftarrow T$
3.     for $k := 1$ to $n$ do
4.         for $i, j := 1$ to $n$ do begin
5.             $T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj}$
6.             if $T_{ij} = \emptyset$ then
7.                 exit (the network is inconsistent)
8.         end
9. until $S = T$
```

Figure 3.6: PC-1—a path consistency algorithm.

the algorithm indefinitely will result in a limit network. Each step of the algorithm yields a tighter network and, since the network is bounded below by the minimal network, a limit point is assured. Moreover, analysis shows that for all practical purposes PC-1 terminates in a finite number of steps. This will be shown in two parts: first for STPs, then for general TCSPs.

Comparing Figures 3.4 and 3.6, we see that PC-1 is a generalization of the All-Pairs-Shortest-Paths algorithm. When applied to an STP, the relaxation step that updates T_{ij} amounts to two local operations of updating the shortest path length, d_{ij}, in Floyd-Warshall's algorithm. Therefore:

Theorem 3.11 *Applying* PC-1 *to an STP network is identical to applying Floyd-Warshall's algorithm to its distance graph.*

An immediate corollary of this theorem is that PC-1 terminates and produces a path-consistent network (see [12, 43, 50] for additional relationships between shortest paths algorithms and path consistency).

Regarding general TCSPs, two questions must be addressed: does PC-1 terminate and compute a path-consistent network, and is the resulting network minimal. We will next show that the answer to the first question is affirmative while the answer to the second is negative.

It is simple to show that PC-1 terminates for *integral* TCSPs, in which the extreme points of all intervals are integers. This is so because each intersection operation at Step 5 must tighten a constraint by an integral amount. For nonintegral TCSPs, the same argument holds if the extreme points are rational numbers (these will be called *rational* TCSPs); we simply multiply all quantities by the greatest common divisor of the extreme points. This was shown more formally by Ladkin [38]. Thus, since all practical problems are expressible by rational numbers, PC-1 can be regarded as terminating. Once termination has been ascertained, the path consistency of the resulting network can be established by straightforward application of Montanari's proof [50]; the continuous nature of temporal domains plays no role. In summary,

Theorem 3.12 *Algorithm* PC-1 *computes a path-consistent network.*

Having established that PC-1 terminates and computes a path-consistent network, we next ask whether the resulting network is minimal. Montanari showed that when the constraints obey the distributivity property (i.e., that composition distributes over intersection), any path-consistent network is both minimal and decomposable. Moreover, in such a case only one application of the main loop (Steps 1–9) is sufficient for reaching the fixed point. When constraints are defined by one interval (the STP case), the distributivity property holds and, indeed, for this case, the path-consistent network is minimal (Corollary 3.7), decomposable (Theorem 3.5), and requires only one iteration (see Floyd-Warshall's algorithm). Unfortunately, distributivity does not hold for multi-interval TCSPs, as can be seen in the following example.

Example 3.13 Consider the network shown in Figure 3.7 where, for convenience, both directions of each edge are explicitly given. There are two paths from node 1 to node 3, representing the constraints $T_{13} = \{[25, 50]\}$ and $S_{13} = \{[0, 30], [40, 50]\}$ (the latter is obtained by composing T_{12} with T_{23}). Performing intersection first and composition next, we get

$$\begin{aligned} &T_{01} \otimes (T_{13} \oplus S_{13}) = \\ &\quad \{[0, 1], [10, 20]\} \otimes \{[25, 30], [40, 50]\} = \\ &\quad \{[25, 31], [35, 70]\}. \end{aligned}$$

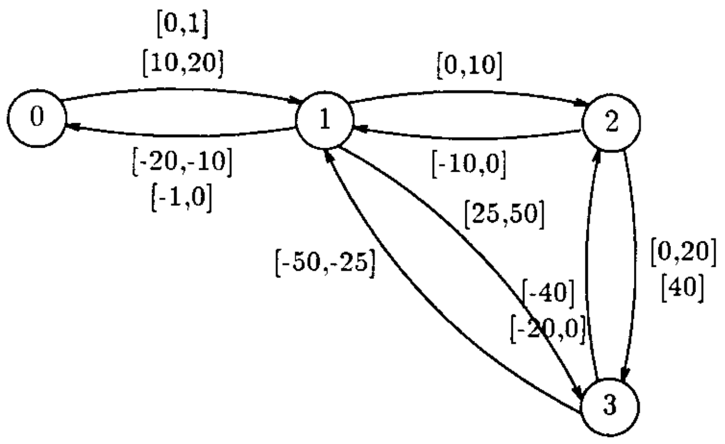

Figure 3.7: A nondistributive network.

Performing composition first and then intersection, results in

$$\begin{aligned}&(T_{01} \otimes T_{13}) \oplus (T_{01} \otimes S_{13}) = \\ &\quad \{[0,31],[40,51],[10,50],[50,70]\} \oplus \{[25,51],[35,70]\} = \\ &\quad \{[25,70]\}.\end{aligned}$$

Clearly, distributivity does not hold. Indeed, if we apply path consistency to this network then after one iteration we have $T_{03} = \{[25,70]\}$, whereas in the minimal network (shown in Table 3.4) $M_{03} = \{[25,31],[35,70]\}$. Interestingly, another application of the main loop does result in a fixed point which is also the minimal network (see Section 3.5). □

In general CSPs, it is well known that path consistency may not converge to the minimal network. The next example (modeled on Montanari [50]) will demonstrate that this phenomenon persists also in temporal problems; path consistency does not even detect inconsistency.

Example 3.14 Consider the 3-coloring problem on K_4, the complete graph of four nodes. The problem is obviously inconsistent and at the same time path consistent—every set of three nodes can be 3-colored. Translating this problem into TCSP notation, as in the proof of Theorem 3.8, yields the desired example. The problem consists of four variables $X_1, \ldots, X_4$, each having a domain $\{[1],[2],[3]\}$, connected by six binary constraints

$$X_j - X_i \in \{[-2],[-1],[1],[2]\},$$

	0	1	2	3
0	[0]	[0,1] [10,20]	[0,30]	[25,31] [35,70]
1	[-20,-10] [-1,0]	[0]	[0,10]	[25,30] [40,50]
2	[-30,0]	[-10,0]	[0]	[15,20] [40]
3	[-70,-35] [-31,-25]	[-50,-40] [-30,-25]	[-40] [-20,-15]	[0]

Table 3.4: The minimal network of Example 3.13.

for $i, j = 1, \ldots, 4, i \neq j$. The resulting network is already path consistent, yet PC-1 will fail to detect its inconsistency. □

A more efficient path consistency algorithm is the temporal equivalent of Mackworth's PC-2 [43], shown in Figure 3.8. The function REVISE$((i,k,j))$ updates T_{ij} by considering the length-2 path from i to j through k

$$T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj},$$

and returns true if T_{ij} has been modified. The function RELATED-PATHS$((i,k,j))$ returns the set of length-2 paths that need to be considered if T_{ij} is changed. The details of RELATED-PATHS are given in [43].

For discrete CSPs, path consistency can be achieved in time polynomial in n (the number of variables) and k (the maximum domain size) [45]. We will now show that the temporal mirror of PC-2 achieves path consistency in $O(n^3R^3)$, where R is the *range* of the network (expressed in terms of the coarsest possible time units).

Definition 3.15 Let T be an integral TCSP. The *range* of a constraint

$$T_{ij} = \{[a_1, b_1], \ldots, [a_n, b_n]\}$$

is $b_n - a_1$. The range of T is the maximum range over all constraints.

For a *rational* network, the range is defined as the range of the equivalent integral network, obtained from the input network by multiplying all extreme points by their greatest common divisor.

Algorithm PC-2

```
1. $Q \leftarrow \{(i,k,j) | i < j, k \neq i,j\}$
2. **while** $Q \neq \emptyset$ **do begin**
3.     select and delete a path $(i,k,j)$ from $Q$
4.     **if** REVISE($(i,k,j)$) **then**
5.         $Q \leftarrow Q \cup$ RELATED-PATHS($(i,k,j)$)
6. **end**
```

Figure 3.8: PC-2—a more efficient path consistency algorithm.

Theorem 3.16 *Temporal path consistency can be achieved in $O(n^3R)$ relaxation steps and $O(n^3R^3)$ arithmetic operations, where R is the range of the TCSP expressed in the coarsest possible time units.*

Proof Let T be a given TCSP. Without loss of generality, we may assume that T is integral; otherwise, we can simulate the algorithm on the equivalent integral network. The worst-case running time of PC-2 occurs when every constraint interval is decreased by only one time unit each time it is tightened by REVISE. In this case, if R is the maximum constraint range, each constraint might be updated $O(R)$ times. Also, in the worst case, when a constraint is modified, $O(n)$ paths are added to Q (see [43]). Thus, if we use the number of relaxation steps (calls to REVISE) as the complexity measure then, since there are $O(n^2)$ constraints, the total complexity of PC-2 is $O(n^3R)$. A more realistic measure would be the number of arithmetic operations. Each relaxation operation, $A \oplus B \otimes C$, where l, m, and n are the number of intervals in A, B, and C, respectively, involves $O(l + m \times n)$ arithmetic operations. Thus, since each relaxation step may involve as many as $O(R^2)$ operations, the total time is $O(n^3R^3)$. □

For comparison to chronological backtracking, note that R must be at least as large as k (the number of intervals per constraint). However, if the edges are labeled by a few intervals, $O(k^e)$ may reflect a lower complexity than $O(R^3)$.

Algorithm DPC

```
1. for $k := n$ downto 1 by -1 do
2.     for all $i, j < k$ such that $(i,k),(j,k) \in E$ do begin
3.         $T_{ij} \leftarrow T_{ij} \oplus T_{ik} \otimes T_{kj}$
4.         $E \leftarrow E \cup (i,j)$
5.         if $T_{ij} = \emptyset$ then
6.             exit (the network is inconsistent)
7.     end
```

Figure 3.9: DPC—an algorithm enforcing directional path consistency.

Although path consistency algorithms are not guaranteed to compute the minimal network, they often provide a practical alternative and a complementary approach to the decomposition scheme. Moreover, they are readily amenable to parallel and distributed computation. In preliminary experiments on small random problems (each consisting of 5–7 variables), PC-1 always found the minimal network (Yaara Levi and Margalit Pinkas, personal communication). On the basis of these experiments, it appears that path consistency will substantially reduce the amount of work done by backtracking. To fully assess the benefits of the path consistency scheme, full-scale experimental studies should be undertaken.

Some problems may benefit from a weaker version of path consistency, called *directional path consistency* [21], which can be enforced more efficiently.

Definition 3.17 Let d be an ordering on the variables, and let X_i precede X_j in d if and only if $i < j$. A constraint graph G is *directional path consistent* with respect to d if, for every pair of values v_i and v_j such that $v_j - v_i \in T_{ij}$ and for every $k > i, j$, there exists a value v_k such that $v_k - v_i \in T_{ik}$ and $v_j - v_k \in T_{kj}$.

Given a TCSP T, its associated constraint graph $G = (V, E)$, and an ordering d, directional path consistency can be achieved by algorithm DPC, shown in Figure 3.9, which is the temporal counterpart of the algorithm given in [21].

DPC is similar to PC-1, but unlike PC-1 it is a single pass algorithm. Note also that in Step 4, the set of edges E is increased dynamically by the relaxation operation of Step 3. The network defined by the final set of edges is called the *induced graph.*

If one of the constraints becomes empty (at Step 5) then the original network must have been inconsistent. However, as in the case of nontemporal CSPs, we are not guaranteed that the algorithm will always detect an inconsistency if one exists. Next we show that such a guarantee can be assured for STPs.

Definition 3.18 Let T be a TCSP. A cycle $i_0, \ldots, i_k = i_0$ is called *valid* if and only if

$$0 \in T_{i_0,i_1} \otimes \cdots \otimes T_{i_{k-1},i_k}.$$

Lemma 3.19 *A given STP T is consistent if and only if all the cycles in its constraint graph are valid.*

Proof If the network is consistent then all the cycles are valid, since if there was an invalid cycle C, $i_0, \ldots, i_k = i_0$, then the path constraint along C would yield

$$X_{i_0} - X_{i_0} \neq 0,$$

reflecting inconsistency.

Conversely, assume that all the cycles are valid. We will show that the network is consistent. According to Theorem 3.3 we need to show only that there is no negative cycle in the corresponding distance graph. Suppose there was such a negative cycle C consisting of nodes $i_0, \ldots, i_k = i_0$ and edge weights $a_{0,1}, a_{1,2}, \ldots, a_{k-1,k} = a_{k-1,0}$. Since C is negative, we have

$$\sum_{j=1}^{k} a_{i_{j-1},i_j} < 0. \tag{3.13}$$

Moreover, from Equations (3.2) through (3.4) we obtain

$$-\, a_{i_j,i_{j-1}} \leq a_{i_{j-1},i_j} \tag{3.14}$$

for $j = 1, \ldots, k$. Thus, combining Equations (3.13) and (3.14) yields

$$0 \notin [\sum_{j=1}^{k} -a_{i_j,i_{j-1}}, \sum_{j=1}^{k} a_{i_{j-1},i_j}].$$

At the same time, applying the composition along C gives

$$T_{i_0,i_1} \otimes \cdots \otimes T_{i_{k-1},i_k} = [\sum_{j=1}^{k} -a_{i_j,i_{j-1}}, \sum_{j=1}^{k} a_{i_{j-1},i_j}],$$

thus rendering C invalid—a contradiction. □

Theorem 3.20 *Given an STP T, algorithm* DPC *halts at Step 5 if and only if the network is inconsistent.*

Proof The *only if* part is trivial; we will show the *if* part. Suppose the network is inconsistent; then, according to Lemma 3.19, there exists an invalid cycle C. Let the nodes of C be the set $\{i_1, \ldots, i_k\}$, and order it along d, namely, i_j will be processed after i_k whenever $j < k$. We next prove the following lemma.

Lemma 3.21 *For all $j, 0 \leq j \leq k-3$, when node i_{k-j} is about to be processed (Step 1), there exists an invalid cycle C_j, consisting of nodes $\{i_1, \ldots, i_{k-j}\}$.*

Proof By induction on j. The lemma holds for $j = 0$ because the cycle $C_0 = C$ was assumed to be invalid in the original network, and DPC can only render constraints tighter. Thus, C_0 must remain invalid when node i_k is processed.

Assume the lemma holds for $j-1, j > 0$. By the induction hypothesis, when node i_{k-j+1} was about to be processed, there was an invalid cycle C_{k-j+1} consisting of nodes $\{i_1, \ldots, i_{k-j+1}\}$. Let s and r be the neighbors of i_{k-j+1} in C_{k-j+1}, and let P_{rs} be the path from r to s in C_{k-j+1}. When node i_{k-j+1} is processed, the constraint T_{sr} is tightened, and the newly created cycle is

$$C_{k-j} = (s, r) \cup P_{rs}.$$

The constraint along C_{k-j} is tighter than the constraint along C_{k-j+1}, and thus C_{k-j} is invalid. Between the time that i_{k-j+1} is processed until the time i_{k-j} is processed, DPC further tightens the constraints along C_{k-j}. Thus, the cycle remains invalid while i_{k-j} is being processed. □

According to Lemma 3.21, when node i_3 is about to be processed, there exists an invalid cycle C_3, consisting of nodes i_1, i_2, and i_3. Let $T_{i_1,i_3} = \{[a,b]\}$, $T_{i_3,i_2} = \{[c,d]\}$, and $T_{i_2,i_1} = \{[e,f]\}$. At Step 3 the constraint T_{i_1,i_2} is updated such that

$$T_{i_1,i_2} = \{[\max\{-f, a+c\}, \min\{-e, b+d\}]\}.$$

Since C_3 is invalid, $0 \notin [a+c+e, b+d+f]$. If $a+c+e > 0$, then $a+c > -e$, and $T_{i_1,i_2} = \emptyset$. Otherwise, $b+d+f < 0$ or $b+d < -f$, and thus $T_{i_1,i_2} = \emptyset$. Hence, at Step 5 the algorithm must halt. □

It is well known that for general CSPs, directional path consistency can be achieved more efficiently than full path consistency [21]; instead of $O(n^3)$ time, DPC runs in $O(nW^*(d)^2)$ time, where $W^*(d)$ is the maximum number of parents that a node possesses in the induced graph. To assess the savings in the context of temporal problems, recall that each relaxation step involves $O(R^2)$ arithmetic operations, thus yielding a worst case bound of $O(nW^*(d)^2R^2)$ operations. Another upper bound emerges from the fact that with every node processed the number of intervals recorded may increase by a factor not greater than k, thus giving a total of at most $O(k^n)$ intervals and arithmetic operations in any relaxation step. Hence, the upper bound is $O(nW^*(d)^2k^n)$.

For STPs, each relaxation step involves a constant number of arithmetic operations, and thus consistency for STPs can be determined in $O(nW^*(d)^2)$ time, in contrast with the $O(n^3)$ time needed for full path consistency. $W^*(d)$ could be substantially lower than n and can be found in time $O(|V| + |E|)$ prior to the actual processing [10, 63, 7].

Note that directional path consistency is weaker, generally speaking, than full path consistency and, hence, might lead to a higher number of dead ends for backtrack. However, the use of directional path consistency yields more dramatic savings if it is embedded within backtracking as the consistency checking routine Consistent-STP (Figure 3.5). Instead of checking consistency by the $O(n^3)$ Floyd-Warshall algorithm, using DPC will reduce the search effort of backtrack by a factor of roughly $(n/W^*(d)^2)$. In the next section, we characterize a class of problems that gain fuller benefit from the efficiency of directional path consistency.

3.5 Network-Based Algorithms

So far we have presented techniques for processing networks of a general structure. The topology of the constraint graph did not play any role in the choice of solution technique. However, considering the topological features of the constraint network may guide us, as they do for nontemporal CSPs, in selecting efficient solution methods that have lower worst-case complexity than naive backtracking.

We first consider the task of finding a single solution to TCSPs. The infinite domains associated with temporal problems prevent us from searching exhaustively through the space of possible scenarios. Instead, we must seek ways of constructing a solution in a guided manner. If the network is decomposable (such as in the case of STPs), a solution can be assembled incrementally, without backtracking, under any ordering we choose. If the network is not decomposable, the feasibility of achieving a backtrack-free solution relies on the topology of the constraint graph. Freuder [26] and Dechter and Pearl [21] have identified sufficient conditions for a network to yield a backtrack-free solution, invoking the notion of higher-order consistency. To demonstrate, we will focus on a class of networks that admit a particularly efficient method when applied to temporal problems. This class is called *series-parallel networks* and is equivalent to the *regular width 2* networks of [21].

Definition 3.22 A network is said to be *series-parallel* with respect to a pair of nodes i and j if it can be reduced to the edge (i, j) by repeated application of the following *reduction* operation: select a node of degree 2 or less, remove it from the network, and connect its neighbors (unless they are connected already). If the network is series-parallel with respect to *any* pair of nodes, it is called a series-parallel network.

Testing whether a network is series-parallel requires $O(|V|)$ time and, as a by-product, the testing algorithm produces an ordering d for which $W^*(d) = 2$ corresponding to an admissible sequence of reduction operations [6, 73]. It can be shown [21] that enforcing directional path consistency, in an ordering opposite to d, renders such networks backtrack-free and computes the minimal constraint between the first two nodes in d. If the network is inconsistent, some constraint will become empty; otherwise, a consistent solution can be constructed in a backtrack-free fashion. Since $W^*(d) = 2$, DPC runs in $O(nK)$ time, where K

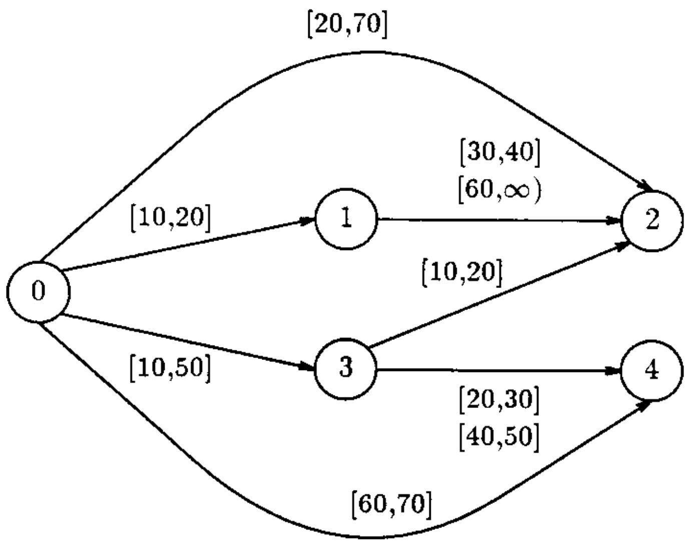

Figure 3.10: A directional path-consistent network of Example 3.1.

is the maximum number of intervals labeling any edge in the induced graph. The solution-construction phase requires an additional $O(nK)$ arithmetic operations.

Montanari [50] showed that full path consistency computes the minimal constraints on every pair of nodes, relative to which the network is series-parallel. In this respect, running full path consistency can be viewed as running DPC along several orderings in parallel, thereby giving any pair of nodes a chance of being the first.

Illustration Consider the network of Example 3.1. The network is obviously series-parallel, since it admits any sequence of reduction operations. Applying DPC in the ordering $d = (0, 1, 2, 3, 4)$ results in the network shown in Figure 3.5. Since no constraint becomes empty, the network is consistent, and a solution can be constructed backtrack-free along d. Moreover, since the network is series-parallel with respect to any pair of nodes, full path consistency computes the full minimal network (see Section 3.4). □

A generalization of directional path consistency, called *adaptive consistency* [21, 22], can render any network backtrack-free by recording higher-order constraints on the neighbors of the nodes processed. This method, although it exhibits a low worst-case complexity in general CSPs, turns out to be impractical in temporal problems, primarily due to difficulties in storing and processing higher-order interval constraints.

Another approach, which exploits the structure of the constraint graph, involves decomposition into *nonseparable components*. We shall show that this can facilitate finding both a consistent solution and the minimal network.

Definition 3.23 (Even [24]). A connected graph $G = (V, E)$ is said to have a *separation vertex* v (sometimes also called an *articulation point*) if there exist vertices a and b, $a \neq v$ and $b \neq v$, such that all the paths connecting a and b pass through v. In this case we also say that v separates a from b. A graph that has a separation vertex is called *separable*, and one that has none is called *nonseparable*. Let $V' \subseteq V$. The induced subgraph $G' = (V', E')$ is called a *nonseparable component* if G' is nonseparable and if for every larger V'', $V' \subset V'' \subseteq V$, the induced subgraph $G'' = (V'', E'')$ is separable.

Definition 3.24 (Even [24]). Let $C_1, \ldots, C_m$ be the nonseparable components of the connected graph $G = (V, E)$, and let $s_1, \ldots, s_p$ be its separating vertices. The *superstructure* of G, $\overline{G} = (\overline{V}, \overline{E})$, is defined as follows:

$$\overline{V} = \{s_1, \ldots, s_p\} \cup \{C_1, \ldots, C_m\},$$

$$\overline{E} = \{(s_i, C_j) | s_i \text{ is a vertex of } C_j \text{ in } G\}.$$

It is well known that the superstructure is a tree. The nonseparable components and their superstructure can be found in time $O(|E|)$ (see [24]).

Definition 3.25 Let $G = (V, E)$ be a constraint graph of a TCSP T, and let $C = (V', E')$ be a nonseparable component of G. The *minimal network of component* C, M_C, is the minimal network of the TCSP defined by C.

Theorem 3.26 *Let M be the minimal network of a consistent TCSP T, and let M_C be the minimal network of a nonseparable component $C = (V', E')$ in the constraint graph $G = (V, E)$ of T. Then, for all $i, j \in V'$, $M_{ij} = (M_C)_{ij}$.*

Proof Clearly, $M_{ij} \subseteq (M_C)_{ij}$. To prove $(M_C)_{ij} \subseteq M_{ij}$, we show that every value in $(M_C)_{ij}$ also appears in M_{ij}. Let $v \in (M_C)_{ij}$. There exists a labeling L_1 of C, having minimal network M_{L_1}, in which $v \in (M_{L_1})_{ij}$. Consider the TCSP defined by $G - C = (V, E - E')$.

Since T is consistent, $G - C$ is also consistent, and thus there exists a consistent labeling L_2 of $G - C$. Consider the labeling L whose restrictions to C and $G - C$ are L_1 and L_2, respectively. Let T_L be the STP corresponding to L. T_L is consistent; otherwise, according to Lemma 3.19 it constitutes an invalid cycle. This cycle must be entirely contained in either C or $G - C$, thus contradicting the consistency of either L_1 or L_2. Let M_L be the minimal network of T_L. The distance graph of T_L shows that $(M_L)_{ij} = (M_{L_1})_{ij}$, because the shortest paths lengths within C are not affected by the edges of $G - C$. Hence, $v \in (M_L)_{ij}$, thus $v \in M_{ij}$. $\square$

Theorem 3.26 suggests an efficient algorithm for determining consistency and computing the minimal network of a general network. We first find the nonseparable components $C_1, \ldots, C_m$, and then solve each one of them independently. If all the components are found to be consistent, then the entire network is consistent and the minimal networks of the individual components coincide with the overall minimal network. If we use backtracking to solve each of the components, then the worst time complexity of this method is $O(nr^3k^c)$, where r and c denote the largest number of nodes and the largest number of edges in any component, respectively, and k, as before, denotes the maximum number of intervals labeling any edge in the graph. When the topology of any component admits a special, more efficient algorithm, that algorithm can be applied directly to that component without affecting the solution of the rest of the problem.

We still must find the minimal constraints on pairs that reside in two different components. This will be determined by Theorem 3.27, after demonstrating how a solution can be constructed to a given TCSP T. We start by finding a solution to the nonseparable component C_0 that contains node 0. All the separation vertices that are connected to C_0 in the superstructure $\overline{G}$ are instantiated. Next, choosing an instantiated separation vertex i, we find a solution to any nonseparable component C_i that is connected to i in $\overline{G}$ and whose vertices have not been instantiated yet. We continue in this fashion until all the variables are instantiated. Since $\overline{G}$ is a tree, we are guaranteed that once a partial solution of some component has been established, it does not need to be revised.

Theorem 3.27 *Let $G = (V, E)$ be the constraint graph of a given TCSP. Let i and j be two nodes that reside in different nonseparable components of G, namely*

$i \in C_i$ and $j \in C_j$. Let P be the unique path

$$P : C_i = C_{i_1}, i_1, C_{i_2}, i_2, \ldots, i_k, C_{i_{k+1}} = C_j,$$

that connects C_i and C_j in the superstructure of G. Then,

$$M_{ij} = M_{i,i_1} \otimes M_{i_1,i_2} \otimes \cdots \otimes M_{i_{k-1},i_k} \otimes M_{i_k,j}. \tag{3.15}$$

Proof It suffices to show that

$$M_{i,i_1} \otimes M_{i_1,i_2} \otimes \cdots \otimes M_{i_{k-1},i_k} \otimes M_{i_k,j} \subseteq M_{ij}.$$

Let

$$v \in M_{i,i_1} \otimes M_{i_1,i_2} \otimes \cdots \otimes M_{i_{k-1},i_k} \otimes M_{i_k,j}.$$

By definition of the composition operation, there exists a sequence of values $v_0, \ldots, v_k$ such that $v_0 \in M_{i,i_1}$, $v_j \in M_{i_j,i_{j+1}}$ for $j = 1, \ldots, k-1$, $v_k \in M_{i_k,j}$, and

$$\sum_{j=0}^{k} v_j = v.$$

By the minimality of the individual minimal networks, we can construct a solution $X = (x_1, \ldots, x_n)$ that satisfies

$$x_{i_1} - x_i = v_0,$$

$$x_{i_{j+1}} - x_{i_j} = v_j,$$

for $j = 1, \ldots, k-1$, and

$$x_j - x_{i_k} = v_k.$$

Hence,

$$x_j - x_i = v,$$

and thus $v \in M_{ij}$. □

The cost of computing a minimal constraint M_{ij}, using the above method, is $O(k^{cd})$, where c is the size of the largest component that resides along the path connecting C_i and C_j, and d is the length of that path. An alternative upper bound is given by $O(k^e)$. Thus, a full recovery of the minimal network costs $O[n^2 k^{\min(cn,e)}]$.

Illustration Consider the network of Example 3.13 (Figure 3.7). There are two nonseparable components: $C_1 = \{0, 1\}$ and $C_2 = \{1, 2, 3\}$. Component C_1 is a tree and thus already minimal. To compute the minimal network of C_2, we can either apply path consistency (note that C_2 is series-parallel with respect to any pair of nodes) or solve the two possible labelings separately. If M is the minimal network, then, by Theorem 3.26,

$$M_{01} = Z_{01}, M_{12} = Z_{12}, M_{13} = Z_{13}, M_{23} = Z_{23}, \tag{3.16}$$

where Z_{ij} are constraints taken from the minimal networks of the components. The rest of the network can be computed using Equation (3.15):

$$\begin{aligned} M_{02} &= M_{01} \otimes M_{12}, \\ M_{03} &= M_{01} \otimes M_{13}. \end{aligned} \tag{3.17}$$

Recall that in this example path consistency does compute the minimal network (see Section 3.4). This phenomenon can be explained by Theorems 3.26 and 3.27. We have already noted that path consistency computes the minimal networks of both components. We now show that, in general, this should suffice for computing the minimal constraints on edges that go across components. When path consistency terminates, the computed constraints T_{ij} satisfy

$$T_{02} \subseteq T_{01} \otimes T_{12},$$

and

$$T_{03} \subseteq T_{01} \otimes T_{13}.$$

Together with Equations (3.16) and (3.17), we get

$$T_{02} \subseteq M_{02},$$

$$T_{03} \subseteq M_{03}.$$

Since M is minimal, $T_{02} = M_{02}$ and $T_{03} = M_{03}$; namely, path consistency computes the full minimal network. □

Finally, we note that another network-based approach for solving general CSPs, the *cycle-cutset method* [18], cannot be employed beneficially in temporal problems. The reason is that the backtracking used in the solution of TCSPs instantiates arcs, rather than variables, and such instantiations do not decompose the original network.

3.6 Relations to Other Formalisms

In this section we relate the TCSP model to two other models of temporal reasoning—Allen's IA and Vilain and Kautz's PA. We show how the constraints in these representation schemes can be encoded within the TCSP model. To facilitate such encoding, we allow the interval representation of our constraints to include open and semi-open intervals, with the obvious effect on the definitions of the union and intersection operations. Similarly, an interval that results from a composition operation may be open on one side or on both sides, depending on the operands. For example,

$$\{[1,2],(6,8)\} \otimes \{[0,3),(12,15]\} = \{[1,5),(6,11),(13,17],(18,23)\}.$$

It is easy to verify that all our theorems still hold with this extended provision.

Any constraint network in Vilain and Kautz's PA is a special case of a TCSP lacking metric information. Recall (see Chapter 2), that it can be viewed as a CSP involving a set of variables $\{X_1, \ldots, X_n\}$ and binary constraints of the form $X_i \; R \; X_j$, where

$$R \in \{<, \leq, >, \geq, =, \neq\}. \tag{3.18}$$

Translating a PA network into a TCSP is straightforward. Constraints of the form $X_j < X_i$ and $X_j \leq X_i$ are expressed by the interval representations $T_{ij} = \{(-\infty, 0)\}$ and $T_{ij} = \{(-\infty, 0]\}$, respectively. The constraint $X_i = X_j$ translates into $T_{ij} = \{[0]\}$. The only relation that needs to be represented by a disjunction is $X_i \neq X_j$, translated into $T_{ij} = \{(-\infty, 0), (0, \infty)\}$.

Vilain and Kautz have addressed the tasks of determining consistency and computing the minimal network for problems expressed in the PA. They suggested the use of path consistency for computing the minimal network, which turned out to be insufficient [66].

VanBeek [66] addressed a subset of the PA, called convex point algebra (CPA), which excludes $\neq$. He showed that CPA networks may be solved in time $O(n^3)$ by applying path consistency. This follows immediately from the TCSP representation, since every CPA network is equivalent to an STP with edges labeled by intervals from the set

$$\{(-\infty, 0), (-\infty, 0], [0], [0, \infty), (0, \infty)\}. \tag{3.19}$$

Thus, when the constraints are taken from Equation (3.19), path consistency for TCSPs coincides with path consistency for CPA networks. Moreover, algorithms devised for solving STPs' tasks reduce to equivalent, often simpler algorithms for solving the same tasks in CPA networks. For example, directional path consistency can determine consistency in CPA networks in $O(nW^*(d)^2)$ operations, which amounts to linear time when $W^*(d)$ is bounded.

The full PA, including the $\neq$ relation, translates into TCSPs with disjunctions, for which our general methods can be applied and the special structure of the constraints exploited. In [39] and [66], it is shown that enforcing 3- and 4-consistency suffices for deciding consistency and for computing the minimal network, respectively, in PA networks (see Chapter 2). These results take special advantage of the non-metric nature of the relations in Equation (3.18).

In contrast, IA networks cannot be translated into binary TCSPs. Recall that an IA network can be viewed as a CSP involving a set of variables $X_1, \ldots, X_n$ whose domains are pairs of time points, representing the beginning and ending times of events intervals. The constraints are IA relations. Unfortunately, the translation of an IA network into a TCSP introduces nonbinary constraints (see Example 2.4) that cannot be encoded as a binary TCSP.

Problems involving higher-order constraints can be expressed as disjunctions of STPs, and solutions can be assembled by taking the union of the individual solutions. Although the number of such subproblems may be large, advantage can be taken of the simple procedures available for solving each STP. It seems likely, however, that unless metric constraints are specified, the representation suggested in [2] can be handled more conveniently.

CHAPTER 4

General Networks: Combining Qualitative and Quantitative Constraints

In the previous chapters we discussed several constraint-based formalisms for temporal reasoning: Allen's interval algebra, Vilain and Kautz's point algebra, and metric networks (the TCSP model). In these formalisms, temporal reasoning tasks are formulated as constraint satisfaction problems, where the variables are temporal objects such as points and intervals, and temporal statements are viewed as constraints on the location of these objects along the time line. Unfortunately, none of these formalisms can conveniently handle all forms of temporal knowledge. The qualitative approaches, Allen's interval algebra and Vilain and Kautz's point algebra, have difficulties in representing and reasoning about metric, numerical information, while metric networks exhibit limited expressiveness when it comes to qualitative information.

In this chapter we offer a general, constraint-based computational model for temporal reasoning that is capable of handling both qualitative and quantitative information. In this model, variables represent both points and intervals (as opposed to existing formalisms, where one has to commit to a single type of objects), and constraints may be either metric (between points) or qualitative, disjunctive relations (between temporal objects).

The main contribution of this model lies in providing a formal unifying framework for temporal reasoning, thereby generalizing the interval algebra, point algebra, and metric networks formalisms. This model facilitates the representation and processing of both all types of qualitative constraints considered in the literature to date and the metric constraints of Chapter 3.

This chapter is organized as follows. Section 4.1 describes the representation language and the constraint types under consideration. The definitions of the new model are given in Section 4.2. Section 4.3 reviews and extends the hierarchy of qualitative networks. Section 4.4 discusses new tractable classes of problems, called *augmented qualitative networks*—qualitative networks augmented by quantitative domain constraints. Section 4.5 presents two methods for solving general networks: a decomposition scheme and path consistency, and identifies a class of networks for which path consistency is exact. Section 4.6 relates our model to another general approach to temporal reasoning by Kautz and Ladkin [35].

4.1 The Representation Language

In this section we formally define the constraint types considered in our model. We shall demonstrate the new concepts using Example 1.1 (see Chapter 1).

4.1.1 Qualitative Constraints

A qualitative constraint between two objects O_i and O_j, each of which may be a point or an interval, is a disjunction of the form

$$(O_i\ r_1\ O_j) \vee \cdots \vee (O_i\ r_k\ O_j), \tag{4.1}$$

where each of the r_i's is a *basic relation* that may exist between the two objects. There are three types of basic relations.

- Basic *Interval-Interval (II) relations* that can hold between a pair of intervals [2]—*before, meets, starts, during, finishes, overlaps*, their inverses, and the equality relation, a total of 13 relations, denoted by the set $\{b, m, s, d, f, o, bi, mi, si, di, fi, oi, =\}$.
- Basic *Point-Point (PP) relations* that can hold between a pair of points [72], denoted by the set $\{<, =, >\}$.
- Basic *Point-Interval (PI) relations* that can hold between a point and an interval, and basic *Interval-Point (IP) relations* that can hold between an interval and a point. These relations are shown in Figure 4.1 and in Table 4.1 (see also [71, 39]).

A subset of basic relations (of the same type) corresponds to an ambiguous, disjunctive relationship between objects. For example, Equation (4.1) may also be written as $O_i\ \{r_1, \ldots, r_k\}\ O_j$; alternatively, we say that the constraint between O_i and O_j is the relation set $\{r_1, \ldots, r_k\}$. Consider, for instance, Example 1.1.

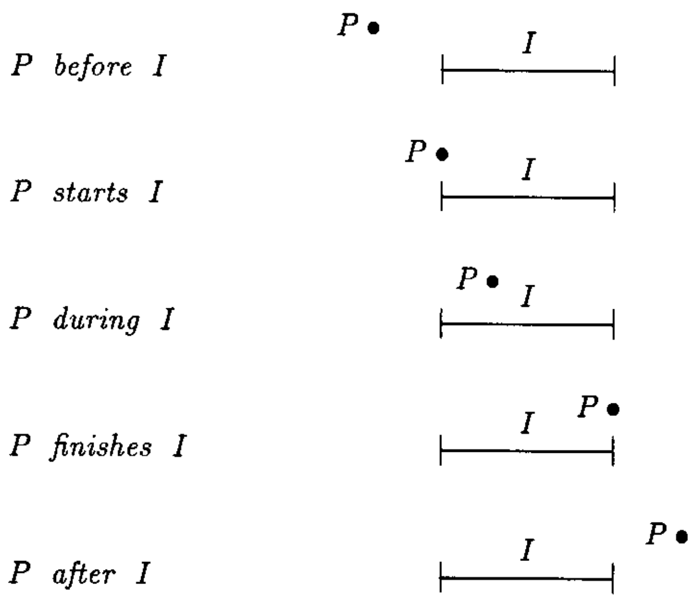

Figure 4.1: The basic relations between a point P and an interval I.

One qualitative constraint given in this example reflects the fact that John and Fred met at a traffic light. It is expressed by an II relation specifying that intervals J and F are not disjoint:

$$J \; \{s, si, d, di, f, fi, o, oi, =\} \; F.$$

To facilitate the processing of qualitative constraints, we define a *qualitative algebra(QA)*, whose elements are all legal constraints (all subsets of basic relations of the same type)—2^{13} II relations, 2^3 PP relations, 2^5 PI relations, and 2^5 IP relations. Two binary operations are defined on these elements: intersection and

Relation	Symbol	Inverse	Relations on Endpoints
P before I	b	bi	$P < I^-$
P starts I	s	si	$P = I^-$
P during I	d	di	$I^- < P < I^+$
P finishes I	f	fi	$P = I^+$
P after I	a	ai	$P > I^+$

Table 4.1: The basic relations between a point P and an interval $I = [I^-, I^+]$.

	PP	PI	IP	II
PP	$[T_{PA}]$	$[T_1]$	$[\emptyset]$	$[\emptyset]$
PI	$[\emptyset]$	$[\emptyset]$	$[T_2]$	$[T_4]$
IP	$[T_1]^t$	$[T_3]$	$[\emptyset]$	$[\emptyset]$
II	$[\emptyset]$	$[\emptyset]$	$[T_4]^t$	$[T_{IA}]$

Table 4.2: A full transitivity table.

composition. The *intersection* of two qualitative constraints, R' and R'', denoted by $R' \oplus R''$, is the set-theoretic intersection $R' \cap R''$. The *composition* of two constraints, R' between objects O_i and O_j, and R'' between objects O_j and O_k, is a new relation between objects O_i and O_k, induced by R' and R''. Formally, the composition of R' and R'', denoted by $R' \otimes R''$, is the composition of the constituent basic relations, namely,

$$R' \otimes R'' = \{r' \otimes r'' | r' \in R', r'' \in R''\}.$$

Composition of two basic relations, r' and r'', is defined by a *transitivity table*, shown in Table 4.2. Six transitivity tables, $T_1, \ldots, T_4, T_{PA}, T_{IA}$, are required; each defines the composition of basic relations of a certain type. For example, composition of a basic PP relation and a basic PI relation is defined as transitivity table T_1. Two important subsets of QA are Allen's interval algebra (IA), the restriction of QA to II relations, and Vilain and Kautz's point algebra (PA), its restriction to PP relations. The corresponding transitivity tables are given in [2] and [72] (see also Chapter 2), and appear in Table 4.2 as T_{IA} and T_{PA}, respectively. The rest of the transitivity tables are shown in Tables 4.3–4.6.[1] Illegal combinations in Table 4.2 are denoted by $\emptyset$.

4.1.2 Quantitative Constraints

Quantitative constraints refer to absolute location or the distance between *points*. We consider two types of quantitative constraints (those described in Chapter 3):

[1]In these tables, ? refers to subsets that contain all basic relations: for example, $\{<, =, >\}$ for PP relations.

T_1	b	s	d	f	a
$<$	b	b	b s d	b s d	?
$=$	b	s	d	f	a
$>$	?	d f a	d f a	a	a

Table 4.3: Composition of PP and PI relations.

T_2	ai	fi	di	si	bi
b	$<$	$<$	$<$	$<$	?
s	$<$	$<$	$<$	$=$	$>$
d	$<$	$<$	?	$>$	$>$
f	$<$	$=$	$>$	$>$	$>$
a	?	$>$	$>$	$>$	$>$

Table 4.4: Composition of PI and IP relations.

T_3	b	s	d	f	a
ai	b	b	b m o d s	b m o d s	?
fi	b	m	o s d	f fi =	a mi oi si di
di	b m o di fi	o di fi	o oi = s d f si di fi	oi di si	a mi oi si di
si	b m o di fi	s si =	oi d f	mi	a
bi	?	a mi oi d f	a mi oi d f	a	a

Table 4.5: Composition of IP and PI relations.

T_4	b	a	d	di	o	oi	m	mi	s	si	f	fi	=
b	b	?	b s d	b	b	b s d	b	b s d	b	b	b s d	b	b
s	b	a	d	b	b	d	b	f	s	s	d	b	s
d	b	a	d	?	b s d	d f a	b	a	d	d f a	d	b s d	d
f	b	a	d	a	d	a	s	a	d	a	f	f	f
a	?	a	d f a	a	d f a	a	d f a	a	d f a	a	a	a	a

Table 4.6: Composition of PI and II relations.

- A *unary* constraint, on point P_i, restricts the location of P_i to a given set of intervals:
$$(P_i \in I_1) \vee \cdots \vee (P_i \in I_k).$$
- A *binary* constraint, between points P_i and P_j, constrains the permissible values for the distance $P_j - P_i$:
$$(P_j - P_i \in I_1) \vee \cdots \vee (P_j - P_i \in I_k).$$

In both cases the constraint is represented by a set of intervals $\{I_1, \ldots, I_k\}$; each interval may be open or closed in either side.[2] For instance, one binary constraint given in Example 1.1 specifies the duration of interval J (the event "John was going to work"):
$$P_2 - P_1 \in \{(0,20),(60,\infty)\}.$$
The fact that John left home between 7:05–7:10 a.m. is translated into a unary constraint on P_1: $P_1 \in \{(5,10)\}$, or $5 < P_1 < 10$ (note that all times are relative to P_0, namely, 7:00 a.m.). Sometimes it is easier to treat a unary constraint on P_i as a binary constraint between P_0 and P_i, which has the same interval representation. For example, the above unary constraint is equivalent to the binary constraint, $P_1 - P_0 \in \{(5,10)\}$.

[2]The set $\{I_1, \ldots, I_k\}$ represents the set of real numbers $I_1 \cup \cdots \cup I_k$. Throughout this chapter we shall use the convention whereby a real number v is in $\{I_1, \ldots, I_k\}$ if and only if $v \in I_1 \cup \cdots \cup I_k$.

The intersection and composition operations for quantitative constraints assume the following form. Let C' and C'' be quantitative constraints, represented by interval sets I' and I'', respectively. Then, their intersection is defined as

$$C' \oplus C'' = \{x | x \in I', x \in I''\}.$$

The composition of C' and C'' is defined as

$$C' \otimes C'' = \{z | \exists x \in I', \exists y \in I'',\ x + y = z\}.$$

Illustration Let $C_1 = \{[1,4),(6,8)\}$ and $C_2 = \{(0,1],(3,5),[6,7]\}$. Then,

$$C_1 \oplus C_2 = \{[1],(3,4),(6,7]\}.$$

Let $C_3 = \{[1,2],(6,8)\}$ and $C_4 = \{[0,3),(12,15]\}$. Then,

$$C_3 \otimes C_4 = \{[1,5),(6,11),(13,17],(18,23)\}.$$

4.1.3 Relationships Between Qualitative and Quantitative Constraints

The existence of a constraint of one type sometimes implies the existence of an implicit constraint of the other type. This can only occur when the constraint involves two points. Consider a pair of points P_i and P_j. If a quantitative constraint, C, between P_i and P_j is given (by an interval set $\{I_1, \ldots, I_k\}$), then the implied qualitative constraint, $\mathsf{QUAL}(C)$, is defined as follows (see also [35]).

- If $0 \in \{I_1, \ldots, I_k\}$, then $= \in \mathsf{QUAL}(C)$.
- If there exists a value $v > 0$ such that $v \in \{I_1, \ldots, I_k\}$, then $< \in \mathsf{QUAL}(C)$.
- If there exists a value $v < 0$ such that $v \in \{I_1, \ldots, I_k\}$, then $> \in \mathsf{QUAL}(C)$.

Similarly, if a qualitative constraint, C, between P_i and P_j is given (by a relation set R), then the implied quantitative constraint, $\mathsf{QUAN}(C)$, is defined as follows.

- If $< \in R$, then $(0, \infty) \in \mathsf{QUAN}(C)$.
- If $= \in R$, then $[0] \in \mathsf{QUAN}(C)$.
- If $> \in R$, then $(-\infty, 0) \in \mathsf{QUAN}(C)$.

C	$\mathsf{QUAN}(C)$
$<$	$(0, \infty)$
$\leq$	$[0, \infty)$
$=$	$[0]$
$>$	$(-\infty, 0)$
$\geq$	$(-\infty, 0]$
$\neq$	$(-\infty, 0), (0, \infty)$
$?$	$(-\infty, \infty)$

Table 4.7: The QUAN translation.

An alternative definition of QUAN is given in Table 4.7.

The intersection and composition operations can be extended to cases where the operands are constraints of different types. If C' is a quantitative constraint and C'' is qualitative, then intersection is defined as quantitative intersection:

$$C' \oplus C'' = C' \oplus \mathsf{QUAN}(C''). \tag{4.2}$$

Composition, on the other hand, depends on the type of C''.

- If C'' is a PP relation, then composition (and consequently the resulting constraint) is quantitative:

$$C' \otimes C'' = C' \otimes \mathsf{QUAN}(C'').$$

- If C'' is a PI relation, then composition is qualitative:

$$C' \otimes C'' = \mathsf{QUAL}(C') \otimes C''.$$

Illustration Let $C_1 = \{(0,3)\}$ be a quantitative constraint, $C_2 = \{<,=\}$ be a PP relation, and $C_3 = \{b, d\}$ be a PI relation. Then,

$$C_1 \otimes C_2 = \{(0,3)\} \otimes \{[0,\infty)\} = \{(0,\infty)\},$$

and

$$C_1 \otimes C_3 = \{<\} \otimes \{b, d\} = \{\text{b,s,d}\}.$$

4.2 General Temporal Constraint Networks

We now present a network-based model that facilitates the processing of all constraints described in the previous section. The definitions of the new model follow closely those developed for discrete constraint networks [50] and for metric networks (Chapter 3).

A *general temporal constraint network* involves a set of variables $\{X_1, \ldots, X_n\}$, each representing a temporal object (a point or an interval), and a set of unary and binary constraints. When a variable represents a time point, its domain is the set of real numbers $\Re$. When a variable represents a temporal interval, its domain is the set of ordered pairs of real numbers, namely, $\{(a,b)|a,b \in \Re, a < b\}$. Constraints may be quantitative or qualitative. Each qualitative constraint is represented by a relation set R. Each quantitative constraint is represented by an interval set I. Constraints between variables representing points are always maintained in their quantitative form. We also assume that unary quantitative constraints are represented by equivalent binary constraints, as shown in the previous section. A set of internal constraints relates each interval $I = [I^-, I^+]$ to its endpoints I^- $\{$*starts*$\}$ I and I^+ $\{$*finishes*$\}$ I.

A constraint network is associated with a *directed constraint graph*, where nodes represent variables and an arc $i \rightarrow j$ indicates that a constraint C_{ij}, between variables X_i and X_j, is specified. The arc is labeled by an interval set (when the constraint is quantitative) or by a QA element (when it is qualitative). We assume that whenever a constraint C_{ij} is given, the inverse constraint C_{ji} is also provided; however, in the constraint graph only one of these will be shown. The constraint graph of Example 1.1 is shown in Figure 4.2.

A tuple $X = (x_1, \ldots, x_n)$ is called a *solution* if the assignment $\{X_1 = x_1, \ldots, X_n = x_n\}$ satisfies all the constraints (note that the value assigned to a variable that represents an interval is a pair of real numbers). It corresponds to a *feasible scenario*—an arrangement of the temporal objects along the time line in a way that is consistent with the given information. The network is *consistent* if at least one solution exists. A value v is a *feasible value* for variable X_i if there exists a solution in which $X_i = v$. The set of all feasible values of a variable is called its *minimal domain*.

We define a partial order $\subseteq$ among binary constraints of the *same type*. A constraint C' is *tighter* than constraint C'', denoted by $C' \subseteq C''$, if every pair

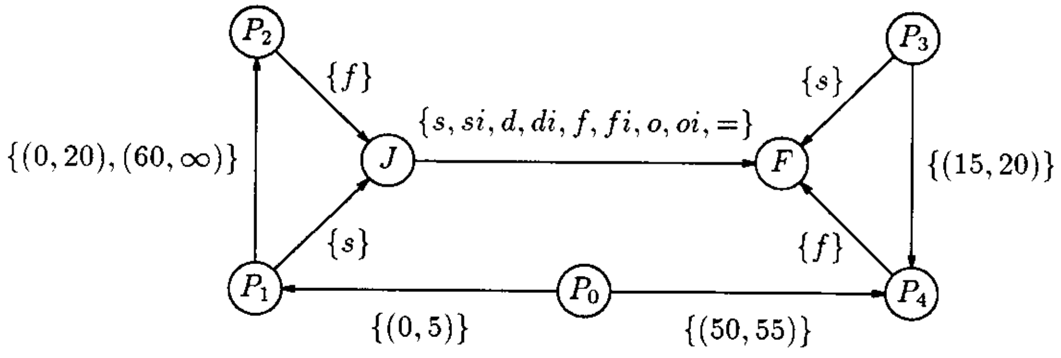

Figure 4.2: The constraint graph of Example 1.1.

of values allowed by C' is also allowed by C''. If C' and C'' are qualitative, represented by relation sets R' and R'', respectively, then $C' \subseteq C''$ if and only if $R' \subseteq R''$. If C' and C'' are quantitative, represented by interval sets I' and I'', respectively, then $C' \subseteq C''$ if and only if for every value $v \in I'$, we have also $v \in I''$. This partial order can be extended to networks in the usual way. A network N' is tighter than network N'', if the partial order $\subseteq$ is satisfied for all the corresponding constraints. Two networks are *equivalent* if they possess the same solution set. A network may have many equivalent representations; in particular, there is a unique equivalent network M, which is minimal with respect to $\subseteq$, called the *minimal network* (the minimal network is unique because equivalent networks are closed under intersection). The arc constraints specified by M are called the *minimal constraints*.

The minimal network is an effective, more explicit encoding of the given knowledge. Consider, for instance, the minimal network of Example 1.1, whose constraints are shown in Table 4.8. The minimal constraint between P_1 and P_2 is $\{(60,\infty)\}$, the minimal constraint between P_0 and P_2 is $\{(65,\infty)\}$, and the minimal constraint between P_0 and P_3 is $\{(30,40)\}$. From this minimal network representation, we can infer that today John was working in the main office; he arrived at work after 8:05 a.m., while Fred arrived at work between 7:30–7:40 a.m. A feasible scenario, which can be easily constructed from the minimal network representation, is shown in Figure 4.3.

Given a network N, the first interesting task is to determine its consistency. If the network is consistent, we are interested in other reasoning tasks, such as computing a solution to N, the minimal domain of a given variable X_i, the

	P_0	P_1	P_2	P_3	P_4	J	F
P_0	$[0]$	$(5, 10)$	$(65, \infty)$	$(30, 40)$	$(50, 55)$	b	b
P_1	$(-10, -5)$	$[0]$	$(60, \infty)$	$(20, 35)$	$(40, 50)$	s	b
P_2	$(-\infty, -65)$	$(-\infty, -60)$	$[0]$	$(-\infty, -25)$	$(-\infty, -10)$	f	a
P_3	$(-40, -30)$	$(-35, -20)$	$(25, \infty)$	$[0]$	$(15, 20)$	d	s
P_4	$(-55, -50)$	$(-50, -40)$	$(10, \infty)$	$(-20, -15)$	$[0]$	d	f
J	bi	si	fi	di	di	=	di
F	bi	bi	ai	si	fi	d	=

Table 4.8: The minimal network of Example 1.1.

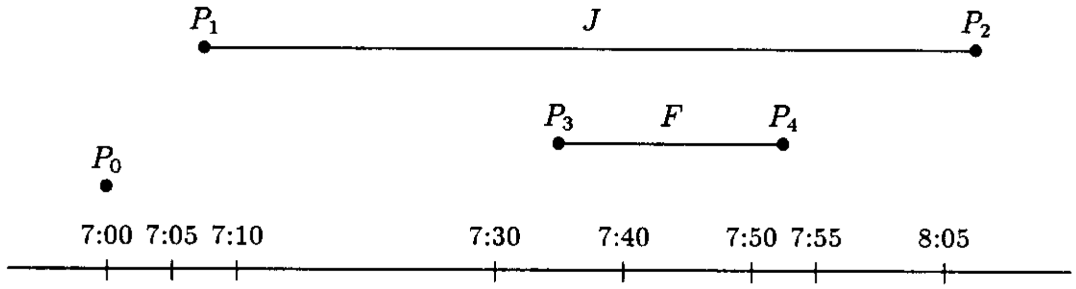

Figure 4.3: A feasible scenario.

minimal constraint between a given pair of variables X_i and X_j, and the full minimal network. The rest of this chapter is concerned with solving these tasks.

Solving any of the above tasks for a general network is difficult. Even the simplest task, deciding consistency, is NP-hard. This follows trivially from the fact that deciding consistency for either metric networks or IA networks is NP-hard (Chapter 3, Vilain and Kautz [72]). Therefore, it is unlikely that there exists a general polynomial-time algorithm for deciding the consistency of a network, and consequently for solving the other tasks. Hence, we settle for the following alternatives. In Sections 4.3 and 4.4 we pursue "islands of tractability"—special classes of networks that admit polynomial solution. Then, in Section 4.5, we describe brute-force, exponential techniques that can handle any general network, and discuss the use of *path consistency* as an approximation scheme.

4.3 The Hierarchy of Qualitative Networks

We wish to find tractable classes of general networks, namely networks containing both qualitative and quantitative constraints. We shall form such networks by adding metric constraints to certain classes of qualitative networks. Of course, in our quest for tractability it would make sense to concentrate only on *tractable* qualitative networks. As the first step in this direction, we discuss in this section the computational complexity of solving qualitative networks. We briefly describe the qualitative networks hierarchy and then draw the line between tractable and intractable networks. In Section 4.4 we show how the tractable classes—*CPA networks* and *PA networks*—can be augmented by various quantitative constraints to obtain new tractable classes.

Consider a qualitative network G. If all constraints are II relations (namely IA elements) or PP relations (PA elements), then the network is called an *IA network* or a *PA network*, respectively [67]. If all constraints are PI and IP relations, then the network is called an *Interval-Point Algebra (IPA) network.*[3] A special case of a PA network, where the relations are convex (taken only from $\{<, \leq, =, \geq, >\}$, i.e., excluding $\neq$), is called a *convex PA (CPA) network.*

It can easily be shown that any qualitative network can be represented by an IA network. On the other hand, some qualitative networks cannot be represented by a PA network, such as (see [72]) a network consisting of two intervals I and J and a single constraint between them I $\{$*before*, *after*$\}$ J. Formally, the following relationship can be established among qualitative networks.

Proposition 4.1 *Let QN be the set of all qualitative networks. Let $net(CPA)$, $net(PA)$, $net(IPA)$, and $net(IA)$ denote the set of qualitative networks that can be represented by CPA networks, PA networks, IPA networks, and IA networks, respectively. Then,*

$$net(CPA) \subset net(PA) \subset net(IPA) \subset net(IA) = QN.$$

Proof Trivial. □

[3]We use this name to comply with the names IA and PA, although technically these relations, together with the intersection and composition operations, do not constitute an algebra, because they are not closed under composition.

Remark 4.2 Clearly, any CPA network is in *net*(CPA). On the other hand, *net*(CPA) contains some qualitative networks that are not CPA networks; for example, the IA network I {*starts*, *during*, *finishes*, *equal*} J. Therefore, the CPA networks are strictly contained in *net*(CPA). Similarly, the PA, IPA, and IA networks are contained in *net*(PA), *net*(IPA), and *net*(IA), respectively. □

By moving up the qualitative networks hierarchy from CPA networks towards IA networks we gain expressiveness, but at the same time lose tractability. For example, deciding the consistency of a PA network can be done in time $O(n^2)$ [68, 48], but it becomes NP-complete for IA networks [72], or even for IPA networks, as stated in the following theorem.

Theorem 4.3 *Deciding the consistency of an IPA network is NP-hard.*

Proof Reduction from the *betweenness* problem, which is defined as follows [27].
Instance: Finite set A, collection C of ordered triplets (a, b, c) of distinct elements from A.
Question: Is there a one-to-one function $f : A \to \{1, 2, \ldots, |A|\}$ such that for each $(a, b, c) \in C$, we have either $f(a) < f(b) < f(c)$ or $f(c) < f(b) < f(a)$?

Consider an instance of *betweenness*. We construct an IPA network in the following way. Each element $a \in A$ is associated with a unique point P_a. For each triplet $(a, b, c) \in C$, we create an interval I_{abc}, and impose the constraints

$$P_a \ \{starts, finishes\} \ I_{abc}$$

$$P_c \ \{starts, finishes\} \ I_{abc}$$

$$P_b \ \{during\} \ I_{abc}.$$

In addition, we force all points to be distinct. For each pair of elements $(a, b) \in A$, we create an interval I_{ab}, and impose the constraints

$$P_a \ \{starts, during, finishes\} \ I_{ab}$$

$$P_b \ \{before, after\} \ I_{ab},$$

forcing $P_a \neq P_b$. Clearly, this network is consistent if and only if the answer to the given betweenness problem is YES. □

Other reasoning tasks are usually harder than deciding consistency. Thus, it is unlikely that any task in IPA or IA networks can be solved in polynomial time. This suggests that the line between tractable and intractable qualitative networks can be drawn somewhere between PA and IPA networks. Consequently, we shall focus our search for new tractable classes on extending CPA and PA networks.

4.4 Augmented Qualitative Networks

In this section we consider the simplest type of network having both qualitative and quantitative constraints, an *augmented qualitative network*. It is a qualitative network—a CPA network or a PA network—augmented by unary constraints on its domains.

We shall consider CPA and PA networks over three domain classes, each of importance in temporal reasoning applications:

1. *Discrete domains*, where each variable may assume only a finite number of values. For instance, when we settle for crude timing of events, such as the day or the year in which they occurred.

2. *Single-interval domains*, where we have only an upper and/or a lower bound on the timing of events. We shall also consider *almost-single-interval domains*, where each domain consists of a single interval, from which a finite set of values, called *holes*, may be excluded.

3. *Multiple-intervals domains*. This case subsumes the two previous cases.[4]

Illustration A CPA network over multiple-intervals domains is depicted in Figure 4.4, where each variable is labeled by its domain intervals. Note that in this example, as well as throughout the rest of this section, we express the domain constraints as *unary constraints*. □

Let us consider in detail the representation of the domains.

When the domains are discrete, a domain D_i of a variable X_i consists of a set of up to k values $\{v_1, \ldots, v_k\}$, where $v_1 < \cdots < v_k$. It is represented as an array

[4]Note that a discrete domain $\{v_1, \ldots, v_k\}$ is essentially a multiple-intervals domain $\{[v_1, v_1], \ldots, [v_k, v_k]\}$.

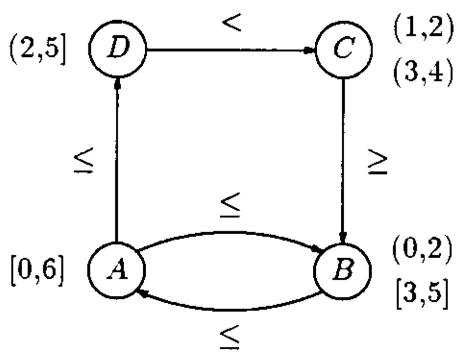

Figure 4.4: A CPA network over multiple-intervals domains.

of size k sorted in an ascending order. We also maintain two pointers, Inf and Sup, to $\inf(D_i) = v_1$ and $\sup(D_i) = v_k$, respectively.

When the domains are continuous, namely they consist of multiple intervals (or as a special case consist of a single interval or an almost-single interval), then a domain D_i is given by an interval set $I = \{I_1, \dots, I_k\}$, where $I_i = \{a_i, b_i\}$. The symbols $\{$ and $\}$ reflect the fact that each interval may be open or closed in either side. The domain D_i will be represented by the points $a_1, b_1, \dots, a_k, b_k$, which are called the *extreme points* of D_i. These extreme points are maintained in an array of size $2k$. In an accompanied array we maintain an indicator as to whether each extreme point is in the domain (i.e., whether the corresponding interval is open or closed). An interval I_i can be regarded as a set of real numbers, and thus its extreme points can be referred to as $a_i = \inf(I_i)$ and $b_i = \sup(I_i)$. Similarly, an interval set $I = \{I_1, \dots, I_k\}$ can be regarded as a set of real numbers consisting of the values in $I_1 \cup \dots \cup I_k$. Thus, we have $\inf(D_i) = \inf(I_1) = a_1$ and $\sup(D_i) = \sup(I_k) = b_k$. As with discrete domains, we shall keep two pointers, Inf and Sup, to $\inf(D_i) = v_1$ and $\sup(D_i) = v_k$, respectively.

We shall use three parameters in analyzing the computational complexity of algorithms: n—the number of nodes in the network, e—the number of arcs, and k—the maximum *domain size*, that is, the number of values in a domain (for discrete domains) or the number of intervals per domain (for continuous domains).

In the rest of this section we show that for augmented CPA networks and for some augmented PA networks, all interesting tasks can be solved in polynomial time using local consistency algorithms such as *arc consistency (AC)* and *path consistency (PC)*.

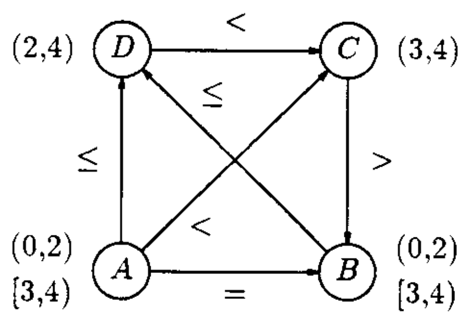

Figure 4.5: An arc- and path-consistent form of the network in Figure 4.4.

4.4.1 Arc and Path Consistency

Let us review the definitions of arc and path consistency [43, 50].

Definition 4.4 An arc $i \rightarrow j$ is *arc consistent* if and only if for any value $x \in D_i$, there is a value $y \in D_j$ such that the pair (x, y) satisfies the constraint C_{ij}. A *network* G is arc consistent if all its arcs are consistent.

Definition 4.5 A path P from i to j, $i_0 = i \rightarrow i_1 \rightarrow \cdots \rightarrow i_m = j$, is *path consistent* if the direct constraint C_{ij} is tighter than the composition of the constraints along P, namely

$$C_{ij} \subseteq C_{i_0,i_1} \otimes \cdots \otimes C_{i_{m-1},i_m}.$$

A *network* G is path consistent if all its paths are consistent.

Illustration Figure 4.5 shows an equivalent, arc- and path-consistent form of the network in Figure 4.4. □

Note that our definition of path consistency is slightly different than the original definition [43], since it disregards domain constraints. The following example illustrates the difference between the two definitions.

Example 4.6 Consider the network in Figure 4.6. The network is path consistent according to Definition 4.5, since the underlying qualitative network is path consistent. However, it is not path consistent according to the common definition (namely, 3-consistency), because the instantiation $A = 1, B = 1$ cannot be extended to C. □

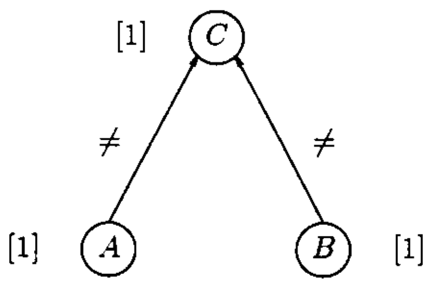

Figure 4.6: An augmented PA network.

Algorithm AC-3

```
1. $Q \leftarrow \{i \rightarrow j | i \rightarrow j \in E\}$
2. while $Q \neq \emptyset$ do
3.         select and delete any arc $k \rightarrow m$ from $Q$
4.         if REVISE($(k, m)$) then
5.            $Q \leftarrow Q \cup \{i \rightarrow k | i \rightarrow k \in E, i \neq m\}$
6. end
```

Figure 4.7: AC-3—an arc consistency algorithm.

The most common arc consistency algorithm that converts a network into an equivalent arc-consistent form is algorithm AC-3 [43], shown in Figure 4.7. AC-3 repeatedly applies the function REVISE((i, j)), which makes arc $i \rightarrow j$ consistent, until a fixed point, at which all arcs are consistent, is reached. The function REVISE restricts the domain D_i using quantitative operations on constraints[5]:

$$D_i \leftarrow D_i \oplus D_j \otimes \mathsf{QUAN}(C_{ji}). \tag{4.3}$$

It returns true if the domain D_i is changed.

In some cases we shall use a weaker version of arc consistency, called *directional arc consistency* [21].

[5]Note that Equation (4.3) is the temporal equivalent of Mackworth's REVISE, when the later is expressed using intersection and composition of discrete constraints: $D_i \leftarrow D_i \oplus D_j \otimes C_{ji}$.

```
Algorithm DAC

1. for $i := n$ downto 1 do
2.     for each arc $j \to i, j < i$ do
3.         X ← REVISE($(j,i)$)
4. end
```

Figure 4.8: DAC—a directional arc consistency algorithm.

Definition 4.7 (Dechter and Pearl [21]) Let G be a constraint network. Let d be an ordering of the nodes, namely, $i < j$ if and only if i precedes j in d. We say that G is *directional arc consistent* if all arcs directed along d are arc consistent.

Algorithm DAC [21], shown in Figure 4.8, converts a given network into an equivalent directional-arc-consistent form. Being weaker than full arc consistency, directional arc consistency can be enforced more efficiently, as we shall see later in this section.

A network can be converted into an equivalent path-consistent form by applying any path consistency algorithm to the underlying qualitative network [43, 72, 67]. Path consistency algorithms impose local consistency among triplets of variables (i, k, j) by using a relaxation operation:

$$C_{ij} \leftarrow C_{ij} \oplus C_{ik} \otimes C_{kj}. \tag{4.4}$$

Relaxation operations are applied until a fixed point is reached, or until some constraint becomes empty (which indicates an inconsistent network).

We shall use an efficient path consistency algorithm, PC-2 [43], shown in Figure 4.9. The function REVISE((i, k, j)) performs the relaxation operation of Equation (4.4) and returns **true** if the constraint C_{ij} is changed. Algorithm PC-2 runs to completion in $O(n^3)$ time [45].

4.4.2 The Precedence Graph

Many of the algorithms presented in this section make use of an auxiliary data structure, called a *precedence graph* (see also [48, 68]), which displays precedence

Algorithm PC-2

```
1. $Q \leftarrow \{(i,k,j) | (i < j), (k \neq i,j)\}$
2. **while** $Q \neq \emptyset$ **do**
3.     select and delete any triplet $(i,k,j)$ from $Q$
4.     **if** REVISE($(i,k,j)$) **then**
5.         $Q \leftarrow Q \cup$ RELATED-PATHS($(i,k,j)$)
6. **end**
```

Figure 4.9: PC-2—a path consistency algorithm.

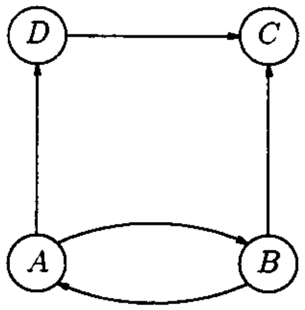

Figure 4.10: The precedence graph of the network in Figure 4.4.

relations between variables.

Definition 4.8 Let $G = (V, E)$ be a PA network. The *precedence graph* of G is a directed graph $G_p = (V, E_p)$, which has the same node set as G and whose edges are oriented in the following way.[6]

1. If C_{ij} is $<$ or $\leq$ then $i \rightarrow j \in E_p$.
2. If C_{ij} is $=$ then both $i \rightarrow j \in E_p$ and $j \rightarrow i \in E_p$.

Illustration The precedence graph of the network in Figure 4.4 is depicted in Figure 4.10. □

[6]Note that $\neq$ constraints are not reflected in the precedence graph.

The following theorem states a necessary and sufficient condition for the consistency of a PA network in terms of its precedence graph.

Theorem 4.9 (Van Beek [68]) *Let G be a given PA network, and let G_p be its precedence graph. Then, G is consistent if and only if for any pair of nodes i, j, that belong to the same strongly connected component*[7] *in G_p, $\{=\} \subseteq C_{ij}$.*

According to Theorem 4.9 we can decide the consistency of a PA network by finding the strongly connected components in its precedence graph and then testing whether all constraints satisfy the condition of Theorem 4.9 [68]. The complexity of this method is $O(e)$.

When solving augmented qualitative networks, we shall distinguish between networks having acyclic precedence graphs, called *acyclic* networks, and *cyclic* networks, which contain directed cycles; the former can be solved more efficiently than the latter. Specifically, in the next sections we shall show that for some tractable classes, acyclic networks can be solved using arc consistency, while cyclic networks can be solved using both arc and path consistency.

It turns out that any cyclic network G can be converted, in a quadratic time, into an equivalent acyclic representation, called a *reduced network*. The conversion scheme is based on the next lemma, which states an important property of the strongly connected components in the precedence graph.

Lemma 4.10 *Let $G = (V, E)$ be a nonempty path-consistent PA network. Let $G_p = (V, E_p)$ be the precedence graph of G. Nodes i and j belong to the same strongly connected component in G_p if and only if C_{ij} is $=$.*

Proof See Appendix A. □

It follows that, in any solution $X = (x_1, \ldots, x_n)$ to G, if nodes i and j belong to the same component in G_p, then $x_i = x_j$. This suggests that all nodes that belong to a common component C_i can be collapsed into a single representative node. The domain of this new node will be the intersection of all domains in C_i. This idea is expressed more formally in the following definition.

[7]Nodes i and j belong to the same *strongly connected component* if there exist directed paths from i to j and from j to i.

Definition 4.11 Let $G = (V, E)$ be an augmented PA network, having a *consistent* underlying qualitative network. Let $G_p = (V, E_p)$ be the precedence graph of G, and let $C_1, \ldots, C_m$ be the strongly connected components of G_p. The *reduced network* of G, $G^r = (V^r, E^r)$, is defined as follows.

- The nodes are the strongly connected components of G_p, namely, $V^r = \{C_1, \ldots, C_m\}$. The domain of node C_i in G^r, D_i^r, is the intersection of all domains of nodes in component C_i, namely,

$$D_i^r = \bigoplus_{j \in C_i} D_j. \tag{4.5}$$

- An edge $C_i \rightarrow C_j \in E^r$ if and only if there exists an edge $i \rightarrow j \in E_p$ such that $i \in C_i$ and $j \in C_j$. The constraint between nodes C_i and C_j in G^r, C_{ij}^r, is the intersection of all constraints between nodes in C_i and nodes in C_j, namely,

$$C_{ij}^r = \bigoplus_{k \in C_i, l \in C_j} C_{kl}. \tag{4.6}$$

Note that the intersection operations in Equations (4.5) and (4.6) may result in an empty domain or an empty constraint. This may occur only if the input network G is inconsistent.

Definition 4.11 requires that the underlying qualitative network is consistent. Thus, before constructing the reduced network, we first need to verify that G is consistent. This can be done in $O(e)$ time by testing the precedence graph according to the condition of Theorem 4.9. The construction of G^r itself is straightforward and can be accomplished in $O(n^2k)$ time. It involves $O(n)$ binary domain intersections (Equation (4.5)), because each node belongs to exactly one component, and $O(e)$ constraint intersections (Equation (4.6)), because each arc in G contributes to exactly one cross-component arc in G^r. The cost of a domain intersection is $O(nk)$. A constraint intersection takes a constant time. Hence, the total complexity is $O(n^2k)$.

The reduced network is an equivalent representation of the input network in the sense that there exists a one-to-one correspondence between the solution sets: any solution $X^r = (x_1^r, \ldots, x_m^r)$ to G^r corresponds to a solution $X = (x_1, \ldots, x_n)$ to G, in which all nodes that belong to a component C_i are assigned the value x_i^r, and vice versa. It also follows that the reduced network is consistent if and only if the input network is consistent.

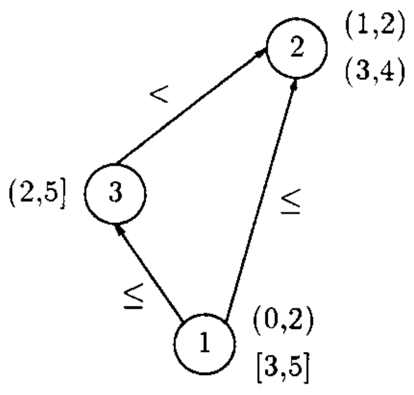

Figure 4.11: The reduced network of the network in Figure 4.4.

The main importance of the reduced network is that it is an *acyclic* representation of the input network. In the sequel, we shall take advantage of this fact in solving cyclic networks: we shall solve cyclic networks by applying techniques devised for acyclic networks to their reduced-network representation.

Illustration Consider the network in Figure 4.4. The strongly connected components in its precedence graph (shown in Figure 4.10) are $C_1 = \{A, B\}$, $C_2 = \{C\}$, and $C_3 = \{D\}$. The reduced network is shown in Figure 4.11, where component C_i is represented by node i. One solution of the reduced network is the tuple

$$(C_1 = 1, C_2 = 3.5, C_3 = 3).$$

It corresponds to the solution

$$(A = 1, B = 1, C = 3.5, D = 3)$$

of the original network. □

We conclude the discussion of the precedence graph by considering the special case of arc- and path-consistent networks.

Proposition 4.12 *Any nonempty path-consistent PA network is consistent.*

Proof By Theorem 4.9 and Lemma 4.10.[8] □

Lemma 4.13 *Let $G = (V, E)$ be a nonempty path-consistent PA network. Let $G_p = (V, E_p)$ be the precedence graph of G. Let C' and C'' ($C' \neq C''$) be two*

[8]Another proof is given by Ladkin and Maddux in [39]

strongly connected components in G_p. If $i \rightarrow j \in E$ and $k \rightarrow l \in E$, where $i, k \in C'$ and $j, l \in C''$, then $C_{ij} = C_{kl}$.

> **Proof** See Appendix A. □

From Lemma 4.10 we have the following corollary.

Corollary 4.14 *Let G be a nonempty arc- and path-consistent augmented PA network. Let G_p be the precedence graph of G. If nodes i and j belong to the same strongly connected component in G_p, then $D_i = D_j$.*

Using Lemma 4.12, Lemma 4.13, and Corollary 4.14, we obtain the following properties of the reduced network of an arc- and path-consistent PA network.

Lemma 4.15 *The reduced network of a nonempty arc- and path-consistent augmented PA network is (1) nonempty and (2) arc and path consistent.*

> **Proof** From Lemma 4.12, the underlying qualitative network is consistent. From Lemma 4.13 and Corollary 4.14, we have (1) and (2). □

In addition, when constructing the reduced network of an arc- and path-consistent network, instead of performing the intersection operations of Equations (4.5) and (4.6), we may choose any domain D_j, $j \in C_i$, as the domain D_i^r (from Corollary 4.14), and we may choose any constraint C_{kl}, $k \in C_i$, $j \in C_l$, as the constraint C_{ij}^r (from Lemma 4.13). Hence, the reduced network of an arc- and path-consistent network can be constructed in $O(e)$ time.

Illustration The reduced-network representation of the network in Figure 4.5 is shown in Figure 4.12. As before, node i represents component C_i, where $C_1 = \{A, B\}$, $C_2 = \{C\}$, and $C_3 = \{D\}$. Note, for example, that the domain of C_1 is identical to the domains D_A and D_B in the original network. Similarly, the constraint between C_1 and C_3 is identical to the constraints C_{AD} and C_{BD} in the input network. It can be verified that the reduced network is arc and path consistent. □

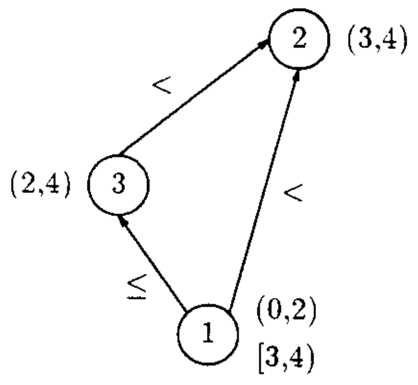

Figure 4.12: The reduced network of the network in Figure 4.5.

4.4.3 Augmented Convex Point Algebra Networks

This subsection is organized as follows. Section 4.4.3.1 presents a solution technique for CPA networks over discrete domains. Then, we discuss CPA networks over multiple-intervals domains: in Section 4.4.3.2 we present solution techniques for acyclic networks, and in Section 4.4.3.3 we extend those techniques to cyclic networks.

4.4.3.1 Discrete Domains

The consistency of a CPA network over discrete domains can be decided using arc consistency.

Theorem 4.16 *A nonempty arc-consistent CPA network over discrete domains is consistent; in particular, the tuple $H = (h_1, \ldots, h_n)$, where h_i is the highest value in domain D_i, is a solution.*

Proof See Appendix A. □

Theorem 4.16 provides an effective test for deciding the consistency of a given CPA network over discrete domains. We simply enforce arc consistency and then check whether the resulting domains are empty; the input network is consistent if and only if its arc-consistent form is nonempty. We shall say that arc consistency decides the consistency of a CPA network over discrete domains.

The fastest known arc-consistency algorithm for discrete domains is algorithm AC-4, which runs in $O(ek^2)$ time (Mohr and Henderson [49]). Recently,

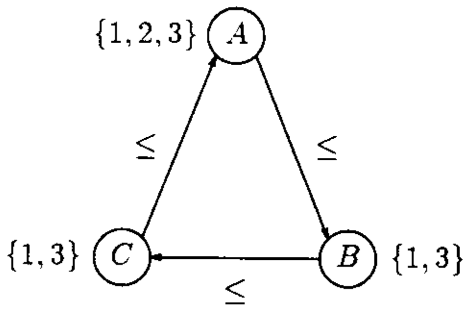

Figure 4.13: An arc-consistent CPA network over discrete domains.

Deville and Van Hentenryck [23] have devised a special-purpose arc-consistency algorithm. This algorithm runs in $O(ek)$ time for CPA networks over discrete domains. Hence, the complexity of deciding consistency and of finding a solution is bounded by $O(ek)$.

When computing the minimal domains, it turns out that arc consistency is insufficient.

Example 4.17 Consider the network in Figure 4.13. It has two solutions: $A = B = C = 1$ and $A = B = C = 3$. Clearly, the network is arc consistent; however, the value $A = 2$ is not part of any solution. Hence, the domain of A is not minimal. □

In Section 4.4.3.3 we shall show that the minimal domains can be computed by establishing both arc and path consistency.

4.4.3.2 Multiple-Intervals Domains—Acyclic Networks

An acyclic CPA network over multiple-intervals domains can be solved by establishing arc consistency and then instantiating the variables in a *backtrack-free* fashion [26] along any topological ordering of the precedence graph.

Lemma 4.18 *A nonempty arc-consistent acyclic CPA network over multiple-intervals domains is backtrack-free along any topological ordering of its precedence graph.*

Proof Let $G = (V, E)$ be an acyclic CPA network over multiple-intervals domains. Let $G_p = (V, E_p)$ be the precedence graph of G, and let d be a topological ordering of G_p. Suppose the first k variables along d, $X_1, \ldots, X_k$, were already instantiated to the values $v_1, \ldots, v_k$, respectively. We have to show that for any other variable X_i, $i > k$, there exists a value $v_i \in D_i$ such that all constraints C_{ji} $(1 \leq j \leq k)$ are satisfied.

If i is a source in G_p (i.e., it has no incoming arcs), then we may choose any value $v_i \in D_i$. Since all constraints C_{ji} are universal, they are trivially satisfied. If i is not a source in G_p, then let P be the parent set of i (namely, all nodes j such that $j \rightarrow i \in E_p$). Consider an arbitrary constraint C_{ji}, $j \in P$. Since G_p is acyclic, C_{ji} cannot be the equality constraint; furthermore, by the construction of G_p, it must be either $<$ or $\leq$. From arc consistency, we can select a value $l_j \in D_i$ that satisfies C_{ji}, namely, it is consistent with v_j. Let $v_i = \max\{l_j | j \in P\}$. Clearly, this value satisfies all the constraints C_{ji}, $j \in P$. Hence, G is backtrack-free along d. □

As an immediate corollary of Lemma 4.18, we have the following theorem, showing that arc consistency decides the consistency of an acyclic CPA network.

Theorem 4.19 *A nonempty arc-consistent acyclic CPA network over multiple-intervals domains is consistent.*

A solution to an arc-consistent acyclic CPA network G can be assembled in a backtrack-free fashion by algorithm Solve-Acyclic-CPA, shown in Figure 4.14. Based on the solution technique used in the proof of Lemma 4.18, algorithm Solve-Acyclic-CPA constructs a solution $V = (v_1, \ldots, v_n)$ to G by instantiating the nodes along a topological ordering d of the precedence graph $G_p = (V, E_p)$. Algorithm Solve-Acyclic-CPA is $O(e)$: a topological ordering d can be found in $O(e)$ time, each arc in G_p is considered only once (in Steps 4–6), and the time spent for each arc is constant.

Lemma 4.20 *The complexity of algorithm* AC-3 *for a PA network over multiple-intervals domains is* $O(en^2k^2)$.

Proof See Appendix A. □

Algorithm Solve-Acyclic-CPA

```
1. for $i := 1$ to $n$ do
2.     $v_i \leftarrow$ any value $v \in D_i$
3.     $L \leftarrow \emptyset$
4.     for each node $j$ such that $j \rightarrow i \in E_p$ do
5.         $L \leftarrow L \cup \{$a value in $D_i$ which is consistent with $v_j\}$
6.     $v_i \leftarrow \max(\{v_i\} \cup L)$
7. end
```

Figure 4.14: Solve-Acyclic-CPA—an algorithm for constructing a solution to an arc-consistent acyclic CPA network over multiple-intervals domains.

From Lemma 4.20, deciding consistency and finding a solution to a CPA network are both $O(en^2k^2)$. A more efficient approach would be to enforce *directional*, instead of full, arc consistency. Since in the proof of Lemma 4.18 we needed only directional arc consistency, Lemma 4.18 and consequently Theorem 4.19 can be modified as follows.

Lemma 4.21 *Let G be a nonempty acyclic CPA network over multiple-intervals domains. Let G_p be the precedence graph of G. Let d be a topological ordering of G_p, and let G be directional arc consistent along d. Then, G is backtrack-free along d.*

Theorem 4.22 *Let G be a nonempty acyclic CPA network over multiple-intervals domains. Let G_p be the precedence graph of G. If G is directional arc consistent along any topological ordering of G_p, then G is consistent.*

According to Theorem 4.22, directional arc consistency decides the consistency of an acyclic CPA network. A solution can still be constructed using algorithm Solve-Acyclic-CPA, because it employs only directional arc consistency.

Lemma 4.23 *The complexity of algorithm* DAC *for an acyclic CPA network over multiple-intervals domains is $O(e \log k)$.*

Proof See Appendix A. □

```
Algorithm 2DAC

1. $d \leftarrow$ a topological ordering of $G_p$
2. run DAC along $d$
3. $d_r \leftarrow$ the reverse of $d$
4. run DAC along $d_r$
```

Figure 4.15: 2DAC—an arc-consistency algorithm for acyclic CPA networks.

We conclude that the complexity of deciding consistency and of finding a solution to an acyclic CPA network is $O(e \log k)$, improving the upper bound of $O(en^2k^2)$ obtained by using full arc consistency.

Arc consistency can be also used in computing the minimal domains.

Theorem 4.24 *The domains of a nonempty arc-consistent acyclic CPA network over multiple-intervals domains are minimal.*

Proof See Appendix A. □

We have already seen (Lemma 4.20) that arc consistency can be achieved in $O(en^2k^2)$ time using algorithm AC-3. For an acyclic network, a tighter upper bound, $O(e \log k)$, can be achieved using algorithm 2DAC, shown in Figure 4.15. This algorithm performs two directional arc-consistency steps. The first moves backward, from sinks to the sources, and REVISEs arcs along a topological ordering of the precedence graph. The second moves forward, from sources to sinks, and REVISEs arcs along the reverse ordering. The first directional arc-consistency step changes only upper bounds of domains, while the second changes only lower bounds. Thus, upon termination of 2DAC all arcs are consistent, that is, the resulting network is arc consistent. The running time of algorithm 2DAC is $O(e \log k)$. We conclude that the minimal domains of an acyclic CPA network can be computed in $O(e \log k)$ time.

Illustration Consider the acyclic network of Figure 4.11. Running DAC along the ordering $d = (1, 3, 2)$ results in the directional arc-consistent network depicted in Figure 4.16. Then, running DAC along the reverse ordering $d_r = (2, 3, 1)$ yields the arc-consistent network of Figure 4.12. □

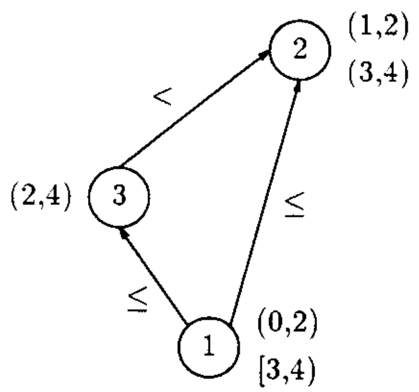

Figure 4.16: A directional arc-consistent form of the network in Figure 4.11.

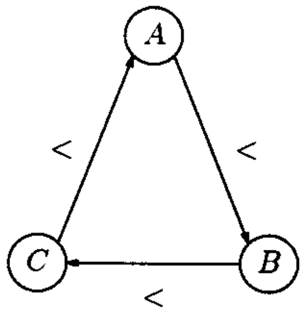

Figure 4.17: An arc-consistent CPA network.

4.4.3.3 Multiple-Intervals Domains—Cyclic Networks

Solving a *cyclic* CPA network requires more than just enforcing arc consistency. Arc consistency alone cannot even detect the inconsistency of a network.

Example 4.25 Consider the CPA network in Figure 4.17, where the domains are $(-\infty, \infty)$. The network is trivially arc consistent; however, it does not have any solution. □

One solution technique for cyclic networks is to establish both arc and path consistency.

Theorem 4.26 *A nonempty arc- and path-consistent CPA network over multiple-intervals domains is consistent.*

Proof Let G be a nonempty arc- and path-consistent CPA network over multiple-intervals domains. According to Lemma 4.15, the reduced network G^r is both nonempty and arc consistent. By Theorem 4.19, G^r is consistent. Hence, G is consistent. □

Theorem 4.26 provides an effective test for deciding consistency of an augmented CPA network. We establish both arc and path consistency, and then check whether the domains and constraints are empty. The network is consistent if and only if all domains and all constraints are nonempty. Similarly, arc and path consistency can be used in computing the minimal domains.

Theorem 4.27 *The domains of a nonempty arc- and path-consistent CPA network over multiple-intervals domains are minimal.*

Proof Let G be a nonempty arc- and path-consistent CPA network over multiple-intervals domains. According to Lemma 4.15, the reduced network G^r is nonempty and arc consistent. By Theorem 4.24, the domains of G^r are minimal. Since, as explained in Section 4.4.2, there exists a one-to-one correspondence between the solution sets of G and G^r, the domains of G are also minimal. □

A solution to a given arc- and path-consistent CPA network G can be found by first constructing its reduced network G^r, and then solving G^r using algorithm Solve-Acyclic-CPA.

The complexity of deciding consistency, finding a solution, and computing the minimal domains depends on the time needed to achieve arc and path consistency. Since path consistency is performed first, when arc consistency is executed the number of edges is $O(n^2)$. Hence, the complexity of the above reasoning tasks (using PC-2 and AC-3) is $O(n^4k^2)$.

An alternative, more efficient approach for solving a cyclic network is to convert it into a reduced-network representation, as explained in Section 4.4.2, and then solve the reduced network using techniques developed for acyclic networks. In particular, we can decide the consistency of the reduced network by using directional arc consistency, find a solution to the input network by applying algorithm Solve-Acyclic-CPA to the reduced network, and compute the minimal domains by enforcing full arc consistency on the reduced network. The complexity of all these tasks is dominated by the time needed to construct the reduced network, namely, $O(n^2k)$.

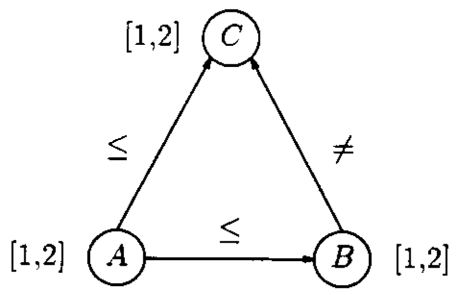

Figure 4.18: An arc-consistent PA network over single-interval domains.

4.4.4 Augmented Point Algebra Networks

When we move up the qualitative networks hierarchy from CPA networks to PA networks (allowing also the $\neq$ relation between points), deciding consistency becomes NP-hard for discrete domains, and consequently for multiple-intervals domains.

Proposition 4.28 *Deciding the consistency of a PA network over discrete domains is NP-hard.*

Proof Straightforward reduction from graph coloring. □

We shall now show that when the domains range over single intervals, deciding consistency and computing the minimal domains remain tractable. Actually, in the subsequent presentation we shall consider the more general case of *almost-single-interval domains*. Each domain D_i will consists of a single interval, from which a finite set of *holes* $H_i = \{h_{i_1}, \ldots, h_{i_k}\}$ is excluded.

As for CPA networks, we start by concentrating on acyclic networks and showing that arc consistency can be used in their solution. Recall that an arc-consistent acyclic CPA network is backtrack-free along any topological ordering of its precedence graph. Unfortunately, this property does not hold in PA networks.

Example 4.29 Consider the arc-consistent network in Figure 4.18. The precedence graph of this network consists of two arcs: $A \rightarrow B$ and $A \rightarrow C$. The ordering $d = (A, B, C)$ is a topological ordering of the precedence graph; however, the instantiation $A = B = 2$ cannot be extended to C. □

One way to alleviate this problem is to consider a *restricted network*, obtained from the input network by excluding the extreme points from all infinite domains.

Definition 4.30 Let G be a PA network. The *restricted network* of G, G', is obtained from G by restricting the domains as follows. If a domain D_i contains more than one value, then the domain of variable X_i in G' is

$$D'_i = D_i - \{\inf(D_i), \sup(D_i)\}.$$

An important property of the restricted network is that it remains arc consistent whenever the input network is arc consistent.

Lemma 4.31 *The restricted network of an arc-consistent PA network over almost-single-interval domains is arc consistent.*

Proof See Appendix A. □

Although, as shown in Example 4.29, an arc-consistent network is not necessarily backtrack-free, the restricted network *can* be solved in a backtrack-free fashion, along *any* topological ordering of its precedence graph.

Lemma 4.32 *Let G be a nonempty arc-consistent acyclic PA network over almost-single-interval domains. Let G' be the restricted network of G. Then, G' is backtrack-free along any topological ordering of its precedence graph.*

Proof Let $G_p = (V, E_p)$ be the precedence graph of G', and let d be a topological ordering of G_p. From Lemma 4.31, since G is arc consistent, G' is also arc consistent. Suppose the first k variables along d, $X_1, \ldots, X_k$, were already instantiated to the values $v_1, \ldots, v_k$, respectively. We have to show that for any other variable X_i, $i > k$, there exists a value $v_i \in D'_i$ such that all constraints C_{ji} $(1 \leq j \leq k)$ are satisfied.

If i is a source in G_p (namely, it has no incoming arcs), then we may choose any value $v_i \in D'_i$. Since all constraints C_{ji}, $j < i$, are universal, they are trivially satisfied.

If i is not a source in G_p, then we must select a value $v_i \in D'_i$ such that all constraints C_{ji}, $1 \leq j \leq k$, are satisfied. If D'_i consists

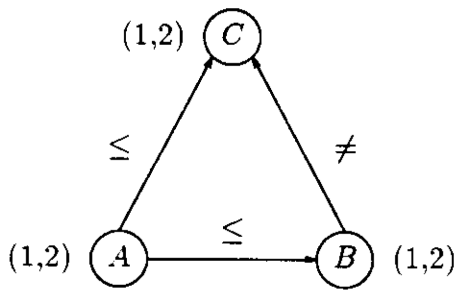

Figure 4.19: The restricted network of the network in Figure 4.18.

of a single value v then, from arc consistency, all these constraints are satisfied. If D'_i contains more than one value, then a value $v_i \in D'_i$ that satisfies all constraints C_{ji}, $1 \leq j \leq k$, can be found as follows. Let P be the parent set of i in G_p (namely all nodes j such that $j \rightarrow i \in E_p$). Consider an arbitrary constraint C_{ji}, $j \in P$. Since G_p is acyclic, C_{ji} cannot be the equality constraint; furthermore, by the construction of G_p, it must be either $<$ or $\leq$. From arc consistency of G', we can select a value $l_j \in D'_i$ that is compatible with v_j. Moreover, l_j can always be selected such that $\max(H_i) < l_j < \sup(D'_i)$. Let $m = \max(\{l_j | j \in P\})$. Let $N = \{v_j | j < i, C_{ji} \text{ is } \neq\}$. Since N is finite, we can always find a value v_i such that $v_i \in [m, \sup(D'_i))$, but $v_i \notin N$. Clearly, $v_i \in D'_i$, and it satisfies all the constraints C_{ji}, $1 \leq j \leq k$. Hence, G' is backtrack-free along d. □

Illustration Consider the network in Figure 4.18. Its restricted network is depicted in Figure 4.19. It can be easily verified that the restricted network is backtrack-free along the orderings $d_1 = (A, B, C)$ and $d_2 = (A, C, B)$. □

As a corollary to Lemma 4.32, we have the following theorem.

Theorem 4.33 *A nonempty arc-consistent acyclic PA network over almost-single-interval domains is consistent.*

In order to make use of Theorem 4.33 and employ an arc consistency algorithm in a procedure for deciding consistency, we still have to show that when the input domains range over almost-single intervals, they remain so after enforcing arc consistency. The next lemma shows that when we use an arc consistency algorithm based on REVISE operations, the domains of the resulting arc-consistent network also consist of almost-single intervals.

Lemma 4.34 *Let G be a PA network over almost-single-interval domains. Let G' be a network produced by applying* REVISE *to G. Then, G' is also a PA network over almost-single-interval domains.*

Proof See Appendix A. □

According to Theorem 4.33 and Lemma 4.34, AC-3 (or any other REVISE-based arc-consistency algorithm) determines the consistency of an acyclic PA network over almost-single-interval domains.

A solution to an arc-consistent acyclic PA network G can be assembled in a backtrack-free fashion by algorithm Solve-Acyclic-PA, shown in Figure 4.20. Based on the solution technique used in the proof of Lemma 4.32, algorithm Solve-Acyclic-PA constructs a solution $V = (v_1, \ldots, v_n)$ to the restricted network G' by instantiating the nodes along a topological ordering d of the precedence graph $G_p = (V, E_p)$. Algorithm Solve-Acyclic-PA is $O(e)$: a topological ordering can be found in $O(e)$ time, each arc in E is considered at most once (in Steps 7–9 or in Steps 11–13), and for each arc the algorithm spends a constant time.

From Lemma 4.20, the complexity of deciding consistency and of finding a solution to an acyclic PA network is $O(en^2k^2)$ for almost-single-interval domains and $O(en^2)$ for single-interval domains.

For the special case of acyclic networks, arc consistency can be achieved even more efficiently by algorithm 4DAC, shown in Figure 4.21. Given an acyclic PA network G, 4DAC enforces directional arc consistency four times: twice along a topological ordering d of the precedence graph G_p, and twice along the reverse ordering d_r.

Lemma 4.35 *Algorithm* 4DAC *computes an arc-consistent network.*

Proof See Appendix A. □

Lemma 4.35 guarantees that four applications of DAC are sufficient to compute an arc-consistent network. Example 4.36 shows that we cannot do better than that—four applications are indeed necessary.

Algorithm Solve-Acyclic-PA

```
1.  for $i := 1$ to $n$ do
2.      if $D'_i$ consists of a single value $v$ then
3.          $v_i \leftarrow v$
4.      else begin
5.          $v_i \leftarrow$ a value in $D'_i$
6.          $L \leftarrow \emptyset$
7.          for each $j$ such that $j \rightarrow i \in E_p$ do
8.              $L \leftarrow L \cup \{$a value in $D'_i$ that is consistent with $v_j\}$
9.          $v_i \leftarrow \max(\{v_i\} \cup L)$
10.         $N \leftarrow \emptyset$
11.         for each $j < i$ such that $C_{ij}$ is $\neq$ do
12.             $N \leftarrow N \cup \{v_j\}$
13.         $v_i \leftarrow$ a value in $[v_i, \sup(D'_i)) - N$
14.     end
15. end
```

Figure 4.20: Solve-Acyclic-PA—an algorithm for constructing a solution to an arc-consistent acyclic PA network over almost-single-interval domains.

Algorithm 4DAC

```
1. $d \leftarrow$ a topological ordering of $G_p$
2. $d_r \leftarrow$ the reverse of $d$
3. run DAC along $d$
4. run DAC along $d_r$
5. run DAC along $d$
6. run DAC along $d_r$
```

Figure 4.21: 4DAC—an arc consistency algorithm for acyclic PA networks over almost-single-interval domains.

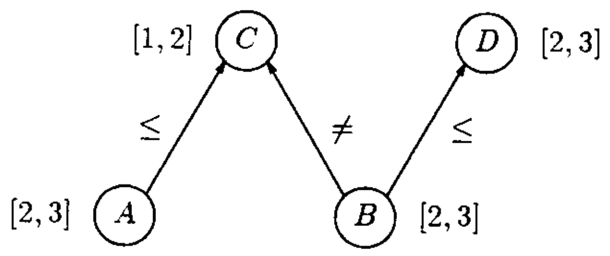

Figure 4.22: A PA network over single-interval domains.

Example 4.36 Consider the network in Figure 4.22. Let us execute algorithm 4DAC along the ordering $d = (A, B, C, D)$. During the first DAC the domain D_A is reduced to a single value [2]. Consequently, during the second DAC the domain D_C is also reduced to [2]. Then, during the third DAC the lower bound of D_B is changed and D_B becomes $(2, 3]$. Finally, in the fourth application of DAC D_D is changed to $(2, 3]$. The resulting network is indeed arc consistent. □

The running time of algorithm 4DAC is proportional to that of algorithm DAC for acyclic PA networks.

Lemma 4.37 *The complexity of algorithm* DAC *for an acyclic PA network over multiple-intervals domains is* $O(e(k + n))$.

Proof See Appendix A. □

We conclude that the complexity of algorithm 4DAC, and consequently the complexity of deciding consistency and of finding a solution, is $O(e(k + n))$ for almost-single-interval domains and $O(en)$ for single-interval domains.

It should be noted that, unlike CPA networks, PA networks cannot be solved using *directional* arc consistency. There are two possible ways to decide consistency in PA networks using directional arc consistency: applying DAC to the restricted network, or executing DAC on the input network and then restricting the domains. It can be easily verified that both methods fail to serve as a test for deciding consistency.

Arc consistency can also be used in computing the minimal domains of acyclic PA networks. The next theorem shows that arc consistency computes the minimal domains of the restricted network.

Theorem 4.38 *Let G be a nonempty arc-consistent acyclic PA network over almost-single-interval domains. Let G' be the restricted network of G. Then, all domains in G' are minimal.*

Arc consistency does not compute the minimal domains of the *input* network, however. For example, in the arc-consistent network of Figure 4.18, the value $A = 2$ does not participate in any solution and, thus, the domain D_A is not minimal. Nevertheless, arc consistency can still be used in computing the minimal domains. Consider an arc-consistent network G. According to Theorem 4.38, the domains of its restricted network G' are minimal. Thus, all single-value domains are in their minimal form and, for each infinite domain D_i, all values in the open interval $(\inf(D_i), \sup(D_i))$ are in the minimal domain. It remains, for each infinite domain, to check whether, in the case that $\inf(D_i) \in D_i$ or $\sup(D_i) \in D_i$, these values are also part of the minimal domain. This can be tested by Theorem 4.38. We set $D_i \leftarrow \inf(D_i)$ and then test the consistency of this network (by running arc consistency). If this network is consistent then $\inf(D_i)$ is in the minimal domain. The same test is performed for $\sup(D_i)$. The complexity of computing the minimal domains using this method is $O(n)$ times the complexity of determining consistency, namely, $O(en(k + n))$ for almost-single-interval domains and $O(en^2)$ for single-interval domains.

Illustration Consider the network in Figure 4.18. Let us compute the minimal domain of variable A. Every value in the open interval $(1, 2)$ is guaranteed to be in the minimal domain. We need to check whether $A = 1$ and $A = 2$ are in the minimal domain. Setting $D_A \leftarrow 1$ and running arc consistency yields a nonempty network; hence, $A = 1$ is contained in the minimal domain. Setting $D_A \leftarrow 2$ and running arc consistency yields an empty network; hence, $A = 2$ is not part of the minimal domain. We conclude that the minimal domain of variable A is $[1, 2)$. □

Solving *cyclic* PA networks over almost-single-interval domains can be done in two ways: by using arc and path consistency or by applying solution techniques for acyclic networks to the reduced network representation. Let us first consider the use of arc and path consistency.

Theorem 4.39 *A nonempty arc- and path-consistent PA network over almost-single-interval domains is consistent.*

Proof Let G be a nonempty arc- and path-consistent PA network over almost-single-interval domains. According to Lemma 4.15, the reduced network G^r is nonempty and arc consistent. By Theorem 4.33, G^r is consistent. Hence, G is consistent. □

Theorem 4.39 shows that, as for CPA networks, arc and path consistency decide consistency in PA networks. A solution to an arc- and path-consistent PA network G over almost-single-interval domains can be found by first constructing its reduced network G^r and then solving G^r using algorithm Solve-Acyclic-PA.

The complexity of deciding consistency and of finding a solution to a PA network using arc and path consistency is dominated by the time needed to establish arc consistency. The complexity of these reasoning tasks (using PC-2 and AC-3) is $O(n^4k^2)$ for almost-single-interval domains and $O(n^4)$ for single-interval domains.

Arc and path consistency can be also used in computing the minimal domains.

Theorem 4.40 *Let G be a nonempty arc- and path-consistent PA network over almost-single-interval domains. Let G' be the reduced network of G. Then, the domains of G' are minimal.*

Proof According to Lemma 4.31 G' is arc consistent and, from Lemma 4.15, its reduced network $(G')^r$ is nonempty and arc consistent. It can also be easily verified that $(G')^r$ is already in its restricted form. Hence, by Theorem 4.38, the domains of $(G')^r$ are minimal. Since, as explained in Section 4.4.2, there exists a one-to-one correspondence between the solution sets of G' and $(G')^r$, the domains of G' are also minimal. □

As in the case of an acyclic network, in order to compute the minimal domains of the input, cyclic network, we still have to test whether for each domain D_i the extreme points are in the minimal domain. This can be done by the same method described for acyclic networks, that is, by setting $D_i \leftarrow \inf(D_i)$ and $D_i \leftarrow \sup(D_i)$ and then testing consistency (using only arc consistency, since the network is already path consistent). The complexity of this method is $O(n)$ times the complexity of arc consistency, namely, $O(n^5k^2)$ for almost-single-interval domains and $O(n^5)$ for single-interval domains.

PA networks can be solved even more efficiently by applying the best algorithms for acyclic networks to the reduced network representation. Recall that constructing the reduced network representation requires $O(n^2k)$ time. Therefore, deciding consistency and finding a solution can be done in time $O(n^2k + e(k + n)) = O(n^2k + en)$ for almost-single-interval domains and time $O(en)$ for single-interval domains, and computing the minimal domains can be done in time $O(n^2k+en(k+n)) = O(en(k+n))$ for almost-single-interval domains and time $O(en^2)$ for single-interval domains.

4.5 Solving General Networks

In this section we focus on solving general networks. The input network may now contain all the types of constraints allowed in our language. We first describe an exponential, brute-force algorithm. Then, we investigate the applicability of path consistency algorithms.

We return to the network representation described in Section 4.2. Namely, in contrast with Section 4.4, we now use a binary-constraint representation for unary constraints, which means that the network now consists solely of binary constraints.

Let G be a given general network. A ***basic label*** of an arc $i \rightarrow j$ is a selection of a single interval from the interval set (if C_{ij} is quantitative) or a basic relation from the QA element (if C_{ij} is qualitative). A network whose arcs are labeled by basic labels of G is called a ***singleton labeling*** of G. We may solve G by generating all its singleton labelings, solving each of them independently, and then combining the results. Specifically, G is consistent if and only if there exists a consistent singleton labeling of G; the minimal network can be computed by taking the union over the minimal networks of all the singleton labelings.

Each qualitative constraint in a singleton labeling can be translated into a set of up to four linear inequalities on points. These inequalities, in turn, can be translated into metric constraints using the QUAN translation. It follows that a singleton labeling is equivalent to an STP network—a metric network whose constraints are labeled by single intervals. Recall (from Section 3.2) that an STP network can be solved in $O(n^3)$ time. Thus, the overall complexity of this decomposition scheme is $O(n^3k^e)$, where n is the number of variables, e is the number of arcs in the constraint graph, and k is the maximum number of basic labels per arc.

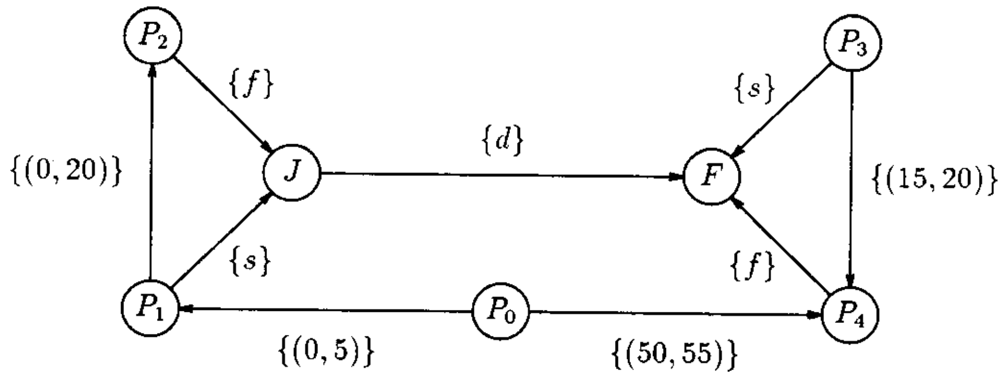

Figure 4.23: A singleton labeling of the constraint graph of Figure 4.2.

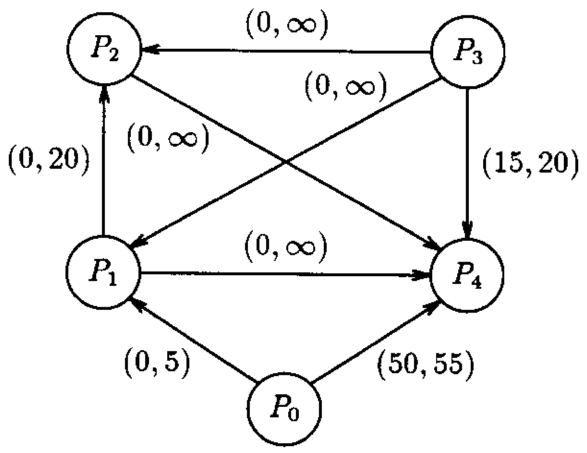

Figure 4.24: The STP network of the singleton labeling of Figure 4.23.

Illustration Consider the constraint graph of Figure 4.2. One singleton labeling is shown in Figure 4.23. The qualitative constraint J $\{during\}$ F can be translated into four linear inequalities on the endpoints of J and F: $P_1 > P_3$, $P_1 < P_4$, $P_2 > P_3$, and $P_2 < P_4$. Using the QUAN translation, these inequalities are translated into the following metric constraints: $P_1 - P_3 \in \{(0,\infty)\}$, $P_4 - P_1 \in \{(0,\infty)\}$, $P_2 - P_3 \in \{(0,\infty)\}$, and $P_4 - P_2 \in \{(0,\infty)\}$. The resulting STP network is shown in Figure 4.24. □

The brute-force enumeration of singleton labelings can be pruned significantly by running a backtracking algorithm on a meta-CSP in which the variables are the network arcs and the domains are the possible basic labels. This algorithm is similar to the backtracking algorithm for metric networks, described in Section 3.3. It assigns a basic label to an arc, as long as the corresponding STP network is consistent; if no such assignment is possible, it backtracks.

Imposing local consistency among subsets of variables may serve as a preprocessing step to improve backtrack. This strategy has been proven successful (see [20]), since enforcing local consistency can be achieved in polynomial time, while it may substantially reduce the number of dead-ends encountered in the search phase itself. In particular, experimental evaluation shows that enforcing a low consistency level, such as arc or path consistency, gives the best results [20]. Following this rationale, we next show that path consistency, which in general networks amounts to the least amount of preprocessing,[9] can be achieved in polynomial time.

Theorem 4.41 *Algorithm* PC-2 *calls* REVISE $O(n^3R)$ *times, and its timing is bounded by* $O(n^3R^3)$*, where* R *is the range*[10] *of* G*.*

> **Proof** Let G be a given network. Without loss of generality, we may assume that G is integral; otherwise, we can simulate the algorithm on the equivalent integral network. The number of calls to REVISE is proportional to the total number of triplets on Q throughout the execution of PC-2. The initial size of Q is $O(n^3)$. The worst case running time of PC-2 occurs when each metric constraint is decreased by only one unit and each qualitative constraint is decreased by only one basic relation each time a constraint is tightened by REVISE. In this case, if R is the range of G, then each metric constraint might be updated $O(R)$ times and each qualitative constraint may be updated no more than 13 times. Also, in the worst case, when a constraint is modified, $O(n)$ triplets are added to Q [43]. Thus, each constraint may cause the addition of $O(nR)$ triplets to Q. Hence, since there are $O(n^2)$ constraints, the total number of new entries on Q is $O(n^3R)$, namely, PC-2 performs $O(n^3R)$ calls to REVISE. A call to REVISE involves intersection and composition. The worst case occurs when all operands are metric constraints. In this case, the cost of REVISE is $O(R^2)$. Hence, the total timing of PC-2 is $O(n^3R^3)$. □

[9]General networks are trivially arc consistent since unary constraints are represented as binary constraints.

[10]See Section 3.4 for the definition of the range of a metric network. The range of a general network is the range of its quantitative subnetwork.

Path consistency can also be regarded as an alternative approach to exhaustive enumeration, serving as an approximation scheme that often yields the minimal network. For example, applying path consistency to the network of Figure 4.2 produces the minimal network. Although, in general, a path-consistent network is not necessarily minimal and may not even be consistent, in some cases path consistency is guaranteed to determine the consistency of a network.

Proposition 4.42 *Let G be a path-consistent network. If the qualitative subnetwork of G is in $net(CPA)$ and the quantitative subnetwork constitutes an STP network, then G is consistent and its metric constraints are minimal.*

> **Proof** Let G_M be the metric subnetwork of G. Consider a metric constraint C_{ij}. Let x and y be values, assigned to variables X_i and X_j, respectively, that satisfy C_{ij}. In Section 3.2 we show that since G_M is path consistent, this partial assignment can be extended to a full solution of G_M. Since the qualitative subnetwork is in $net(CPA)$, this assignment satisfies all qualitative constraints, and hence it is a solution to G. We conclude that C_{ij} is minimal and that G is consistent. □

Note that the condition in Proposition 4.42 cannot be weakened to include networks whose qualitative part is in $net(PA) - net(CPA)$. The reason is that the networks satisfying the condition of Proposition 4.42 are closed under REVISE, namely, applying REVISE to any network in this class produces a network that still belongs to the same class. This is not true when the qualitative subnetwork is in $net(PA) - net(CPA)$. In this case, REVISE may introduce holes in metric constraints, yielding a non-STP metric subnetwork.

Unfortunately, even for networks satisfying the condition of Proposition 4.42, path consistency is not guaranteed to compute the minimal network. According to Proposition 4.42, path consistency computes the minimal constraints for the *metric* part of the network. Yet, it may not reduce some qualitative constraints to their minimal form.

Example 4.43 Consider the network in Figure 4.25. It consists of two intervals, $I = [A, B]$ and $J = [C, D]$, and two metric constraints on their length,

$$B - A \in \{(1, 2)\}$$

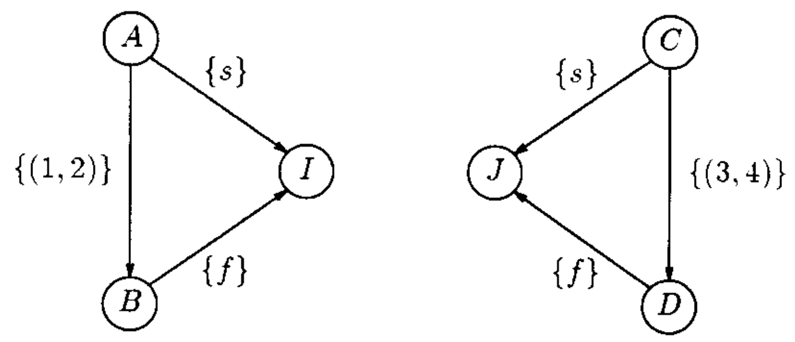

Figure 4.25: A path-consistent singleton labeling.

and

$$D - C \in \{(3,4)\}.$$

Note that the constraint between I and J is the universal constraint, permitting all 13 basic relations. This network is path consistent; however, it can be easily verified that the basic relation $=$ is not in the minimal constraint between I and J. □

One way to compute the minimal qualitative constraints is the following. Let C_{ij} be a qualitative constraint labeled by a relation set R. For each basic relation $r \in R$ we set $C_{ij} \leftarrow r$ and then test the consistency of the resulting network. Because the new network still satisfies the condition of Proposition 4.42, path consistency can be used to decide its consistency. If the new network is consistent, then r is in the minimal constraint between i and j. Since there are $O(n^2)$ qualitative constraints, each one consisting of no more than 13 basic relations, the entire minimal network can be computed using $O(n^2)$ applications of path consistency.

For some networks, path consistency is even guaranteed to compute the entire minimal network.

Proposition 4.44 *Any path-consistent singleton labeling is minimal.*

> **Proof** We need to show that the qualitative constraints are minimal. According to Proposition 4.42 the network is consistent. Thus, since each qualitative constraint consists of a single basic relation, it must be in its minimal form. □

We feel that more classes of temporal problems may be solved by path consistency algorithms. Further investigation may reveal new classes that can be solved using these algorithms.

4.6 Relations to Other Formalisms

Kautz and Ladkin [35] have introduced an alternative model for temporal reasoning. It consists of two components: a metric network and an IA network. These two networks, however, are not connected via internal constraints; rather, they are kept separately, and the inter-component relationships are managed by means of external control. To solve reasoning tasks in this model, Kautz and Ladkin proposed an algorithm that solves each component independently and then circulates information between the two parts, using the QUAL and QUAN translations, until a fixed point is reached. Our model has two advantages over Kautz and Ladkin's model:

1. It is conceptually clearer, since all information is stored in a single network and constraint propagation takes place in the knowledge level itself.
2. It is computationally more efficient, since we are able to provide tighter bounds for various reasoning tasks. For example, in order to convert a given network into an equivalent path-consistent form, Kautz and Ladkin's algorithm may require $O(n^2)$ information transferences, resulting in an overall complexity of $O(n^5R^3)$, compared to $O(n^3R^3)$ in our model.

CHAPTER 5

Conclusions

Existing constraint-based approaches for temporal reasoning—Allen's interval algebra and Vilain and Kautz's point algebra—facilitate reasoning about qualitative relations between either points or intervals, but they cannot handle many forms of quantitative knowledge. This thesis offers two alternative formalisms.

The first formalism, called temporal constraint satisfaction problem (TCSP), provides a framework for dealing with quantitative information, such as duration and timing of events. In this framework, variables represent time points, and temporal information is represented by a set of unary and binary constraints, each specifying a set of permitted intervals. We present algorithms for performing the following reasoning tasks: finding all feasible times that a given event can occur, finding all possible relationships between two given events, and generating one or more scenarios consistent with the information provided.

We distinguish between simple temporal problems (STPs) and general temporal problems, the former admitting at most one interval constraint on any pair of time points. We show that the STP can be solved in polynomial time, $O(n^3)$, using the well-known Floyd-Warshall's all-pairs-shortest-paths algorithm. For general TCSPs, we present a decomposition scheme that provides answers to the reasoning tasks considered, but its computational efficiency, in the worst case, might be limited. The decomposition scheme can be improved by traditional constraint satisfaction techniques, such as backjumping, learning, various ordering schemes, and preprocessing techniques. We study the applicability of path consistency algorithms as preprocessing of temporal problems, demonstrate their termination, and bound their complexities; path consistency algorithms seem to offer a practical compromise in very complex problems. In particular, the more efficient directional path consistency was shown to retain the essential properties of full path consistency in determining consistency of STPs and in enhancing backtrack search of general TCSPs. Among the specialized network-based algorithms, only decomposition into nonseparable components was found applicable to TCSPs. It offers a method for computing the minimal network in time exponential in the largest nonseparable component.

We see the main application of the TCSP framework to be in temporal reasoning tasks involving metric information, namely, expressions involving absolute time differences (e.g., "John came home an hour after Mary"). In this respect, the expressiveness of the TCSP language supersedes that of Allen's interval algebra. However, it is weaker than the interval algebra, being limited to problems involving constraints on pairs of time points. The TCSP framework includes Vilain and Kautz's point algebra as a special case and provides a variety of techniques and intuitions for solving problems in this domain.

The second formalism presented in this thesis, general temporal networks, is a general constraint-based model for temporal reasoning that is capable of handling both qualitative and quantitative information. It facilitates the processing of quantitative constraints on points and all qualitative constraints between temporal objects. The unique feature of this framework is that it allows the representation and processing of all types of qualitative constraints considered in the literature to date, as well as the quantitative constraints of the TCSP model. It can be seen as a generalization of Allen's interval algebra, Vilain and Kautz's point algebra, and the TCSP model.

We utilize constraints satisfaction techniques in solving reasoning tasks in this model. In particular, general networks can be solved by a backtracking algorithm, or by path consistency, which computes an approximation to the minimal network.

Using our integrated model we were able to identify new classes of tractable networks—those networks that can be solved by path consistency algorithms, for example, singleton labelings.

Other tractable classes were obtained by augmenting tractable qualitative networks, PA and CPA networks, with various domain constraints. We show that some of these networks can be solved using arc and path consistency. Tables 5.1 and 5.2 summarize the complexity of determining consistency and computing the minimal domains in augmented qualitative networks. Each entry gives the consistency level which can be used to solve the corresponding task (AC, PC, or both), and the timing of the best algorithm discussed in this thesis.

Future research should enrich the representation language to facilitate model-

	Discrete	Single interval	Multiple intervals
CPA networks	AC $O(ek)$	AC + PC $O(n^2)$	AC + PC $O(n^2k)$
PA networks	NP-complete	AC + PC $O(en)$	NP-complete
IPA networks	NP-complete	NP-complete	NP-complete

Table 5.1: Complexity of deciding consistency in augmented qualitative networks.

	Discrete	Single interval	Multiple intervals
CPA networks	AC + PC $O(n^2k)$	AC + PC $O(n^2)$	AC + PC $O(n^2k)$
PA networks		AC + PC $O(en^2)$	

Table 5.2: Complexity of computing the minimal domains in tractable augmented qualitative networks.

ing of more involved reasoning tasks. In particular, the following constraint types should be incorporated in a temporal reasoning system.

1. *Nonbinary constraints.* Consider, for example, the statement "John's drive to work is at least 30 minutes longer than Fred's." Let $J = [A, B]$ and $F = [C, D]$ be intervals representing the events "John was going to work" and "Fred was going to work," respectively. Then, the above statement is expressed by the 4-ary constraint:

$$B - A \geq D - C + 30.$$

2. *Logical constraints.* For example, statements such as "If John leaves home before 7:15 a.m., he arrives at work before Fred." These are essentially nonbinary constraints, but they may have a special structure (e.g., causal (horn-type) constraints or a disjunction of binary constraints).

3. *Recurring events.* In the formalisms discussed in this thesis, each event is associated with a single, convex interval. Some events, however, need to

be represented by a set of intervals [37]. For example, when representing knowledge about last week, the event "John was going to work" is associated with a set of five intervals corresponding to the subevents: "John was going to work on Monday," ... "John was going to work on Friday."

Although some work has been done on these issues (e.g., [37]), most of the interesting problems have yet to be solved.

APPENDIX A

Proofs

Proof of Lemma 4.10 The *if* part is trivial—if C_{ij} is $=$ then, by definition, both $i \rightarrow j \in E_p$ and $j \rightarrow i \in E_p$, and thus i and j belong to the same strongly connected component.

We now show the *only if* part. Suppose i and j belong to the same strongly connected component in G_p. Then, there exists a directed path $i_1 = i \rightarrow i_2 \rightarrow \cdots \rightarrow i_k = j$ from i to j in G_p. By the construction of G_p, all the corresponding constraints in G are either $<$, $\leq$, or $=$. It can be verified easily that the composition of these constraints cannot contain $>$. Thus

$$C_{i_1,i_2} \otimes \cdots \otimes C_{i_{k-1},i_k} \subseteq \{<,=\}$$

and, from path consistency,

$$C_{ij} \subseteq C_{i_1,i_2} \otimes \cdots \otimes C_{i_{k-1},i_k} \subseteq \{<,=\}. \tag{A.1}$$

Similarly, there exists a directed path $j_1 = j \rightarrow j_2 \rightarrow \cdots \rightarrow j_k = i$ from j to i in G_p. The corresponding constraints in G, in the direction from i to j, are either $>$, $\geq$, or $=$. Thus

$$C_{j_k,j_{k-1}} \otimes \cdots \otimes C_{j_2,j_1} \subseteq \{>,=\}$$

and, from path consistency,

$$C_{ij} \subseteq C_{j_k,j_{k-1}} \otimes \cdots \otimes C_{j_2,j_1} \subseteq \{>,=\}. \tag{A.2}$$

From Equations (A.1) and (A.2), $C_{ij} \subseteq \{=\}$, and since all constraints are nonempty, C_{ij} must be $=$. □

Proof of Lemma 4.13 There are three cases:

1. Case 1: $i = k, j \neq l$. From path consistency, $C_{il} \subseteq C_{ij} \otimes C_{jl}$. According to Lemma 4.10, C_{jl} is $=$, thus
$$C_{il} \subseteq C_{ij}. \tag{A.3}$$

Similarly, from path consistency, $C_{ij} \subseteq C_{il} \otimes C_{lj}$. According to Lemma 4.10, C_{lj} is $=$, thus

$$C_{ij} \subseteq C_{il}. \tag{A.4}$$

From Equations (A.3) and (A.4), $C_{ij} = C_{il} = C_{kl}$.

2. Case 2: $j = l, i \neq k$. From Case 1, $C_{ji} = C_{jk} = C_{lk}$, and thus $C_{ij} = C_{kl}$.

3. Case 3: $i \neq k, j \neq l$. From previous cases we have $C_{ij} = C_{il} = C_{kl}$.

Hence, for all cases $C_{ij} = C_{kl}$. □

Proof of Theorem 4.16 Let G be a nonempty arc-consistent CPA network over discrete domains. We shall show that the tuple $H = (h_1, \ldots, h_n)$ is a solution. Consider an arbitrary constraint C_{ij}, and the values h_i and h_j assigned to variables X_i and X_j, respectively. There are three cases depending on C_{ij}.

1. C_{ij} is $=$. Then, h_i must be equal to h_j. Otherwise, suppose $h_i \neq h_j$. Without loss of generality, we may assume that $h_i < h_j$. From arc consistency, there exists a value $h_j \in D_i$. This contradicts the fact that h_i is the highest value in D_i. Hence, $h_i = h_j$.

2. C_{ij} is $<$ or $>$. Without loss of generality, we may assume that C_{ij} is $<$ (otherwise we consider C_{ji}). Then, from arc consistency, there exists a value $v \in D_j$ such that $h_i < v$. By definition, $v \leq h_j$, and thus $h_i < h_j$.

3. C_{ij} is $\leq$ or $\geq$. Without loss of generality, we may assume that C_{ij} is $\leq$ (otherwise we consider C_{ji}). Then, from arc consistency, there exists a value $v \in D_j$ such that $h_i \leq v$. By definition, $v \leq h_j$, and thus $h_i \leq h_j$.

We conclude that the assignment $X_i = h_i, X_j = h_j$ satisfies the constraint C_{ij}. Since all the constraints are satisfied, H is a solution, and thus the network is consistent. □

The next lemmas are needed in analyzing the complexity of algorithm AC-3 in PA networks. As usual, let n, e, and k be the number of nodes, number of edges, and the maximum domain size, respectively.

Lemma A.1 *During the execution of* AC-3 *only input extreme points may occur in any domain.*

Proof All operations on domains (Equation (4.3)) involve quantitative composition of domain intervals with intervals from the set

$$\{(0,\infty),[0,\infty),[0],(-\infty,0],(-\infty,0)\},$$

and then intersection. It can be easily verified that these operations do not introduce new extreme points. □

Corollary A.2 *The number of intervals per domain is $O(nk)$.*

Lemma A.3 *The number of calls to* REVISE *is $O(enk)$.*

Proof We follow the analysis of Mackworth and Freuder [45]. The number of calls to REVISE is identical to the number of iterations of the while loop (Steps 2–6), that is, the total number of arcs on Q. Initially, there are $O(e)$ arcs on Q. We observe that when a domain changes, either some extreme points are added or deleted, or a closed interval becomes open. The worst case occurs when all the possible changes take place and none of the arcs to be added to Q is already on it. In this worst case, each call to REVISE either adds or deletes exactly one extreme point or opens one closed interval. From Lemma A.1, only input extreme points can occur in any domain; thus, a domain may change $O(nk)$ times.

Entries are made in Q only when a call to REVISE has changed a domain. If a domain D_i has been changed, then in the worst case $O(d_i)$ arcs are added to Q, where d_i is the degree of node i. Hence, the total number of new entries in Q is:

$$\sum_{i=1}^{n} O(d_i)O(nk) = O(enk).$$

Hence, the number of calls to REVISE is $O(enk)$. □

Proof of Lemma 4.20 The cost of REVISE is proportional to the number of intervals per domain—$O(nk)$ (Corollary A.2). The overall complexity of AC-3 is the number of calls to REVISE times the cost of REVISE, namely, $O(en^2k^2)$. □

Proof of Lemma 4.23 Since all constraints are from the set $\{<,\leq,>,\geq\}$, REVISE can be implemented, using binary search and then updating the pointers Inf and Sup, in $O(\log k)$ time. Since the number of calls to REVISE is proportional to the number of arcs, the total complexity is $O(e\log k)$. □

The next lemma is needed for the proof of **Theorem 4.24**.

Lemma A.4 *A nonempty arc-consistent acyclic CPA network $G = (V, E)$ over multiple-intervals domains is backtrack-free along any reverse topological ordering of its precedence graph.*

Proof Let $G = (V, E)$ be a nonempty arc-consistent acyclic CPA network over multiple-intervals domains. Let $G_p = (V, E_p)$ be the precedence graph of G, and let d be a reverse topological ordering of G_p. Suppose the first k variables along d, $X_1, \ldots, X_k$, were already instantiated to the values $v_1, \ldots, v_k$, respectively. We have to show that for any other variable X_i, $i > k$, there exists a value $v_i \in D_i$ such that all constraints C_{ji} $(1 \leq j \leq k)$ are satisfied.

If i is a sink in G_p (i.e., it has no outgoing arcs), then we may choose any value $v_i \in D_i$. Since all constraints C_{ji} are universal, they are trivially satisfied. If i is not a source in G_p, then let S be the successor set of i (namely, all nodes j such that $i \rightarrow j \in E_p$). Consider an arbitrary constraint C_{ji}, $j \in S$. Since G_p is acyclic, C_{ji} cannot be the equality constraint; furthermore, by the construction of G_p, it must be either $>$ or $\geq$. From arc consistency, we can select a value $l_j \in D_i$ that satisfies C_{ji}, namely, is consistent with v_j. Let $v_i = \min\{l_j | j \in S\}$. Clearly, this value satisfies all the constraints C_{ji}, $j \in S$. Hence, G is backtrack-free along d. $\square$

Proof of Theorem 4.24 Let $G = (V, E)$ be a nonempty arc-consistent acyclic CPA network over multiple-intervals domains. Let $G_p = (V, E_p)$ be the precedence graph of G. To show that a domain D_i is minimal, we need to show that every value $x \in D_i$ is part of a solution X of G.

Let x be an arbitrary value in D_i. Let V_1 be the set of all nodes $v \in G$ such that there exists a path from v to i in G_p. We construct a solution to G by instantiating first the nodes in V_1 and then the rest of the nodes. Consider $G_1 = (V_1, E_1)$, the subgraph induced by V_1 (containing only arcs connecting nodes in V_1). Let d_1 be a reverse topological ordering of G_1. According to Lemma A.4, G_1 is backtrack-free along d_1. Hence, we can construct a solution X' to G_1 by instantiating X_i to x and then instantiating the rest of the variables in V_1 in a backtrack-free fashion along d_1. Having instantiated the nodes in V_1, we can now extend X' to a full solution X of G as follows: Let d be a topological ordering of G whose restriction to G_1 is the reverse of d_1. The nodes in V_1 are already instantiated. According to Lemma 4.18, we can extend X' to a full solution of G by instantiating the rest of the nodes, $V - V_1$, backtrack-free along d. Therefore, there exists a solution to G in which $X_i = x$. □

Proof of Lemma 4.31 Let $G = (V, E)$ be an arc-consistent PA network over almost-single-interval domains, and let G' be its restricted network. Suppose G' is not arc consistent. Then there exists a pair of variables X_i and X_j, and a value $x \in D'_i$ such that x has no compatible value in D'_j. On the other hand, since G is arc consistent, x must have a compatible value in D_j. Thus, $D'_j \subset D_j$, that is, D_j contains more than one value, and x must be compatible with either $\inf(D_j)$ or $\sup(D_j)$. There are 4 cases depending on C_{ij}.

1. C_{ij} is either $<$ or $\leq$. If x was compatible with $\inf(D_j)$ (i.e., $x \leq \inf(D_j)$), then it would also be compatible with another value $y \in D'_j$, contradicting our assumption that x has no match in D'_j. Thus, x is incompatible with $\inf(D_j)$, and hence it must be compatible with $\sup(D_j)$, namely, $\inf(D_j) < x \leq \sup(D_j)$. We distinguish between two cases.

   (a) If $x < \sup(D_j)$ then let
   $$y = \frac{1}{2}[\max(\{x, \inf(D_j)\} \cup H_j) + \sup(D_j)].$$
   Clearly, $y \in D'_j$ and $x < y$. Hence, x has a match in D'_j; contradiction.

   (b) If $x = \sup(D_j)$ then, from arc consistency of G, C_{ij} must be $\leq$, and we must also have that $x = \sup(D_i)$. Thus, by definition of the restricted network, since $\sup(D_i) \in D'_i$, the domain D_i consists of a single value, that is, $D_i = D'_i = \{x\}$. Since C_{ij} is $\leq$, the constraint C_{ji} is $\geq$, and, by arc consistency of G, $D_j = \{x\}$. Thus, D_j consists of a single value; contradiction.

2. C_{ij} is either $>$ or $\geq$. This case is symmetric to the previous case. If x was compatible with $\sup(D_j)$ (i.e., $x \geq \sup(D_j)$), then it would also be compatible with another value $y \in D'_j$, contradicting our assumption that x has no match in D'_j. Thus, x is incompatible with $\sup(D_j)$, and hence it must be compatible with $\inf(D_j)$, namely, $\inf(D_j) \leq x < \sup(D_j)$. We distinguish between two cases.

   (a) If $x > \inf(D_j)$ then let
   $$y = \frac{1}{2}[\min(\{x, \sup(D_j)\} \cup H_j) + \inf(D_j)].$$
   Clearly, $y \in D'_j$ and $x > y$. Hence, x has a match in D'_j; contradiction.

(b) If $x = \inf(D_j)$ then, from arc consistency of G, C_{ij} must be $\geq$, and we must also have that $x = \inf(D_i)$. Thus, by definition of the restricted network, since $\inf(D_i) \in D'_i$, the domain D_i consists of a single value, that is, $D_i = D'_i = \{x\}$. Since C_{ij} is $\geq$, the constraint C_{ji} is $\leq$, and, by arc consistency of G, $D_j = \{x\}$. Thus, D_j consists of a single value; contradiction.

3. If C_{ij} is $=$ then, from arc consistency of G, $D_i = D_j$. Since x is compatible with either $\inf(D_j)$ or $\sup(D_j)$, we must also have $x = \inf(D_i)$ or $x = \sup(D_i)$. Thus, since either $\inf(D_i) \in D'_i$ or $\sup(D_i) \in D'_i$, by definition of the restricted network, D_i consists of a single value, namely, $D_i = \{x\}$. Hence, $D_j = \{x\}$, namely, it consists of a single value; contradiction.

4. If C_{ij} is $\neq$ then, since D'_j contains more than one value, there must be a value $y \in D'_j$ such that $x \neq y$, contradicting our assumption that x has no match in D'_j.

We conclude that x must have a compatible value in D'_j; hence, G' is arc consistent. □

Proof of Lemma 4.34 Consider the operation of REVISE (Equation (4.3)):

$$D_i \leftarrow D_i \oplus D_j \otimes \mathsf{QUAN}(C_{ji}).$$

There are three cases depending on C_{ij}.

1. If C_{ij} is a relation from the set $\{<, \leq, \geq, >\}$, then the composition of D_j with $\mathsf{QUAN}(C_{ji})$ yields a single, convex interval. The intersection of a convex interval with an almost-single interval gives an almost-single interval.

2. If C_{ij} is $=$, then the domain D_i is intersected with the domain D_j, yielding an almost-single-interval domain.

3. If C_{ij} is $\neq$, then there are two cases. If D_j contains more than one value, then D_i is not changed. If D_j consists of a single value v, then at most most one new hole, v, may be introduced.

We conclude that a call to REVISE produces a PA network over almost-single-interval domains. □

Proof of Lemma 4.35 Let G be a PA network over almost-single-interval domains. We first observe that the only case where a $\neq$ constraint C_{ij} may change a domain D_j occurs when the domain D_i consists of a single value v. In this case, either a new hole v is introduced in D_j or one of its extreme points, $\inf(D_j)$ or $\sup(D_j)$, is removed and thus a closed-interval domain is opened. All other constraints are CPA relations that change upper and lower bounds.

Consider the first two applications of DAC (Steps 3 and 4). If we disregard the $\neq$ constraints, then these two steps mimic algorithm 2DAC, in which the CPA constraints establish new lower and upper bounds on domains. However, the existence of the $\neq$ constraints may remove *finite* sets of values from some domains, introducing new holes or deleting extreme points. This forces more applications of DAC (Steps 5 and 6). It can be verified that, in these later applications, the CPA constraints may only fix some bounds by removing extreme points from domains, and the inequality constraints, as before, may remove only finite sets of values from domains. Thus, in Steps 5 and 6, only a finite set of values may be removed from each domain. As a result, all domains that will eventually consist of a single value v are reduced to this value during Steps 3 and 4.

Consider Steps 4 and 5. If a domain D_i was reduced to a single value (in Step 3 or Step 4), then during Step 4 all arcs $j \rightarrow i$, such that $i < j$ and the constraint C_{ij} is $\neq$, are made consistent. Then, in Step 5, for each domain D_i, which was reduced to a single value in previous steps, all arcs $i \rightarrow j$, such that $i < j$ and the constraint C_{ij} is $\neq$, are made consistent. Altogether, when Step 5 terminates, all arcs $i \rightarrow j$ such that C_{ij} is $\neq$ are made consistent and, since domains are monotonically reducing, they remain consistent when 4DAC terminates.

It remains to show that all arcs $i \rightarrow j$, such that C_{ij} is a CPA relation, are consistent when 4DAC terminates. However, this can be seen from the fact that, in Steps 5 and Step 6, the corresponding DACs only change upper and lower bounds, respectively. We therefore conclude that when 4DAC terminates all arcs are consistent, namely, the network is arc consistent. □

Proof of Lemma 4.37 The cost of REVISE is proportional to the number of intervals per domain. Initially, the domain size is $O(k)$. A domain D_i can change by an application of Equation (4.3). Note that since the network is acyclic, C_{ij} cannot be the equality constraint. When C_{ij} is a relation from the set $\{<, \leq, \geq, >\}$, the bound on the domain size is not changed. When C_{ij} is $\neq$, then D_i may be changed only when D_j contains exactly one value v. In this case, an interval in D_i may be split into two new intervals, thus increasing the size of D_i by 1. This situation can occur at most $O(n)$ times (once for every node). Hence, the number of intervals per domain, and consequently the cost of REVISE, is $O(k+n)$. Since the number of calls to REVISE is proportional to the number of arcs, the total complexity is $O(e(k+n))$. □

The next lemma is needed for the proof of **Theorem 4.38**.

Lemma A.5 *Let G be a nonempty arc-consistent acyclic PA network over almost-single-interval domains. Let G' be the restricted network of G. Then, G' is backtrack-free along any reverse topological ordering of its precedence graph.*

Proof Let $G_p = (V, E_p)$ be the precedence graph of G', and let d be a reverse topological ordering of G_p. From Lemma 4.31, since G is arc consistent, G' is also arc consistent. Suppose the first k variables along d, $X_1, \ldots, X_k$, were already instantiated to the values $v_1, \ldots, v_k$, respectively. We have to show that for any other variable X_i, $i > k$, there exists a value $v_i \in D'_i$ such that all constraints C_{ji} $(1 \leq j \leq k)$ are satisfied.

If i is a sink in G_p (i.e., it has no outgoing arcs), then we may choose any value $v_i \in D'_i$. Since all constraints C_{ji}, $j < i$, are universal, they are trivially satisfied.

If i is not a sink in G_p, then we must select a value $v_i \in D'_i$ such that all the constraints C_{ji}, $1 \leq j \leq k$, are satisfied. If D'_i consists of a single value v then, from arc consistency, all these constraints are satisfied. If D'_i contains more than one value, then a value $v_i \in D'_i$ that satisfies all constraints C_{ji}, $1 \leq j \leq k$, can be found as follows. Let S be the successor set of i in G_p (namely, all nodes j such that $i \to j \in E_p$). Consider an arbitrary constraint C_{ji}, $j \in S$. Since G_p is acyclic, C_{ji} cannot be the equality constraint; furthermore, by the construction of G_p, it must be either $>$ or $\geq$. From arc consistency of G', we can select a value $l_j \in D'_i$ that is compatible with v_j. Moreover, l_j can always be selected such that $\inf(D'_i) < l_j < \min(H_i)$. Let $m = \min(\{l_j | j \in S\})$. Let $N = \{v_j | j < i, C_{ji} \text{ is } \neq\}$. Since N is finite, we can always find a value v_i such that $v_i \in (\inf(D'_i), m]$, but $v_i \notin N$. Clearly, $v_i \in D'_i$, and it satisfies all the constraints C_{ji}, $1 \leq j \leq k$. Hence, G' is backtrack-free along d. □

Proof of Theorem 4.38 Let $G_p = (V, E_p)$ be the precedence graph of G'. To show that a domain D'_i is minimal, we need to show that every value $x \in D'_i$ is part of a solution X of G'.

Let x be an arbitrary value in D'_i. Let V_1 be the set of all nodes $v \in G'$ such that there exists a path from v to i in G_p. We construct a solution to G' by instantiating first the nodes in V_1 and then the rest of the nodes. Consider $G'_1 = (V_1, E_1)$, the subgraph induced by V_1 (containing only arcs connecting nodes in V_1). Let d_1 be a reverse topological ordering of G'_1. According to Lemma A.5, G_1 is backtrack-free along d_1. Hence, we can construct a solution X' to G'_1 by instantiating X_i to x and then instantiating the rest of the variables in V_1 in a backtrack-free fashion along d_1. Having instantiated the nodes in V_1, we can now extend X' to a full solution X of G' as follows. Let d be a topological ordering of G_p whose restriction to G'_1 is the reverse of d_1. The nodes in V_1 are already instantiated. According to Lemma 4.32, we can extend X' to a full solution of G' by instantiating the rest of the nodes, $V - V_1$, backtrack-free along d. Therefore, there exists a solution to G' in which $X_i = x$. □

REFERENCES

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

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

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

[4] J. F. Allen and P. J. Hayes. A common-sense theory of time. In *Proceedings of IJCAI-85, Los Angeles, CA*, pages 528–531, 1985.

[5] M. J. Almeida. *Reasoning About the Temporal Structure of Narrative.* PhD thesis, State University of New York, Buffalo, NY, 1987.

[6] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey. *BIT*, 25:2–23, 1985.

[7] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Algebraic Discrete Methods*, 8:177–184, 1987.

[8] B. Aspvall and Y. Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM Journal of Computing*, 9:827–845, 1980.

[9] R. C. Backhouse and B. A. Carŕe. Regular algebra applied to path-finding problems. *J. Inst. Math. Applications*, 15:161–186, 1975.

[10] U. Bertelé and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, New York, 1972.

[11] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1962.

[12] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[13] E. Davis. Private communication, 1989.

[14] T. Dean. Large-scale temporal data bases for planning in complex domains. In *Proceedings of IJCAI-87, Milan, Italy*, pages 860–866, 1987.

[15] T. Dean. Using temporal hierarchies to efficiently maintain large temporal databases. *JACM*, 36:687–718, 1989.

[16] T. Dean, J. Firby, and D. Miller. Hierarchical planning involving deadline, travel time and resources. *Computational Intelligence*, 4:381–398, 1990.

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

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

[19] R. Dechter. Constraint networks. In S. Shapiro, editor, *2nd Edition of Encyclopedia of Artificial Intelligence.* John Wiley, New York, 1992.

[20] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proceedings of IJCAI-89, Detroit, MI*, pages 271–277, 1989.

[21] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[22] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1988.

[23] Y. Deville and P. Van Hentenryck. An efficient arc consistency algorithm for a class of CSP problems. In *Proceedings of IJCAI-91, Sydney, Australia*, pages 325–330, 1991.

[24] S. Even. *Graph Algorithms.* Computer Science Press, Rockville, MD, 1979.

[25] E. C. Freuder. Synthesizing constraint expressions. *CACM*, 21:958–965, 1978.

[26] E. C. Freuder. A sufficient condition of backtrack-free search. *JACM*, 29:24–32, 1982.

[27] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, 1979.

[28] J. Gaschnig. *Performance Measurement and Analysis of Certain Search Algorithms.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1979.

[29] M. C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. Technical Report RRR-22-91, Center for Operations Research, Rutgers University, New Brunswick, NJ, 1991.

[30] I. Hamlet and J. Hunter. A representation of time for medical expert systems. In *Proceedings of the European Conference on AI in Medicine, Marseilles, France*, pages 112–119, 1987.

[31] S. Hanks and D. V. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of AAAI-86, Philadelphia, PA*, pages 328–333, 1986.

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

[33] K. Kahn and G. A. Gorry. Mechanizing temporal knowledge. *Artificial Intelligence*, 9:87–108, 1977.

[34] M. G. Kahn, J. C. Ferguson, E. H. Shortliffe, and L. M. Fagan. Representation and use of temporal information in ONCOIN—cancer therapy planning program. In M. K. Chytil and R. Engelbrecht, editors, *Expert Medical Systems*, pages 35–44. Sigma Press, 1987.

[35] H. Kautz and P. B. Ladkin. Integrating metric and qualitative temporal reasoning. In *Proceedings of AAAI-91, Anaheim, CA*, pages 241–246, 1991.

[36] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[37] P. B. Ladkin. Time representation: A taxonomy of interval relations. In *Proceedings of AAAI-86, Philadelphia, PA*, pages 360–366, 1986.

[38] P. B. Ladkin. Metric constraint satisfaction with intervals. Technical Report TR-89-038, International Computer Science Institute, Berkeley, CA, 1989.

[39] P. B. Ladkin and R. D. Maddux. On binary constraint networks. Technical report, Kestrel Institute, Palo Alto, CA, 1989.

[40] D. J. Lehmann. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4:59–76, 1977.

[41] C. E. Leiserson and J. B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. In *Proceedings—21st Annual Allerton Conference on Communication, Control, and Computing*, pages 204–213, 1983.

[42] Y. Z. Liao and C. K. Wong. An algorithm to compact a vlsi symbolic layout with mixed constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-2:62–69, 1983.

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

[44] A. K. Mackworth. Constraint satisfaction. In S. Shapiro, editor, *2nd Edition of Encyclopedia of Artificial Intelligence.* John Wiley, New York, 1992.

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

[46] J. Malik and T. O. Binford. Reasoning in time and space. In *Proceedings of IJCAI-83, Karlsruhe, West Germany*, pages 343–345, 1983.

[47] D. V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[48] I. Meiri. Faster constraint satisfaction algorithms for temporal reasoning. Technical Report TR-151, Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, CA, 1990.

[49] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[50] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[51] K. Nökel. Temporal matching: Recognizing dynamic situations from discrete measurements. In *Proceedings of IJCAI-89, Detroit, MI*, pages 1255–1260, 1989.

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

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

[54] D. S. Parker. Partial order programming. Technical Report CSD-870067, Computer Science Department, University of California, Los Angeles, CA, 1987.

[55] M. Poesio and R. J. Brachman. Metric constraints for maintaining appointments: Dates and repeated activities. In *Proceedings of AAAI-91, Anaheim, CA*, pages 253–259, 1991.

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

[57] Y. Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence.* MIT Press, Cambridge, MA, 1988.

[58] R. Shostak. Deciding linear inequalities by computing loop residues. *JACM*, 28:769–779, 1981.

[59] F. Song and R. Cohen. The interpretation of temporal relations in narrative. In *Proceedings of AAAI-88, St. Paul, MN*, pages 745–750, 1988.

[60] F. Song and R. Cohen. Temporal reasoning during plan recognition. In *Proceedings of AAAI-91, Anaheim, CA*, pages 247–252, 1991.

[61] R. E. Tarjan. Fast algorithms for solving path problems. *JACM*, 28:594–614, 1981.

[62] R. E. Tarjan. A unified approach to path problems. *JACM*, 28:577–593, 1981.

[63] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13:566–579, 1984.

[64] R. E. Valdés-Pérez. Spatio-temporal reasoning and linear inequalities. Technical Report AIM-875, Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1986.

[65] R. E. Valdés-Pérez. The satisfiability of temporal constraint networks. In *Proceedings of AAAI-87, Seattle, WA*, pages 256–260, 1987.

[66] P. VanBeek. Approximation algorithms for temporal reasoning. In *Proceedings of IJCAI-89, Detroit, MI*, pages 1291–1296, 1989.

[67] P. VanBeek. *Exact and Approximate Reasoning About Qualitative Temporal Relations.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1990.

[68] P. VanBeek. Reasoning about qualitative temporal information. In *Proceedings of AAAI-90, Boston, MA*, pages 728–734, 1990.

[69] J. F. A. K. VanBenthem. *The Logic of Time.* D. Reidel, 1983.

[70] S. A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 5:246–267, 1983.

[71] M. Vilain. A system for reasoning about time. In *Proceedings of AAAI-82, Pittsburgh, PA*, pages 197–201, 1982.

[72] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of AAAI-86, Philadelphia, PA*, pages 377–382, 1986.

[73] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2-trees, and minimum ifi networks. *Networks*, 13:159–167, 1983.

[74] B. C. Williams. Doing time: Putting qualitative reasoning on firmer ground. In *Proceedings of AAAI-86, Philadelphia, PA*, pages 105–113, 1986.