Combining Qualitative and Quantitative Constraints in Temporal Reasoning*

Itay Meiri†
Cognitive Systems Laboratory
Computer Science Department
University of California
Los Angeles, CA 90024
itay@cs.ucla.edu

October 12, 1995

Abstract

This paper presents a general model for temporal reasoning that is capable of handling both qualitative and quantitative information. This model allows the representation and processing of many types of constraints discussed in the literature to date, including metric constraints (restricting the distance between time points) and qualitative, disjunctive constraints (specifying the relative position of temporal objects). Reasoning tasks in this unified framework are formulated as constraint satisfaction problems and are solved by traditional constraint satisfaction techniques, such as backtracking and path consistency. New classes of tractable problems are characterized, involving qualitative networks augmented by quantitative domain constraints, some of which can be solved in polynomial time using arc and path consistency.

*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.

1 Introduction

In recent years, several constraint-based formalisms have been proposed for temporal reasoning, most notably Allen's interval algebra [1], Vilain and Kautz's point algebra [29], Dean and McDermott's time map [2], and metric networks (Dechter, Meiri, and Pearl [4]). 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 the existing formalisms can conveniently handle all forms of temporal knowledge. Qualitative approaches such as Allen's interval algebra and Vilain and Kautz's point algebra have difficulties in representing and reasoning about metric, numerical information, while the quantitative approaches exhibit limited expressiveness when it comes to qualitative information [4].

In this paper we offer a general, network-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 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 most types of constraints discussed in the literature to date.

The main contribution of this paper lies in providing a formal unifying framework for temporal reasoning, thereby generalizing the interval algebra, point algebra, and metric networks formalisms. In this framework, we are able to utilize constraint satisfaction techniques in solving several reasoning tasks. Specifically,

1. General networks 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. Path consistency algorithms can be used in preprocessing the input network to improve search efficiency or to compute an approximation to the minimal network.

4. We were able to identify two classes of tractable problems, solvable in polynomial time. The first consists of *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 consists of networks for which path consistency algorithms are exact.

We also show that our model compares favorably, both conceptually and computationally, with an alternative approach for combining quantitative and qualitative constraints, proposed by Kautz and Ladkin [10].

The paper is organized as follows. Section 2 formally defines the constraint types under consideration. The definitions of the new model are given in Section 3. Section 4 reviews and extends the hierarchy of qualitative networks. Section 5 discusses *augmented qualitative networks*—qualitative networks augmented by domain constraints. Section 6 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 7 provides concluding remarks.

2 The Representation Language

Consider a typical temporal reasoning problem. We are given the following information.

Example 2.1 *John and Fred work for a company that has 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?".

Involved are two types of temporal objects: points and intervals. Intervals correspond to time periods during which events occur or propositions hold, and points represent the beginning and ending points of some events, as well as neutral points of time. For example, in our story 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 the world" in our story. One possible choice for P_0 is 7:00 a.m. Temporal statements in the story are treated as constraints on the location of objects (such as intervals J and F and points $P_0, \ldots, P_4$) along the time line. There are two types of constraints: qualitative and quantitative. Qualitative constraints specify the relative position of paired objects. For instance, the fact that John and Fred met at a traffic light forces intervals J and F to overlap. Quantitative constraints place absolute bounds or restrict the

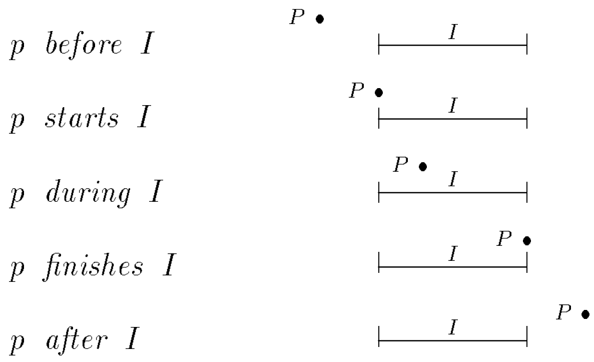

Figure 1: The basic relations between a point p and an interval I.

temporal distance between points. For example, the information on Fred's commuting time constrains the length of interval F, that is, the distance between P_3 and P_4. In the rest of this section we formally define qualitative and quantitative constraints, and the relationships between them.

2.1 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

$$(O_i \; r_1 \; O_j) \vee \cdots \vee (O_i \; r_k \; O_j), \tag{1}$$

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 [1]—*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 [29], 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 1 and in Table 1 (see also [28, 14]).

A subset of basic relations (of the same type) corresponds to an ambiguous, disjunctive relationship between objects. For example, Equation (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 $\{r_1, \ldots, r_k\}$. One qualitative constraint given in Example 2.1 reflects the fact that John

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

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 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 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 [1] and [29], and appear in Table 2 as T_{IA} and T_{PA}, respectively. The rest of the transitivity tables are shown in Tables 3–6.[1] Illegal combinations in Table 2 are denoted by $\emptyset$.

2.2 Quantitative Constraints

Quantitative constraints refer to absolute location or the distance between *points* [4]. There are two types of quantitative constraints:

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

	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 2: A full transitivity table.

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 3: Composition of PP and PI relations.

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

Table 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 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 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 example, one binary constraint given in our story 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, $P_1 - P_0 \in \{(5, 10)\}$.

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,

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

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

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

Table 7: The QUAN translation.

2.3 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 [10]).

- 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 $(-\infty, 0) \in \mathsf{QUAN}(C)$.

An alternative definition of QUAN is given in Table 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:

$$C' \oplus C'' = C' \oplus \mathsf{QUAN}(C''). \tag{2}$$

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

3 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 [20] and for metric networks [4].

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 2.1 is shown in Figure 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

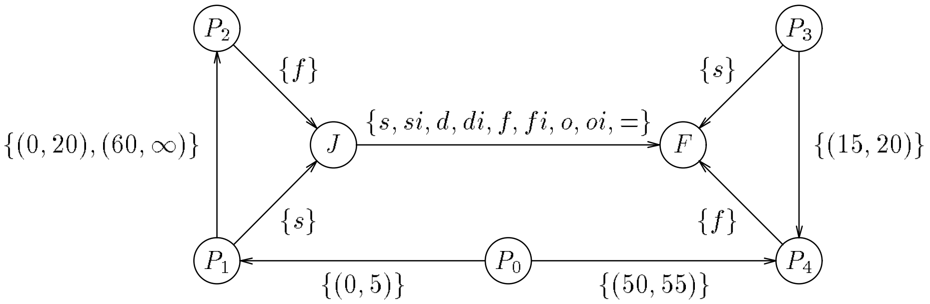

Figure 2: The constraint graph of Example 2.1.

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 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 2.1, whose constraints are shown in Table 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 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 minimal constraint between a given pair of variables X_i and X_j, and the full minimal network. The rest of the paper is concerned with solving these tasks.

	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 8: The minimal network of Example 2.1.

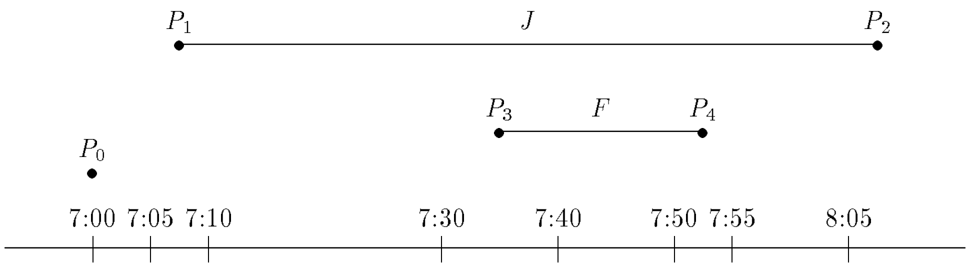

Figure 3: A feasible scenario.

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 [4, 29]. 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 and 5 we pursue "islands of tractability"—special classes of networks that admit polynomial solution. Then, in Section 6, we describe brute-force, exponential techniques that can handle any general network, and discuss the use of *path consistency* as an approximation scheme.

4 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 5 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 [26]. 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 [29]) 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. □

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 can be represented by the CPA network $J^- \leq I^- \leq I^+ \leq J^+$, where $I = [I^-, I^+]$ and $J = [J^-, J^+]$. 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 $O(n^2)$ [27, 18], but it becomes NP-complete for IA networks [29], 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 [8].

[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.

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.

5 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.

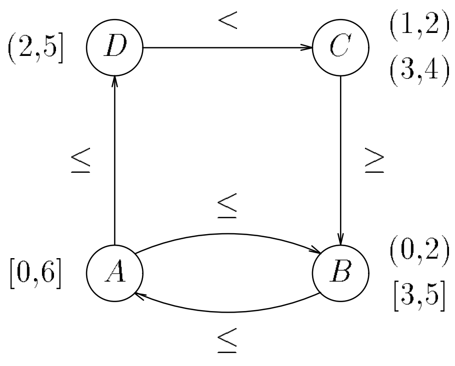

Figure 4: A CPA network over multiple-intervals domains.

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, 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 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 $I = \{I_1, \ldots, I_k\}$, 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 $a_1, b_1, \ldots, a_k, b_k$, which are called the *extreme points* of D_i. These extreme points are maintained in an array of size $2k$. In an accompanying 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 $I = \{I_1, \ldots, I_k\}$ can be regarded as a set of real numbers consisting of the values in $I_1 \cup \cdots \cup I_k$. Thus, we have $\inf(D_i) = \inf(I_1) = a_1$ and $\sup(D_i) = \sup(I_k) = b_k$. As with

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

	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 9: 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 10: Complexity of computing the minimal domains in tractable augmented qualitative networks.

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, the interesting tasks can be solved in polynomial time using local consistency algorithms such as *arc consistency (AC)* and *path consistency (PC)*.

Tables 9 and 10 summarize the results presented in this section regarding the complexity of determining consistency and computing the minimal domains in augmented qualitative networks. Each entry gives the consistency level that can be used to solve the corresponding task (AC,PC, or both), and the timing of the best algorithm discussed in this paper.

5.1 Arc and Path Consistency

Let us review the definitions of arc and path consistency [16, 20].

Definition 5.1 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.

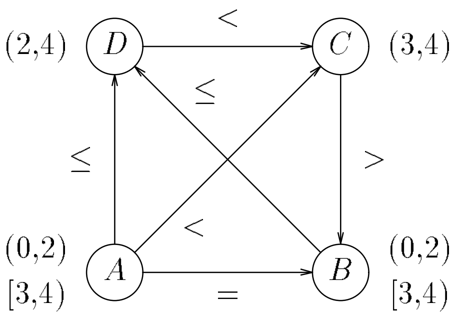

Figure 5: An arc- and path-consistent form of the network in Figure 4.

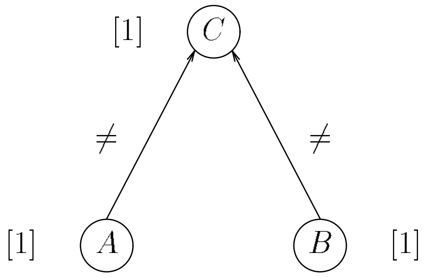

Figure 6: An augmented PA network.

Definition 5.2 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 5 shows an equivalent, arc- and path-consistent form of the network in Figure 4. $\Box$

Note that our definition of path consistency is slightly different than the original definition [16], since it disregards domain constraints. The following example illustrates the difference between the two definitions.

Example 5.3 Consider the network in Figure 6. The network is path consistent according to Definition 5.2, 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. $\Box$

The most common arc consistency algorithm that converts a network into an equivalent arc-consistent form is algorithm AC-3 [16], shown in Figure 7. AC-3 repeatedly applies the function REVISE$((i,j))$, which makes arc $i \rightarrow j$ consistent, until a fixed point, at which

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 7: AC-3—an arc consistency algorithm.

all arcs are consistent, is reached. The function REVISE restricts the domain D_i using quantitative operations on constraints[5]:

$$D_i \leftarrow D_i \oplus D_j \otimes \mathsf{QUAN}(C_{ji}). \tag{3}$$

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* [5].

Definition 5.4 (Dechter and Pearl [5]) 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 [5], shown in Figure 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 [16, 29, 26]. Path consistency algorithms impose local consistency among triplets of variables (i, k, j) by using a relaxation operation:

$$C_{ij} \leftarrow C_{ij} \oplus C_{ik} \otimes C_{kj}. \tag{4}$$

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

[5]Note that Equation (3) is the temporal equivalent of Mackworth's REVISE, when the latter 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 $j \rightarrow i, j < i$ **do**
3. X $\leftarrow$ REVISE$((j, i))$
4. **end**

Figure 8: DAC—a directional arc consistency algorithm.

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 9: PC-2—a path consistency algorithm.

We shall use an efficient path consistency algorithm, PC-2 [16], shown in Figure 9. The function REVISE$((i, k, j))$ performs the relaxation operation of Equation (4) and returns **true** if the constraint C_{ij} is changed. Algorithm PC-2 runs to completion in $O(n^3)$ time [17]. Recently, path-consistency algorithms were evaluated empirically in [23, 22, 24].

5.2 The Precedence Graph

Many of the algorithms presented in this section make use of an auxiliary data structure, called a *precedence graph* (see also [18, 27]), which displays precedence relations between variables.

Definition 5.5 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.

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

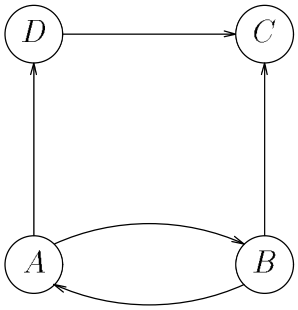

Figure 10: The precedence graph of the network in Figure 4.

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 is depicted in Figure 10. □

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

Theorem 5.6 (Van Beek [27]) *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*[6] *in G_p, $\{=\} \subseteq C_{ij}$.*

According to Theorem 5.6 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 5.6 [27]. 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.

[6]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.

Lemma 5.7 *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.

Definition 5.8 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,
$$D_i^r = \bigoplus_{j \in C_i} D_j. \tag{5}$$
- An edge $C_i \to C_j \in E^r$ if and only if there exists an edge $i \to j \in E_p$ 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,
$$C_{ij}^r = \bigoplus_{k \in C_i, l \in C_j} C_{kl}. \tag{6}$$

Note that the intersection operations in Equations (5) and (6) may result in an empty domain or an empty constraint. This may occur only if the input network G is inconsistent.

Definition 5.8 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 5.6. 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 (5)), because each node belongs to exactly one component, and $O(e)$ constraint intersections (Equation (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 $O(n^2k)$.

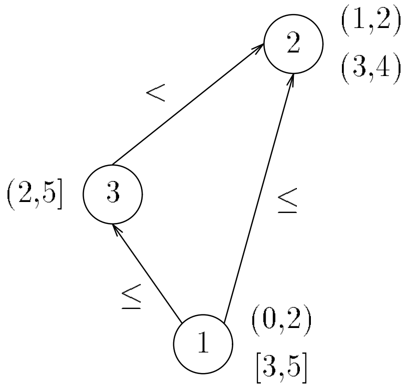

Figure 11: The reduced network of the network in Figure 4.

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.

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. The strongly connected components in its precedence graph (shown in Figure 10) are $C_1 = \{A, B\}$, $C_2 = \{C\}$, and $C_3 = \{D\}$. The reduced network is shown in Figure 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 5.9 *Any nonempty path-consistent PA network is consistent.*

Proof By Theorem 5.6 and Lemma 5.7.[7] □

Lemma 5.10 *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 strongly connected components in G_p. If $i \to j \in E$ and $k \to l \in E$, where $i, k \in C'$ and $j, l \in C''$, then $C_{ij} = C_{kl}$.*

Proof See Appendix A. □

[7] Another proof is given by Ladkin and Maddux in [14].

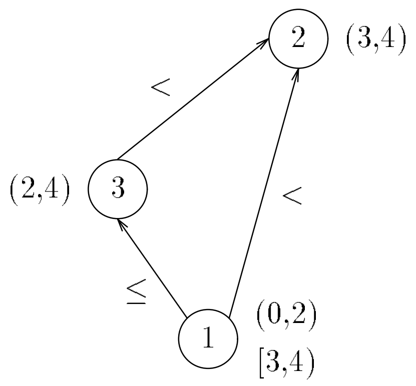

Figure 12: The reduced network of the network in Figure 5.

From Lemma 5.7 we have the following corollary.

Corollary 5.11 *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 Proposition 5.9, Lemma 5.10, and Corollary 5.11, we obtain the following properties of the reduced network of an arc- and path-consistent PA network.

Lemma 5.12 *The reduced network of a nonempty arc- and path-consistent augmented PA network is (1) nonempty and (2) arc and path consistent.*

Proof From Proposition 5.9, the underlying qualitative network is consistent. From Lemma 5.10 and Corollary 5.11, 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 (5) and (6), we may choose any domain D_j, $j \in C_i$, as the domain D_i^r (from Corollary 5.11), and we may choose any constraint C_{kl}, $k \in C_i$, $j \in C_l$, as the constraint C_{ij}^r (from Lemma 5.10). 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 5 is shown in Figure 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. □

5.3 Augmented CPA Networks

This subsection is organized as follows. Section 5.3.1 presents a solution technique for CPA networks over discrete domains. Then, we discuss CPA networks over multiple-intervals domains: in Section 5.3.2 we present solution techniques for acyclic networks, and in Section 5.3.3 we extend those techniques to cyclic networks.

5.3.1 Discrete Domains

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

Theorem 5.13 *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 5.13 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 [19]). Deville and Van Hentenryck [6] have devised a special-purpose arc-consistency algorithm that works for functional and monotone constraints. This algorithm runs in $O(ek)$ time for CPA networks over discrete domains (The $=$ constraints are functional, while the $<$ and the $\leq$ constraints are monotone). 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 5.14 Consider the network in Figure 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 5.3.3 we shall show that the minimal domains can be computed by establishing both arc and path consistency.

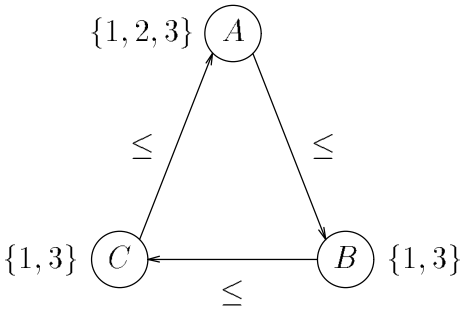

Figure 13: An arc-consistent CPA network over discrete domains.

5.3.2 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 [7] along any topological ordering of the precedence graph.

Lemma 5.15 *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 $j \to i \in E_p$). 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. $\Box$

As an immediate corollary of Lemma 5.15, we have the following theorem, showing that arc consistency decides the consistency of an acyclic CPA network.

Theorem 5.16 *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 14. Based on the solution

Algorithm Solve-Acyclic-CPA

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

Figure 14: Solve-Acyclic-CPA—an algorithm for constructing a solution to an acyclic CPA network over multiple-intervals domains.

technique used in the proof of Lemma 5.15, 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 5.17 *The complexity of algorithm* AC-3 *for a PA network over multiple-intervals domains is $O(en^2k^2)$.*

Proof See Appendix A. □

From Lemma 5.17, 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 5.15 we needed only directional arc consistency, Lemma 5.15 and consequently Theorem 5.16 can be modified as follows.

Lemma 5.18 *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 5.19 *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 5.19, 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.

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 15: 2DAC—an arc-consistency algorithm for acyclic CPA networks.

Lemma 5.20 *The complexity of algorithm* DAC *for an acyclic CPA network over multiple-intervals domains is* $O(e \log k)$.

Proof See Appendix A. □

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 5.21 *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 5.17) 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 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 11. Running DAC along the ordering $d = (1, 3, 2)$ results in the directional arc-consistent network depicted in Figure 16. Then, running DAC along the reverse ordering $d_r = (2, 3, 1)$ yields the arc-consistent network of Figure 12. □

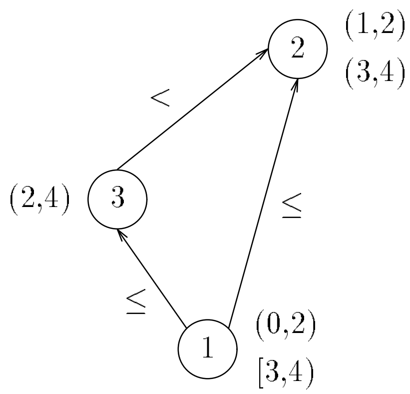

Figure 16: A directional arc-consistent form of the network in Figure 11.

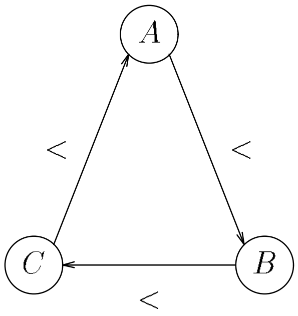

Figure 17: An arc-consistent CPA network.

5.3.3 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 5.22 Consider the CPA network in Figure 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 5.23 *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 5.12, the reduced network G^r is both nonempty and arc consistent. By Theorem 5.16, G^r is consistent. Hence, G is consistent. □

Theorem 5.23 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 5.24 *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 5.12, the reduced network G^r is nonempty and arc consistent. By Theorem 5.21, the domains of G^r are minimal. Since, as explained in Section 5.2, 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 5.2, 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, $O(n^2k)$. Recently, a class of networks called *row-convex* generalizing CPA networks was identified and analyzed [25].

5.4 Augmented PA 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 5.25 *Deciding the consistency of a PA network over discrete domains is NP-hard.*

Proof Straightforward reduction from graph coloring. □

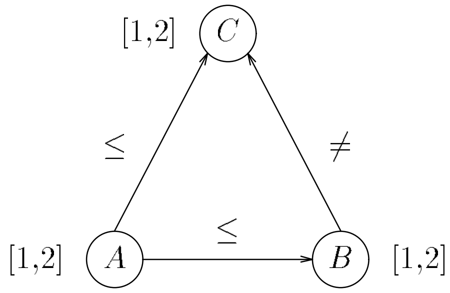

Figure 18: An arc-consistent PA network over single-interval domains.

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. This model was later extended in [11, 12].

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 5.26 Consider the arc-consistent network in Figure 18. The precedence graph of this network consists of two arcs: $A \to B$ and $A \to C$. 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 5.27 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 5.28 *The restricted network of an arc-consistent PA network over almost-single-interval domains is arc consistent.*

Proof See Appendix A. □

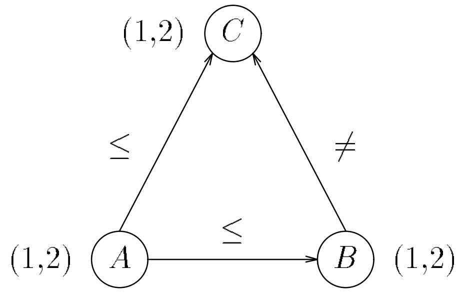

Figure 19: The restricted network of the network in Figure 18.

Although, as shown in Example 5.26, 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 5.29 *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 5.28, 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 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 $j \to i \in E_p$). 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. $\square$

Illustration Consider the network in Figure 18. Its restricted network is depicted in Figure 19. It can be easily verified that the restricted network is backtrack-free along the orderings $d_1 = (A, B, C)$ and $d_2 = (A, C, B)$. $\square$

As a corollary to Lemma 5.29, we have the following theorem.

Theorem 5.30 *A nonempty arc-consistent acyclic PA network over almost-single-interval domains is consistent.*

In order to make use of Theorem 5.30 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 5.31 *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 5.30 and Lemma 5.31, 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 20. Based on the solution technique used in the proof of Lemma 5.29, 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 5.17, 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 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 5.32 *Algorithm* 4DAC *computes an arc-consistent network.*

Proof See Appendix A. □

Algorithm Solve-Acyclic-PA

for $i := 1$ **to** n **do**
if D'_i consists of a single value v **then**
$v_i \leftarrow v$
else begin
$v_i \leftarrow$ a value in D'_i
$L \leftarrow \emptyset$
for each j such that $j \rightarrow i \in E_p$ **do**
$L \leftarrow L \cup \{$a value in D'_i that is consistent with $v_j\}$
$v_i \leftarrow \max(\{v_i\} \cup L)$
$N \leftarrow \emptyset$
for each $j < i$ such that C_{ij} is $\neq$ **do**
$N \leftarrow N \cup \{v_j\}$
$v_i \leftarrow$ a value in $[v_i, \sup(D'_i)) - N$
end
end

Figure 20: Solve-Acyclic-PA—an algorithm for constructing a solution to an acyclic PA network over almost-single-interval domains.

Algorithm 4DAC

$d \leftarrow$ a topological ordering of G_p
$d_r \leftarrow$ the reverse of d
run DAC along d
run DAC along d_r
run DAC along d
run DAC along d_r

Figure 21: 4DAC—an arc consistency algorithm for acyclic PA networks over almost-single-interval domains.

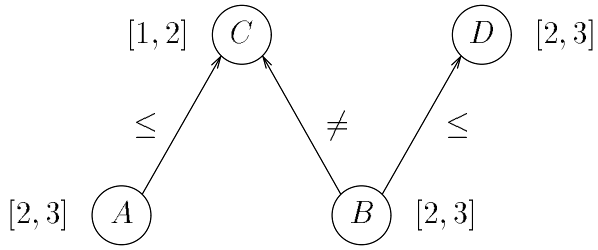

Figure 22: A PA network over single-interval domains.

Lemma 5.32 guarantees that four applications of DAC are sufficient to compute an arc-consistent network. Example 5.33 shows that we cannot do better than that—four applications are indeed necessary.

Example 5.33 Consider the network in Figure 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 5.34 *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 5.35 *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.*

Proof See Appendix A. □

Arc consistency does not compute the minimal domains of the *input* network, however. For example, in the arc-consistent network of Figure 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 5.35, 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 5.35. 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 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 5.36 *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 5.12, the reduced network G^r is nonempty and arc consistent. By Theorem 5.30, G^r is consistent. Hence, G is consistent. □

Theorem 5.36 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 5.37 *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 5.28 G' is arc consistent and, from Lemma 5.12, 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 5.35, the domains of $(G')^r$ are minimal. Since, as explained in Section 5.2, 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.

6 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.

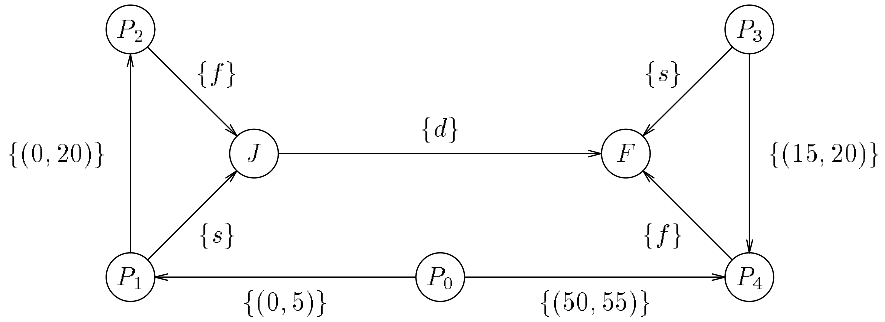

Figure 23: A singleton labeling of the constraint graph of Figure 2.

We return to the network representation described in Section 3. Namely, in contrast with Section 5, 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 $i \to j$ 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 [4]. An STP network can be solved in $O(n^3)$ time [4]. Thus, the overall complexity of this decomposition scheme is $O(n^3 k^e)$, 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.

Illustration Consider the constraint graph of Figure 2. One singleton labeling is shown in Figure 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 24. $\square$

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 algorithms for metric networks [4]. 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. For further details see [4]. Recent empirical evaluations of various temporal backtracking

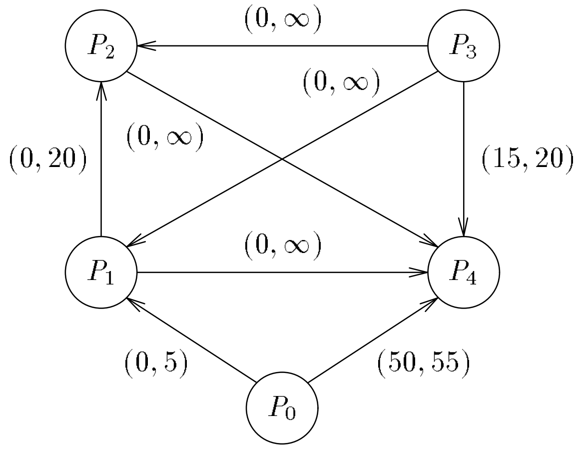

Figure 24: The STP network of the singleton labeling of Figure 23.

algorithms are reported in [24, 15].

Imposing local consistency among subsets of variables may serve as a preprocessing step to improve backtrack. This strategy has been proven successful (see [3]), 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 [3]. Following this rationale, we next show that path consistency, which in general networks amounts to the least amount of preprocessing,[8] can be achieved in polynomial time.

To assess the complexity of PC-2 in the context of general networks, we introduce the notion of a *range* of a network [4]. We first consider the case of an *integral* network, where the extreme points of all metric constraints are integers. The *range* of a metric constraint C, represented by an interval set $\{I_1, \ldots, I_k\}$, is $\sup(I_k) - \inf(I_1)$. The range of the *network* is the maximum range over all its metric constraints. For a *rational* network, whose extreme points are rational numbers, 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. It can be shown that all operations on the input, rational network can be simulated on its equivalent integral network (Ladkin [13]). The next theorem shows that the timing of PC-2 is bounded by $O(n^3R^3)$, where R is the range of the network.

Theorem 6.1 *Algorithm* PC-2 *calls* REVISE $O(n^3R)$ *times, and its timing is bounded by* $O(n^3R^3)$*, where* R *is the range 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

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

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 [16]. 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 $O(n^3R^3)$. □

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 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 6.2 *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 [4] 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 6.2 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 6.2 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 6.2, path consistency is not guaranteed to compute the minimal network. According to Proposition 6.2, 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.

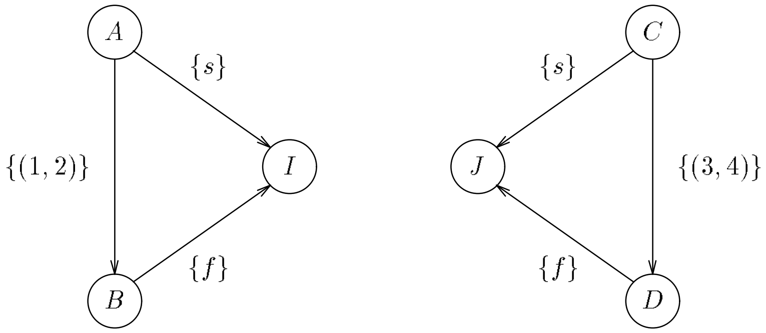

Figure 25: A path-consistent singleton labeling.

Example 6.3 Consider the network in Figure 25. It consists of two intervals, $I = [A, B]$ and $J = [C, D]$, and two metric constraints on their length,

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

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 6.2, 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 6.4 *Any path-consistent singleton labeling is minimal.*

Proof We need to show that the qualitative constraints are minimal. According to Proposition 6.2 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.

7 Conclusions

We describe a general network-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 of all qualitative constraints between temporal objects. We use constraints satisfaction techniques in solving reasoning tasks in this model. In particular, general networks can be solved either 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. A detailed desctiption of tractable classes for the pure qualitative networks is given in [9, 21].

Other tractable classes were obtained by augmenting PA and CPA networks with various domain constraints. We showed that some of these networks can be solved using arc and path consistency.

Kautz and Ladkin [10] 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. All information is stored in a single network and therefore constraint propagation takes place in the knowledge level itself.
2. In our model we are able to establish 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 $O(n^5 R^3)$, compared to $O(n^3 R^3)$ in our model.

Future research should enrich the representation language to facilitate modeling of more involved reasoning tasks. In particular, non-binary constraints (for example, "If John leaves home before 7:15 a.m., he arrives at work before Fred") should be incorporated in our model.

Acknowledgments

I would like to thank Rina Dechter and Judea Pearl for providing helpful comments on an earlier draft of this paper. Also thanks to Rina Dechter and Eddie Schwalb for the effort in revising the final manuscript.

A Proofs

Proof of Lemma 5.7 The *if* part is trivial—if C_{ij} is $=$ then, by definition, both $i \to j \in E_p$ and $j \to i \in E_p$, 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 $i_1 = i \to i_2 \to \cdots \to i_k = j$ 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,

$$C_{ij} \subseteq C_{i_1,i_2} \otimes \cdots \otimes C_{i_{k-1},i_k} \subseteq \{<,=\}. \tag{7}$$

Similarly, there exists a directed path $j_1 = j \to j_2 \to \cdots \to j_k = i$ 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,

$$C_{ij} \subseteq C_{j_k,j_{k-1}} \otimes \cdots \otimes C_{j_2,j_1} \subseteq \{>,=\}. \tag{8}$$

From Equations (7) and (8), $C_{ij} \subseteq \{=\}$, and since all constraints are nonempty, C_{ij} must be $=$. $\Box$

Proof of Lemma 5.10 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 5.7, C_{jl} is $=$, thus
$$C_{il} \subseteq C_{ij}. \tag{9}$$
Similarly, from path consistency, $C_{ij} \subseteq C_{il} \otimes C_{lj}$. According to Lemma 5.7, C_{lj} is $=$, thus
$$C_{ij} \subseteq C_{il}. \tag{10}$$
From Equations (9) and (10), $C_{ij} = C_{il} = C_{kl}$.

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 $C_{ij} = C_{kl}$. $\Box$

Proof of Theorem 5.13 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 (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 [17]. 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 5.17 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 5.20 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 5.21**.

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. □

Proof of Theorem 5.21 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 5.15, 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 $X_i = x$. $\Box$

Proof of Lemma 5.28 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 5.31 Consider the operation of REVISE (Equation (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 5.32 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. $\Box$

Proof of Lemma 5.34 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 (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 $O(e(k+n))$. $\Box$

The next lemma is needed for the proof of **Theorem 5.35**.

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 5.28, 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 $i \rightarrow j \in E_p$). 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 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 5.35 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 5.29, 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] J. F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26:832–843, 1983.

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

[3] 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.

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

[5] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1988.

[6] 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.

[7] E. C. Freuder. A sufficient condition of backtrack-free search. *JACM*, 29:24–32, 1982.

[8] 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.

[9] 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.

[10] H. Kautz and P. B. Ladkin. Integrating metric and qualitative temporal reasoning. In *Proceedings of AAAI-91, Anaheim, CA*, pages 241–246, 1991.

[11] M. Koubarakis. Dense time and temporal constraints with $\neq$. In *Proceedings of KR-92, Cambridge, Massachusetts*, pages 24–35, 1992.

[12] M. Koubarakis. From local to global consistency in temporal constraint networks. In *Proceedings of Principles and Practice of Constraint Programming - CP95, Cassi, France*, pages 53–69, 1995.

[13] P. B. Ladkin. Metric constraint satisfaction with intervals. Technical Report TR-89-038, International Computer Science Institute, Berkeley, CA, 1989.

[14] P. B. Ladkin and R. D. Maddux. On binary constraint networks. Technical report, Kestrel Institute, Palo Alto, CA, 1989.

[15] P. B. Ladkin and A. Reinefeld. Effective solutions of qualitative interval constraint problems. *Artificial Intelligence*, 57:105–124, 1992.

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

[17] 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.

[18] 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.

[19] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[20] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[21] B. Nebel and Burckert H.J. Reasoning about temporal relations: a maximal tractable subclass of allen's interval algebra. In *Proceedings of National Conference of Artificial Intelligence, Seattle, Wa.*, pages 356–361, 1994.

[22] 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.

[23] E. Schwalb and R. Dechter. Coping with disjunctions in temporal constraint satisfaction problems. In *The National Conference of Artificial Intelligence (AAAI-93), Washington DC*, pages 2–8, 1993.

[24] E. Schwalb and R. Dechter. Processing temporal constraint networks. Technical report, Information and Computer Science, UC-Irvine, Irvine, CA, 1995.

[25] P. van Beek and R. Dechter. On the minimality and global consistency of row-convex constraint networks. *J. ACM*, 42(3):543–561, 1995.

[26] P. VanBeek. *Exact and Approximate Reasoning About Qualitative Temporal Relations.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1990.

[27] P. VanBeek. Reasoning about qualitative temporal information. In *Proceedings of AAAI-90, Boston, MA*, pages 728–734, 1990.

[28] M. Vilain. A system for reasoning about time. In *Proceedings of AAAI-82, Pittsburgh, PA*, pages 197–201, 1982.

[29] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of AAAI-86, Philadelphia, PA*, pages 377–382, 1986.