Chapter 8

Processing Qualitative Temporal Constraints

Alfonso Gerevini

In this chapter we provide an overview of the main techniques for processing qualitative temporal constraints. We survey a collection of algorithms for solving fundamental reasoning problems in the context of Allen's Interval Algebra, Vilain and Kautz's Point Algebra, and of other tractable and intractable classes of temporal relations. These problems include determining the satisfiability of a given set S of constraints; finding a consistent instantiation of the variables in S; deducing new (implicit) constraints from S, and computing the minimal network representation of S.

8.1 Introduction

Reasoning about qualitative temporal information has been an important research field in artificial intelligence (AI) for two decades, and it has been applied in the context of various AI areas. Such areas include knowledge representation (e.g., [Schmiedel, 1990; Miller, 1990; Schubert and Hwang, 1989; van Beek, 1991; Artale and Franconi, 1994]), natural language understanding (e.g., [Allen, 1984; Miller and Schubert, 1990; Song and Cohen, 1988]), commonsense reasoning (e.g., [Allen and Hayes, 1985]), diagnostic reasoning and expert systems (e.g., [Nökel, 1991; Brusoni *et al.*, 1998]), reasoning about plans (e.g., [Allen, 1991a; Tsang, 1986; Kautz, 1987; Kautz, 1991; Weida and Litman, 1992; Song and Cohen, 1996; Yang, 1997]) and scheduling (e.g., [Rit, 1986]).

Qualitative temporal information can often be represented in terms of temporal constraints among point or interval variables over a linear, continuous and unbound time domain (e.g., the rational numbers). In the following we give an example from an imaginary trains transportation domain, that will be used through the rest of the chapter to illustrate the prominent approaches to reasoning about qualitative temporal information expressible as binary temporal constraints.

Suppose that we want to represent the temporal information contained in the following simple story (see Figure 8.1):[*]

- *during its travel from city* `C1` *to city* `C2`*, train* `T1` *stopped first at the station* `S1` *and then at station* `S2`*;*

[*]This is a revised and extended version of an example that we gave in [Gerevini, 1997].

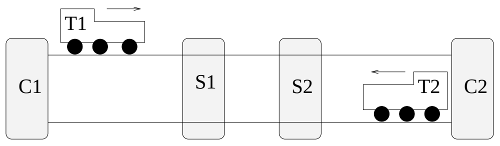


Figure 8.1: Pictorial description of the trains example

- *train* `T2` *traveled in the opposite direction of* `T1` *(i.e., from* `C2` *to* `C1`*);*
- *during its trip,* `T2` *stopped first at* `S2` *and then at* `S1`*;*
- *when* `T2` *arrived at* `S1`, `T1` *was stopping there too;*
- `T1` *and* `T2` *left* `S1` *at different times;*
- `T1` *arrived at* `C2` *before* `T2` *arrived at* `C1`.

These sentences contain some explicit qualitative information about the ordering of the time intervals during which the events described occurred. In particular:

- from the first sentence we can derive that the intervals of time during which `T1` stopped at `S1` (`at(T1,S1)`) and at `S2` (`at(T1,S2)`) are *contained* in the interval of time during which `T1` traveled from `C1` to `C2` (`travel(T1,C1,C2)`);
- from the second sentence we can derive that `at(T1,S1)` is *before* `at(T1,S2)`;
- from the third and the fourth sentences we can derive that `at(T2,S1)` and `at(T2,S2)` are *during* `travel(T2,C2,C1)` and that `at(T2,S2)` is *before* `at(T2,S1)`;
- from the fourth sentence we can derive that the starting time of `at(T1,S1)` *precedes* the starting time of `at(T2,S1)` and that the starting time of `at(T2,S1)` *precedes* the end time of `at(T1,S1)`;
- from the fifth sentence we can derive that `at(T1,S1)` and `at(T2,S1)` cannot finish at the same time (i.e., that the end times of `at(T1,S1)` and `at(T2,S1)` are different time points); and,
- finally, from the last sentence we can derive the constraint that the end time of `travel(T1,C1,C2)` is *before* the end time of `travel(T2,C2,C1)`.[*]

Suppose that in addition we know that no more than one train can stop at `S2` at the same time (say, because it is a very small station). Then we also have that

- `at(T2,S2)` is *disjoint* from `at(T1,S2)`.

[*]Despite when `T2` arrived at `S1` `T1` was still there, it is possible that `T2` had to stay at `S1` enough time to allow T1 to arrive at `C2` before the arrival of `T2` at `C1`.

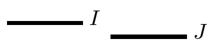
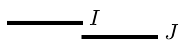
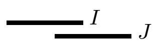
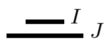
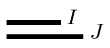
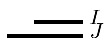
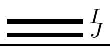
Relation	**Inverse**	**Meaning**
$I\ before\ (b)\ J$	$J\ after\ (a)\ I$	
$I\ meets\ (m)\ J$	$J\ met\text{-}by\ (mi)\ I$	
$I\ overlaps\ (o)\ J$	$J\ overlapped\text{-}by\ (oi)\ I$	
$I\ during\ (d)\ J$	$J\ contains\ (c)\ I$	
$I\ starts\ (s)\ J$	$J\ started\text{-}by\ (si)\ I$	
$I\ finishes\ (f)\ J$	$J\ finished\text{-}by\ (fi)\ I$	
$I\ equal\ (eq)\ J$	$J\ equal\ I$	

Figure 8.2: The thirteen basic relations of IA. IA contains 2^{13} relations, each of which is a disjunction of basic relations.

In terms of relations in Allen's Interval Algebra (IA) [Allen, 1983] (see Figure 8.1), the temporal information that we can derive from the story include the following constraints (assertions of relations) in IA, where **B** is the set of all the thirteen basic relations:[*]

(1) `at(T1,S1)` {during} `travel(T1,C1,C2)`
`at(T1,S2)` {during} `travel(T1,C1,C2)`

(2) `at(T2,S1)` {during} `travel(T2,C2,C1)`
`at(T2,S2)` {during} `travel(T2,C2,C1)`

(3) `at(T1,S1)` {before} `at(T1,S2)`
`at(T2,S2)` {before} `at(T2,S1)`

(4) `at(T1,S1)` {overlaps, contains, finished-by} `at(T2,S1)`

(5) `at(T1,S1)` **B** – {equal, finishes, finished-by} `at(T2,S1)`

(6) `travel(T1,C1,C2)` {before, meets, overlaps, starts, during}
`travel(S2,C2,C1)`

(7) `at(T2,S2)` {before, after} `at(T1,S2)`

A set of temporal constraints can be represented with a *constraint network* [Montanari, 1974], whose vertices represent interval (point) variables, and edges are labeled by the relations holding between these variables. Figure 8.3 gives a portion of the constraint network

[*]We specify such constraints using the set notation. Each set of basic relations denotes the *disjunction* of the relations in the set. E.g., I {before, after} J means $(I\ before\ j) \vee (I\ after\ j)$. Note, however, that when we consider a set of constraints (assertions of relations), such a set should be interpreted as the conjunction of the constraints forming it.

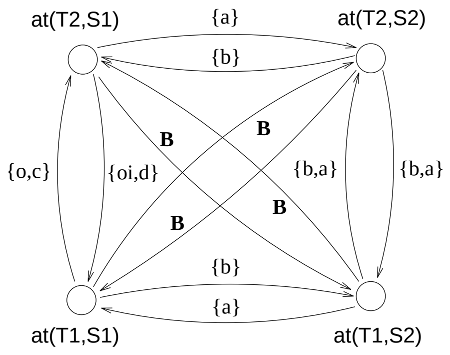


Figure 8.3: A portion of the interval constraint network for the trains example. Since the constraints between `at(T2,S1)` and `at(T1,S2)` are not explicit, they are assumed to be the universal relation **B**.

representing the temporal information of the trains example formalized using relations in IA. (The names of the relations are abbreviated using the notation introduced in Figure 8.1.)

By reasoning about the temporal constraints provided by the previous simple story, we can determine that the story is temporally consistent; we can deduce new constraints that are implicit in the story, such as that `travel(T1,C1,C2)` and `travel(T2,C2,C1)` must have more than one common time point, i.e. that

`travel(T1,C1,C2)` $\mathbf{B} - \{\text{b, a, m, mi}\}$ `travel(T2,C2,C1)`

holds; we can strengthen explicit constraints (e.g., we deduce that `at(T2,S2)` must be before `at(T1,S2)`, ruling out the possibility that `at(T2,S2)` is after `at(T1,S2)`, because `at(T2,S2)` {after} `at(T1,S2)` is *not* feasible); finally, we can determine that the ordering and interpretation of the interval endpoints in Figure 8.4 is consistent with all the (implicit and explicit) temporal constraints in the story.

Note that, if the supplementary information that `T1` stopped at `S2` *before* `T2` were provided, then the story would be temporally *in*consistent. This is because the explicit temporal constraints in the story imply that `T2` left `S2` before `T1` left `S1`, which precedes the arrival of `T1` at `S2`.

More in general, given a set of qualitative temporal constraints, fundamental reasoning tasks include:

- *Determining consistency* (satisfiability) of the set.
- *Finding a consistent scenario*, i.e., an ordering of the temporal variables involved that is consistent with the constraints provided, or *a solution*, i.e., an interpretation of all the temporal variables involved that is consistent with the constraints provided.
- *Deducing new constraints* from those that are known and, in particular, *computing the strongest relation* that is entailed by the input set of constraints between two particular

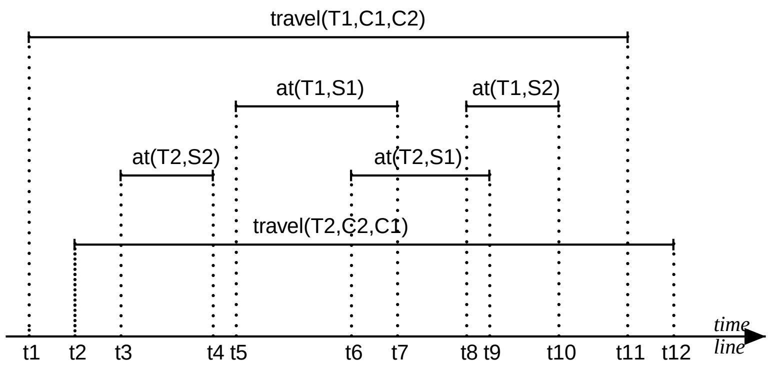


Figure 8.4: A consistent ordering (scenario) and an interpretation (solution) for the interval endpoints in the trains example.

variables, between one particular variable and all the others, or between every temporal variable and every other variable.*

Clearly, consistency checking and finding a solution (or a consistent scenario) are related problems, since finding a solution (consistent scenario) for an input set of constraints determines that the set is consistent. Finding a consistent scenario and finding a solution are strictly related as well, since from a solution for the first problem we can easily derive a solution for the second, and viceversa.

A solution for a set of constraints is also called a *consistent instantiation* of the variables involved, and an *interval realization*, if the variables are time intervals [Golumbic and Shamir, 1993].

In the context of the constraint network representation, the problem of computing the strongest entailed relations between all the pairs of variables corresponds to the problem of computing the *minimal network* representation.† Figure 8.5 gives the minimal network representation of the constraints in Figure 8.3.

The minimal network of an input set of constraints can be implemented as a matrix M, where the entry M$[i, j]$ is the strongest entailed relation between the ith-variable and the jth-variable. Therefore, once we have the minimal network representation, we can derive in constant time the strongest entailed relation between every temporal variable and every other temporal variable.

In some alternative graph-based approaches, the task of computing the strongest entailed relation between two particular variables v and w is sometimes called "querying" the relation between the two vertices of the graph representing v and w. Here we will also call this task "computing the *one-to-one* relation between v and w". Finally, we will call the problem of

*A relation R_1 is stronger than a relation R_2 if R_1 implies R_2 (e.g., "$<$" implies "$\leq$").

†Moreover, note that in the literature this problem is also called computing the "deductive closure" of the constraints [Vilain *et al.*, 1990], computing the "minimal labels" (between all pairs of intervals or points) [van Beek, 1990], the "minimal labeling problem" [Golumbic and Shamir, 1993], and computing the "feasible [basic] relations" [van Beek, 1992].

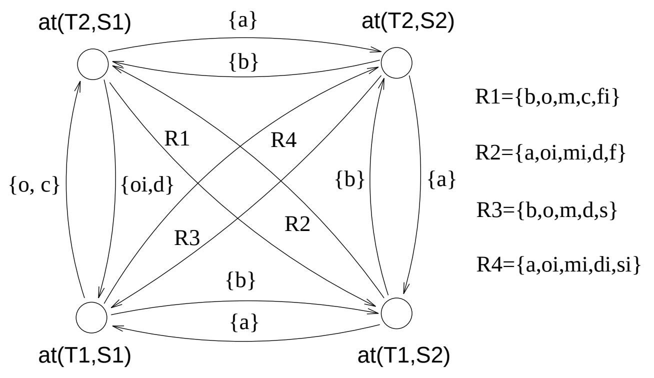


Figure 8.5: The minimal network of the constraint network in Figure 8.3

computing the strongest entailed relation between a (source) variable s and all the others "computing the *one-to-all* relations for s".

All these reasoning tasks are NP-hard if the assertions are in the Interval Algebra [Vilain *et al.*, 1990], while they can be solved in polynomial time if the assertions are in the Point Algebra (PA) ([Ladkin and Maddux, 1988; Vilain *et al.*, 1990; van Beek, 1990; van Beek, 1992; Gerevini and Schubert, 1995b]), or in some subclasses of IA, such as Nebel and Bürckert's ORD-Horn algebra [Nebel and Bürckert, 1995].

Table 8.1 summarizes the computational complexity of the best algorithms, known at the time of writing, for solving fundamental reasoning problems in the context of the following classes of qualitative temporal relations: PA, PA^c, SIA, SIA^c, ORD-Horn, IA, and $\mathcal{A}_3$. PA consists of eight relations between time points, $<$, $\leq$, $=$, $\geq$, $>$, $\neq$, $\emptyset$ (the empty relation) and $?$ (the universal relation); PA^c is the (convex) subalgebra of PA containing all the relations of PA except $\neq$; SIA consists of the set of relations in IA that can be translated into conjunctions of relations in PA between the endpoints of the intervals [Ladkin and Maddux, 1988; van Beek, 1990; van Beek and Cohen, 1990]; SIA^c is the (convex) subalgebra of SIA formed by the relations of IA that can be translated into conjunctions of relations in PA^c; ORD-Horn is the (unique) maximal tractable subclass of IA containing all the thirteen basic relations [Nebel and Bürckert, 1995]; finally, $\mathcal{A}_3$ [Golumbic and Shamir, 1993] is a class of relations formed by all the possible disjunctions of the following three interval relations: the basic Allen's relations *before* and *after*, and the IA relation $intersect$, which is defined as follows

$$intersect \;\equiv\; \mathbf{B} - \{\text{before, after}\}.$$

In the next sections, we will describe the main techniques for processing qualitative temporal constraints, including the algorithms concerning Table 8.1. We will consider the constraint network approach as well as some other graph-based approaches. Section 8.2 concerns the relations in the Point Algebra; Section 8.3 concerns the relations in tractable subclasses of the Interval Algebra; Section 8.4 concerns the relations of the full Interval Algebra; finally, Section 8.5 gives the conclusions, and mention some issues that at the time

Problem	PA^c/SIA^c	PA/SIA	ORD-Horn	IA	$\mathcal{A}_3$
Consistency	$O(n^2)$	$O(n^2)$	$O(n^3)$	exp	exp
Consistent scenario	$O(n^2)$	$O(n^2)$	$O(n^3)$	exp	exp
Minimal network	$O(n^3)$	$O(n^4)$	$O(n^5)$	exp	exp
One-to-one relations	$O(n^2)$	$O(n^2)$	$O(n^3)$	exp	exp
One-to-all relations	$O(n^2)$	$O(n^3)$	$O(n^4)$	exp	exp

Table 8.1: Time complexity of the known best reasoning algorithms for PA^c/SIA^c, PA/SIA, ORD-Horn, IA and $\mathcal{A}_3$, in terms of the number (n) of temporal variables involved."exp" means exponential.

of writing deserve further research.

8.2 Point Algebra Relations

The Point Algebra is a relation algebra [Tarski, 1941; Ladkin and Maddux, 1994; Hirsh, 1996] formed by three basic relations: $<$, $>$ and $=$. The operations unary converse (denoted by $\cdot^{\smile}$), binary intersection ($\cap$) and binary composition ($\circ$) are defined as follows:

$$\begin{array}{lrcl} \forall x,y: & xR^{\smile}y & \leftrightarrow & yRx \\ \forall x,y: & x(R\cap S)y & \leftrightarrow & xRy \wedge xSy \\ \forall x,y: & x(R\circ S)y & \leftrightarrow & \exists z:(xRz)\wedge(zSy). \end{array}$$

Table 8.2 defines the relation resulting by composing two relations in PA. Any set (conjunction) of interval constraints in SIA can be translated into a set (conjunction) of constraints in PA. For example, the interval constraints

`at(T1,S1)` {overlaps, contains, finished-by} `at(T2,S1)`
`at(T1,S1)` **B** – {equal, finishes, finished-by} `at(T2,S1)`

in the example given in the previous section can be translated into the following set of interval endpoint constraints

$\{$ `at(T1,S1)`$^-$ $<$ `at(T1,S1)`$^+$,
 `at(T2,S1)`$^-$ $<$ `at(T2,S1)`$^+$,
 `at(T1,S1)`$^-$ $<$ `at(T2,S1)`$^-$,
 `at(T2,S1)`$^-$ $<$ `at(T1,S1)`$^+$,
 `at(T1,S1)`$^+$ $\neq$ `at(T2,S1)`$^+$ $\}$,

where `at(T1,S1)`$^-$ denotes the starting time of `at(T1,S1)`, and `at(T1,S1)`$^+$ denotes the end time of `at(T1,S1)` (analogously for `at(T2,S1)`).

However, note that not all the constraints of the trains example can be translated into a set of constraints in PA. In particular, in addition to the $<$-constraints ordering the endpoints of the intervals involved, constraint (7) requires either a *disjunction* of constraints involving four interval endpoints,

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

Table 8.2: Composition table for PA relations

$$[\texttt{at(T1,S2)}^{+} < \texttt{at(T2,S2)}^{-}] \vee [\texttt{at(T2,S2)}^{+} < \texttt{at(T1,S2)}^{-}],$$

or two disjunctions involving three interval endpoints each [Gerevini and Schubert, 1994b]:

$$[\texttt{at(T1,S2)}^{-} < \texttt{at(T2,S2)}^{-}] \vee [\texttt{at(T2,S2)}^{+} < \texttt{at(T1,S2)}^{-}],$$
$$[\texttt{at(T2,S2)}^{-} < \texttt{at(T1,S2)}^{-}] \vee [\texttt{at(T1,S2)}^{+} < \texttt{at(T2,S2)}^{-}].$$

Some techniques for handling such "non-pointizable" interval constraints will be considered in Section 8.5.

8.2.1 Consistency Checking and Finding a Solution

The methods for finding a solution of a tractable set of qualitative constraints are typically based on first determining the consistency of the set, and then, if the set turns out to be consistent, using a technique for deriving a consistent scenario. A solution can be derived from a consistent scenario in linear time by simply assigning to each variable a number consistent with the order of the variables in the scenario.

Van Beek proposed a method for consistency checking and finding a consistent scenario for a set of constraints in PA consisting of the following steps [van Beek, 1992]:

1. Represent the input set of constraints as a labeled directed graph, where the vertices represent point variables, and each edge is labeled by the PA-relation of the constraint between the variables represented by the vertices connected by the edge.

2. Compute the strongly connected components (SCC) of the graph as described in [Cormen *et al.*, 1990], ignoring edges labeled "$\neq$". Then, check if any of the SCCs contains a pair of vertices connected by an edge with label "$<$" or "$\neq$". The input set of constraints is consistent if and only if such a SCC does *not* exist.

3. If the set of the input constraints is consistent, then collapse each SCC into an arbitrary vertex v within that component. Such a vertex represents an equivalent class of point variables (those corresponding to the vertices in the collapsed component).

4. Apply to the directed acyclic graph (DAG) obtained in the previous step an algorithm for topologically sorting its vertices [Cormen *et al.*, 1990], and use the resulting vertex ordering as a consistent scenario for the input set of constraints.

Both steps 1–2 (consistency checking) and steps 3–4 (finding a consistent scenario) can be computed in $O(n^2)$ time and space. For more details, the interested reader may see [Tarjan, 1972; van Beek, 1990; Cormen *et al.*, 1990; Gerevini and Schubert, 1994a].

8.2.2 Path Consistency and Minimal PA-networks

Path consistency is an important property for a set of constraints on which, as we will see, some reasoning algorithms rely. Enforcing path consistency to a given set of constraints corresponds to enforcing path consistency to its constraint network representation. This task requires deriving a (possibly) new *equivalent* network in which, for every 3-vertex subnetwork formed by vertices i, j, and k, the relation R_{ik} labeling the edge from i to k is stronger than the composition of the relations R_{ij} and R_{jk} labeling the edges from i to j, and from j to k, respectively. I.e., in a path-consistent network we have that

$$\forall i, j, k \;\; R_{ik} \subseteq R_{ij} \circ R_{jk},$$

where i, j and k are different vertices (variables) of the network. Two networks are equivalent when the represented variables admit the same set of consistent interpretations (solutions) in both the representations. Also, recall that the relations of PA (and of IA) form an algebra, and hence they are closed under the operation of composition, as well as under the operations of converse and intersection. For more details on the algebraic characterization of PA and IA, the interested reader may consult [Tarski, 1941; Ladkin and Maddux, 1994; Hirsh, 1996].

Several algorithms for enforcing path consistency are described in the literature on temporal constraint satisfaction (e.g., [Allen, 1983; van Beek, 1992; Vilain *et al.*, 1990; Ladkin and Maddux, 1994; Bessière, 1996]). Typically these algorithms require $O(n^3)$ time on a serial machine, where n is the number of temporal variables, while on parallel machines the complexity of iterative local path consistency algorithms lies asymptotically between n^2 and $n^2 log\ n$ [Ladkin and Maddux, 1988; Ladkin and Maddux, 1994]. Some path consistency algorithms require $O(n^2)$ space, while others require $O(n^3)$ space.

Figure 8.2.2 gives a path consistency algorithm which is a variant of the algorithm given by van Beek and Manchak in [van Beek and Manchak, 1996], which in turn is based on Allen's original algorithm for interval algebra relations [Allen, 1983]. Note that this is a general algorithm that can be applied not only to point relations in PA, but also to the interval relations in IA, as well as to other classes of binary qualitative relations, such as the spatial relations of the Region Connection Calculus RCC-8 [Randell *et al.*, 1992; Renz and Nebel, 1997; Gerevini and Renz, 1998].

The time complexity of this path consistency algorithm applied to a PA-network is $O(n^3)$, where n is the number of the temporal variables; the space complexity is $O(n^2)$. In order to derive from the input network C an equivalent path-consistent network, the algorithm checks whether the following conditions hold (steps 9 and 17)

(a) $R_{ik} \neq R_{ik} \cap R_{ij} \circ R_{jk}$,

(b) $R_{kj} \neq R_{kj} \cap R_{ki} \circ R_{ij}$.

If (a) holds, then R_{ik} is updated (strengthened), and the pair (i, k) is added to a list L to be processed in turn, provided that (i, k) is not already on L. Similarly, if (b) holds, then R_{kj}

Algorithm: PATH-CONSISTENCY(C)
Input: a $n \times n$ matrix C representing a network of constraints over n variables.
Output: either a path-consistent network equivalent to C or *false*.

```
1.  $L \leftarrow \{(i,j) \mid 1 \leq i < j \leq n\}$
2.  while ($L$ is not empty) do
3.      select and delete an item $(i,j)$ from $L$
4.      for $k \leftarrow 1$ to $n$, $k \neq i$ and $k \neq j$, do
5.          $t \leftarrow R_{ik} \cap R_{ij} \circ R_{jk}$
6.          if $t$ is the empty relation then
7.              return false
8.          else
9.              if $t \neq R_{ik}$ then
10.                 $R_{ik} \leftarrow t$
11.                 $R_{ki} \leftarrow$INVERSE$(t)$
12.                 $L \leftarrow L \cup \{(i,k)\}$
13.             $t \leftarrow R_{kj} \cap R_{ki} \circ R_{ij}$
14.             if $t$ is the empty relation then
15.                 return false
16.             else
17.                 if $t \neq R_{kj}$ then
18.                     $R_{kj} \leftarrow t$
19.                     $R_{jk} \leftarrow$INVERSE$(t)$
20.                     $L \leftarrow L \cup \{(k,j)\}$
21.  return $C$
```

Figure 8.6: A path consistency algorithm. R_{ij} indicates the relation between the i-th variable and the j-th variable which is stored as the entry $C[i,j]$ of C. The function INVERSE implements the unary operation inverse.

is updated (strengthened), and the pair (k,j) is added to L to be processed in turn, provided that (k,j) is not already on L. The algorithm iterates until no more relations (labels on the network edges) can be updated, or a relation is reduced to the empty relation. In the later case the algorithm returns *false* (the network is inconsistent); in the former case, when the algorithm terminates, it returns a path-consistent network equivalent to C.

Enforcing path consistency to a set of PA-constraints is a sound and complete method for determining the consistency of the set [Ladkin and Maddux, 1988; Ladkin and Maddux, 1994]: the set is consistent if and only if a path consistency algorithm applied to the set does not revise a relation to be the empty relation.

An interesting question is whether a path-consistent network of PA-constraints is also a minimal network. The answer is positive only for constraints in PA^c (i.e., the class containing all the relations of PA except "$\neq$"), while for constraints in PA this is not the case. This is because a path-consistent network of PA constraints can contain subnetworks that are

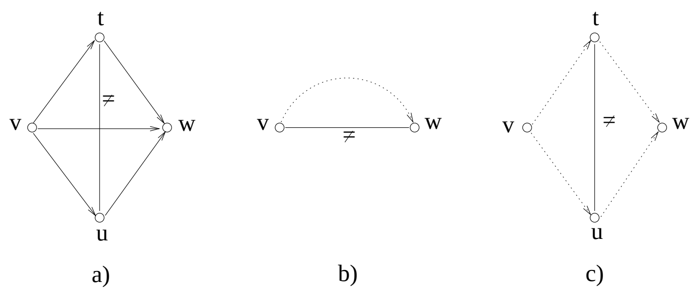


Figure 8.7: a) van Beek's forbidden graph; b) and c) the two kinds of implicit $<$ relation in a TL-graph. Edges with no labels are assumed to be labeled "$\leq$". Dotted arrows indicate paths, solid lines $\neq$-edges. In each of the graphs there is an implicit $<$ relation between v and w.

instances of a particular path-consistent PA-network that is not minimal. Such a subnetwork, called "forbidden subgraph", was identified by van Beek [van Beek, 1992] and it is depicted in Figure 8.7.a). This network is not minimal because the relation "$=$" between v and w is not feasible: there exists no solution for the set of constraints represented by the network in which the variables v and w are interpreted as the same time point.

Gerevini and Schubert proved that any path-consistent network of constraints in PA that contains no forbidden graphs is minimal [Gerevini and Schubert, 1995b].[*] Van Beek proposed an algorithm based on this property for computing the minimal network of a set of PA-constraints whose time complexity is $O(n^3 + m \cdot n^2)$, where m is number of input $\neq$-constraints [van Beek, 1992] (which is $O(n^2)$). This algorithm consists of two main steps:

- the first step computes a path-consistent network representing the input constraints, which can be accomplished in $O(n^3)$ time;
- the second step identifies and reduces all the forbidden graphs that are present in the path-consistent network. The time complexity of this step is $O(m \cdot n^2)$.

It is worth noting that the forbidden subgraph can be seen as a special case of the "metric forbidden graphs" identified by Gerevini and Cristani [Gerevini and Cristani, 1996] for the class of constraints STP$^{\neq}$, which subsumes PA and the class of metric constraints in Dechter, Meiri and Pearl's STP framework [Dechter *et al.*, 1991].

8.2.3 One-to-all Relations for PA

Figure 8.8 gives the algorithm proposed by van Beek and Cohen for computing one-to-all relations (OAC). The input of OAC is a matrix C representing a consistent network of constraints, and a vertex (a temporal variable) s of the network; the output is C with the constraints between s and every other variable (possibly) revised.

[*]This was first claimed (without a correct proof) by van Beek and Cohen in [van Beek and Cohen, 1990].

```
Algorithm: OAC($s, C$)
Input: a source vertex $s$ and a consistent constraints network stored in $C$
Output: $C$ revised to compute one-to-all relations for $s$

1. $L \leftarrow V - \{s\}$   ($V$ is the set of the vertices in the network)
2. while ($L$ is not empty) do
3.     select and delete a vertex $v$ from $L$ s.t. the cost of $R_{sv}$ is minimum
4.     for $t$ in $V$ do
5.         $l \leftarrow R_{st} \cap R_{sv} \circ R_{vt}$
6.         if $l \neq R_{st}$ then
7.             $R_{st} \leftarrow l$
8.             $L \leftarrow L \cup \{t\}$
9. return $C$
```

Figure 8.8: Van Beek and Cohen's algorithm for computing one-to-all relations.

```
Algorithm: OAC-2($s, C$)
Input: a source vertex $s$ and a consistent constraint network stored in $C$
Output: $C$ revised to compute one-to-all relations for $s$

1. for each relation $R_{si}$ ($1 \leq i \leq n$, $i \neq s$) do
2.     for each basic relation $r$ in $R_{si}$ do
3.         $t \leftarrow R_{si}$
4.         $R_{si} \leftarrow r$
5.         if $C$ is not consistent then
6.             $R_{si} = R_{si} - \{r\}$
7.         else $R_{si} = t$
8. return $C$
```

Figure 8.9: A cubic time algorithm for computing one-to-all relations for a (consistent) set of constraints in PA.

OAC takes $O(n^2)$ time and is an adaptation of Dijkstra's algorithm for computing the shortest path from a single source vertex to every other vertex [Cormen *et al.*, 1990]. The algorithm maintains a list of vertices that are processed following an order determined by the "costs" of the relative relation with the source vertex. In principle, these costs can be arbitrarily defined without affecting the worst-case time complexity of the algorithm. However, in practice the order in which the vertices are selected from the list can significantly affect the number of iterations performed by the algorithm. Van Beek and Cohen proposed a weighting scheme for computing the costs of the relations which halves the number of iterations of the algorithm, compared with the number of iterations determined by a random choice of the next vertex to be processed. (For more details the interested reader may see [van Beek and Cohen, 1990].)

Unfortunately, while the previous algorithm is complete for constraints in PA^c, it is not complete for constraints in PA [van Beek and Cohen, 1990]. It appears then that the (current) best method for computing one-to-all constraints for PA is based on independently checking the feasibility of each of the basic relations between the source variable (vertex) and every other variable (vertex). This algorithm is given in Figure 8.9. It is easy to see that the algorithm is correct and complete for the full PA, and that, since it performs $O(n)$ consistency checks (each of which requires $O(n^2)$ time – see Section 8.2.1), the time complexity of the algorithm is $O(n^3)$.

8.2.4 Efficient Graph-based Approaches

The techniques based on the constraint network representation are often elegant, simple to understand and easy to implement. On the other hand, their computational costs (both in time and space) can be inadequate for applications in which efficient temporal reasoning is an important issue.

Some alternative approaches based on graph algorithms have been proposed with the aim of addressing scalability, and supporting efficient reasoning for large data sets. Such techniques are especially suitable for managing "sparse" temporal data bases, i.e., sets of PA-constraints whose size is significantly lower than $(n^2 - n)/2$, where n is the number of the temporal variables involved.

Currently, the most effective of these graph-based methods are the approaches using *ranked temporally labeled graphs* [Gerevini and Schubert, 1995a; Delgrande *et al.*, 2001], *timegraphs* [Miller and Schubert, 1990; Gerevini and Schubert, 1995a], or *series-parallel graphs* [Delgrande and Gupta, 1996; Delgrande *et al.*, 2001]. In the following we will give a general overview of each of them. Another important related method is Ghallab and Mounir Alaoui's *indexed time table* [Ghallab and Mounir Alaoui, 1989]. However, this approach is incomplete for the full Point Algebra, because it can not detect some $<$-constraints that are implied via $\neq$-constraint [Gerevini and Schubert, 1995a]. Moreover, as claimed in [Delgrande *et al.*, 2001], the general idea underlying Ghallab and Mounir Alaoui's method seems subsumed by a variant of the approach based on series-parallel graphs.

In a graph-based approach, instead of precomputing all the constraints that are implicit in the set of input constraints (i.e., computing their minimal network representation), we build a graph representation with some particular data structures that support efficient deduction of constraints at *query time*. In two of the three approaches that we consider here, temporal information is partitioned into a set of specialized graph representations (either chains or series-parallel graphs [Valdes *et al.*, 1982]), that are connected by a metagraph data structure. Queries involving variables within the same specialized representation can be answered in constant time by using special purpose algorithms, while in the general case they require a search on the metagraph.

The time chain representation is particularly suitable for story comprehension and planning applications [Miller and Schubert, 1990; Gerevini and Schubert, 1995a]; the series-parallel graph representation can be used, for instance, to model the qualitative temporal evolution of process execution [Delgrande *et al.*, 2001].

Experimental results conducted using TimeGraph-II, a temporal reasoning system based on timegraphs [Gerevini *et al.*, 1995], show that building a timegraph can be much faster than computing the minimal network representation [Gerevini and Schubert, 1995a]. Moreover, for domains that exhibit structure, in terms of time chains or series-parallel graphs, querying the strongest entailed relations is very efficient [Gerevini and Schubert, 1995a; Delgrande *et al.*, 2001].

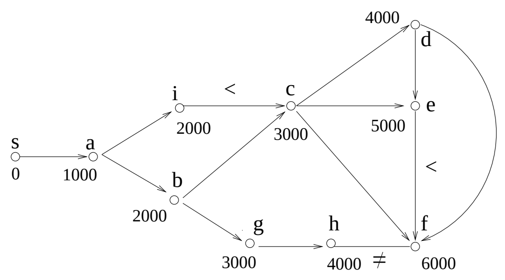


Figure 8.10: An example of TL-graph with ranks. Edges with no label are assumed to be labeled "$\leq$".

Ranked Temporally Labeled Graphs

A *temporally labeled graph* (*TL-graph*) is a graph with at least one vertex and a set of labeled edges, where each edge (v, l, w) connects a pair of distinct vertices v, w. The edges are either directed and labeled $\leq$ or $<$, or undirected and labeled $\neq$.

Every vertex of a TL-graph has at least one name attached to it, and if a vertex has more than one name, then they are alternative names for the same time point. The name sets of any two vertices are required to be disjoint. Figure 8.10 shows an example of TL-graph. A path in a TL-graph is called a $\leq$-path if each edge on the path has label $<$ or $\leq$. A $\leq$-path is called a $<$-path if at least one of the edges has label $<$.

A TL-graph $\mathcal{G}$ contains an *implicit* $<$ relation between two vertices v_1, v_2 when the strongest relation entailed by the set of constraints from which $\mathcal{G}$ has been built is $v_1 < v_2$ and there is no $<$-path from v_1 to v_2 in $\mathcal{G}$. Figures 8.7.b) and 8.7.c) show the two possible TL-graphs which give rise to an implicit $<$ relation. All TL-graphs with an implicit $<$ relation contain one of these graphs as subgraph.

An acyclic TL-graph without implicit $<$ relations is an *explicit* TL-graph. In order to make explicit a TL-graph containing implicit $<$ relations, we can add new edges with label $<$ [Gerevini and Schubert, 1995a]. For example, in Figure 8.7 we add the edge $(v, <, w)$ to the graph. An important property of an explicit TL-graph is that it entails $v \leq w$ if and only if there is a $\leq$-path from v to w; it entails $v < w$ if and only if there is a $<$-path from v to w, and it entails $v \neq w$ if and only if there is a $<$-path from v to w or from w to v, or there is an edge $(v, \neq, w)$.

Given a set S of c PA-constraints, clearly we can construct a TL-graph $\mathcal{G}$ representing S in $O(c)$ time. In order to check consistency of S and transform $\mathcal{G}$ into an equivalent acyclic TL-graph, we can use van Beek's method for PA (see Section 8.2.1). If the TL-graph is consistent, each SCC is collapsed into an arbitrary vertex v within that component. All the cross-component edges entering or leaving the component are transferred to v.[*] The edges within the component are eliminated and a supplementary set of alternative names for v is generated.

[*]When there is more than one edge from different collapsed vertices to the same vertex z that is not collapsed, the label on the edge from x to z is the intersection of the labels on these edges and the label on the current edge from x to z (if any). Similarly for multiple edges from the same vertex z to different collapsed vertices.

In order to query the strongest relation between two variables (vertices) of a TL-graph , there are two possibilities depending on whether (a) we preprocess all $\neq$-relations to make implicit $<$-relations explicit before querying, or (b) we identify implicit $<$-relations by handling $\neq$-relations at query time, using a more elaborated query algorithm [Delgrande *et al.*, 2001]. For generic TL-graphs, option (b) seems more appropriate than (a), because making implicit $<$ relations explicit can be significantly expensive (it requires $O(m \cdot n^2)$ time, where m is the number of input $\neq$-constraints and n the number of the temporal variables). However, as we will briefly discuss in the next section, for a TL-graph that is suitable for the timegraph representation, making $<$ relations explicit can be much faster.

Let $\mathcal{G}$ be a TL-graph with e edges representing a set S of c PA-constraints with no $\neq$-constraint ($e \leq m$). The strongest entailed relation R between two variables v_1 and v_2 in S can be determined in $O(e)$ time by two main steps: (1) check whether v_1 and v_2 are represented by the same vertex of $\mathcal{G}$ (in such a case R is "$=$"); if this is not the case, then (2) search for a $<$-path between the vertices representing v_1 and v_2 (or for a $\leq$-path, if there is no $<$-path). Such a search can be accomplished, for instance, by using the single-source-longest-paths algorithm given in [Cormen *et al.*, 1990].

When S contains $\neq$-constraints, and they pre-processed to make the TL-graph representing S explicit, the query algorithm is the same as above, except for the following addition: if the graph contains an $\neq$-edge between v_1 and v_2, then R is "$\neq$". Alternatively, if the TL-graph is not made explicit before querying, we can handle $\neq$-constraints as proposed in [Delgrande *et al.*, 2001]. During the search for a path from v_1 to v_2, we identify the set $V_{\leq} = \{w \mid v_1 \leq w,\ w \leq v_2\}$ by making the vertices of the graph according to whether they lie on a $\leq$-path from v_1 to v_2. (Similarly when searching for a path from v_2 to v_1.) If $x \neq y \in S$ and $x, y \in V_{\leq} \cup \{v_1, v_2\}$, then the TL-graph entails $v_1 < v_2$. The time complexity of the resulting query procedure is linear with respect to the number of the input constraints.

A *ranked* TL-graph is a simple but powerful extension of an acyclic TL-graph. In a ranked TL-graph, each vertex (time point) has a *rank* associated with it. The *rank* of a vertex v can be defined as the length of the longest $\leq$-paths to v from a source vertex s of the TL-graph representing the "universal start time", times a distance increment k [Ghallab and Mounir Alaoui, 1989; Gerevini and Schubert, 1995a]. s is a special vertex with no predecessor and whose successors are the vertices of the graph that have no other predecessor. The ranks for an acyclic TL-graph can be computed in $O(n + e)$ time using a slight adaptation of the DAG-longest-paths algorithm [Cormen *et al.*, 1990].

The use of the ranks can significantly speed up the search for a path from a vertex p to another vertex q: the search can be pruned whenever a vertex with a rank greater than or equal to the rank of q is reached. For instance, during the search of a path from a to g in the ranked TL-graph of Figure 8.10, when we reach vertex c, we can prune the search from this vertex, because the ranks if its successors are greater than the rank of g. Thus, it suffices to visit at most three nodes (i, c and b) before reaching g from a. The advantage of using ranks to prune the search at query time has been empirically demonstrated in [Delgrande *et al.*, 2001].

The experimental analysis in [Delgrande *et al.*, 2001] indicates that the use of a ranked TL-graph is the best approach when the graph is sparse and temporal information does not exhibit structure that can be "encapsulated" into specialized graph representations, like time chains or series-parallel graphs.

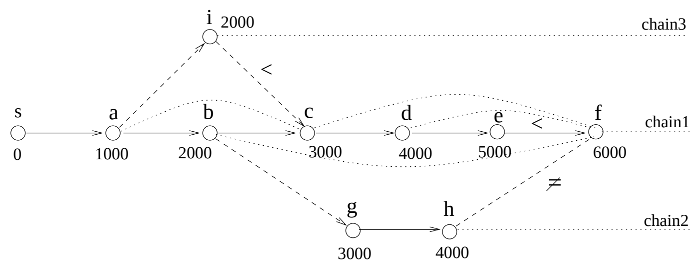


Figure 8.11: The timegraph of the TL-graph of Figure 8.10, with transitive edges and auxiliary edges omitted. Edges with no label are assumed to be labeled "$\leq$".

Timegraphs

A *timegraph* is an acyclic TL-graph partitioned into a set of *time chains*, such that each vertex is on one and only one time chain. A time chain is a $\leq$-path, plus possibly *transitive edges* connecting pairs of vertices on the $\leq$-path. Distinct chains of a timegraph can be connected by *cross-chain edges*. Vertices connected by cross-chain edges are called *metavertices*. Cross-chain edges and certain auxiliary edges connecting metavertices on the same chain are called *metaedges*. The metavertices and metaedges of a timegraph T form the *metagraph* of T.

Figure 8.11 shows the timegraph built from the TL-graph of Figure 8.10. All vertices except d and e are metavertices. The edges connecting vertices a to i, i to c, b to g, h with f, are metaedges. Dotted edges are special links called *nextgreaters* that are computed during the construction of the timegraph and that indicate for each vertex v the nearest descendant v' of v on the same chain as v such that the graph entails $v < v'$.

As in a ranked TL-graph, each vertex (time point) in a timegraph has rank associated with it. The main purpose of the ranks is to allow the computation of the strongest relation entailed by the timegraph between two vertices on the same chain in constant time. In fact, given two vertices v_1 and v_2 on the same chain such that the rank of v_2 is greater than the rank of v_1, if the rank of the nextgreater of v_1 is lower that the rank of v_2, then the timegraph entails $v_1 < v_2$, otherwise it entails $v_1 \leq v_2$. For example, the timegraph of Figure 8.11 entails $a < d$ because a and d are on the same chain, and the rank of the nextgreater of a is lower than the rank of d.

In general, the ranks in a timegraph are very useful to speed up path search both during the construction of the timegraph and at query time.

Given a set S of constraints in PA, in order to build a timegraph representation of S, we start from a TL-graph $\mathcal{G}$ representing S. The construction of the timegraph from $\mathcal{G}$ consists of four main steps: consistency checking, ranking of the graph (assigning to each vertex a rank), formation of the chains of the metagraph, and making explicit the implicit $<$ relations.

We have already described the first two steps in Section 8.2.4. The third step, the formation of the chains, consists of partitioning the TL-graph into a set of time chains ($\leq$-paths), deriving from this partition the metagraph and doing a first-pass computation of the nextgreater links. The first two subtasks take time linear in $n + e$, while the last may require an $O(\hat{e})$ metagraph search for each of the $\hat{n}$ metavertices, where $\hat{e}$ is the number of cross-chain edges in the timegraph.

The fourth step, making explicit all the implicit $<$ relations, can be the most expensive task in the construction of a timegraph. This is the same as in van Beek's approach for eliminating the forbidden graphs in a path-consistent network of PA-constraints (see Section 8.2.2). However, the data structures provided by a timegraph allow us to accomplish this task more efficiently in practice. A final operation is the revision of nextgreater links, to take account of any implicit $<$ relations made explicit by the fourth step.

Figures 8.7.b) and 8.7.c) show the two cases of implicit $<$ relations. The time complexity of the algorithm for handling the first case in the timegraph approach is $O(\hat{e}_{\neq} \cdot (\hat{e} + \hat{n}))$, where $\hat{e}_{\neq}$ is the number of cross-chain edges with label $\neq$.

In order to make implicit $<$ relations of the second kind (Figure 8.7.c) explicit, and removing redundant $\neq$-edges, a number of $\neq$ diamonds of the order of $e_{\neq} \cdot n^2$ may need to be identified in the worst case, where $e_{\neq}$ is the number of edges labeled "$\neq$" in the TL-graph. However, for timegraphs only a subset of these needs to be considered. In fact it is possible to limit the search to the *smallest* $\neq$ diamonds, i.e., the set of diamonds obtained by considering for each edge $(v, \neq, w)$ only the *nearest common descendants* of v and w ($NCD(v, w)$) and their *nearest common ancestors* ($NCA(v, w)$). This is a consequence of the fact that, once we have inserted a $<$ edge from each vertex in $NCA(v, w)$ to each vertex in $NCD(v, w)$, we will have explicit $<$-paths for all pairs of "diamond-connected" vertices. Overall, the worst-case time complexity of the fourth step is $O(\hat{e}_{\neq} \cdot (\hat{e} + \hat{n}))$.

Regarding querying the strongest entailed relations between two vertices (time points) p_1 and p_2 in a timegraph, there are four cases in which this can be accomplished in constant time. The first case is the one where p_1 and p_2 are alternative names of the same point. The second case is the one where the vertices v_1 and v_2 corresponding to p_1 and p_2 are on the same time chain. The third case is the one where v_1 and v_2 are not on the same chain and have the same rank, and there is no $\neq$ edge between them (the strongest entailed relation is ?). The fourth case is the one where p_1 and p_2 are connected by a $\neq$-edge (the strongest entailed relation is $\neq$).

In the remaining cases an explicit search of the graph needs to be performed. If there exists at least one $<$-path from v_1 to v_2, then the answer is $v_1 < v_2$. If there are only $\leq$-paths (but no $<$-paths) from v_1 to v_2, then the answer is $v_1 \leq v_2$. (Analogously for the paths from v_2 to v_1.) Such a graph search can be accomplished in $O(h + \hat{e} + \hat{n})$ time, where h is the constant corresponding to the time required by the four special cases.

Series-Parallel Graphs

In this section we describe Delgrande and Gupta's SPMG approach based on structuring temporal information into *series-parallel graphs* [Valdes *et al.*, 1982]. A series-parallel graph (SP-graph) is a DAG with one source s and one sink t, defined inductively as follows [Valdes *et al.*, 1982; Delgrande *et al.*, 2001]:

- *Base case*. A single edge (s, t) from s to t is a series-parallel graph with source s and sink t.
- *Inductive case*. Let G_1 and G_2 be series-parallel graphs with source and sink s_1, t_1 and s_2, t_2, respectively, such that the sets of vertices of G_1 and G_2 are disjoint. Then,

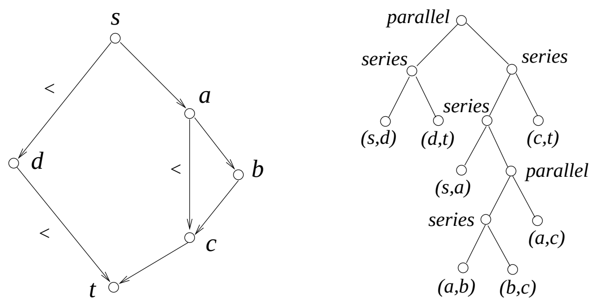


Figure 8.12: An example of an SP-graph with its decomposition tree. Edges with no label are assumed to be labeled "$\leq$".

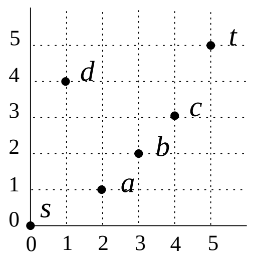
- *Series step*. The graph constructed by taking the disjoint union of G_1 and G_2 and identifying s_2 with t_1 is a series-parallel graph with source s_1 and sink t_2 constructed using a *series step*.

- *Parallel step*. The graph constructed by taking the disjoint union of G_1 and G_2 and identifying s_1 with s_2 (call this vertex s) and t_1 with t_2 (call this vertex t) is a series-parallel graph with source s and sink t constructed using a *parallel step*.

In SPMG, each edge of a SP-graph is labeled either $<$ or $\leq$, and represents either a $<$-constraint or a $\leq$-constraints, respectively ($\neq$-constraints are not explicitly represented in the graph). Moreover, in any SP-graph G with e edges and n vertices, we have $e \leq 2n$. Figure 8.12 gives an example of a simple SP-graph, together with its *decomposition tree*. Any SP-graph G can be decomposed into the decomposition tree of G, that is implicitly given by the definition of SP-graph. Internal nodes of the tree are labeled "series" or "parallel", while leaf nodes are labeled by edges of G. The decomposition tree of a SP-graph can be computed in linear time [Valdes *et al.*, 1982].

Using the decomposition tree of a SP-graph G, SPMG efficiently "compiles" G to derive some information supporting constant time queries on G. This processing requires $O(n)$ time, and consists of two main steps: computing a *planar embedding* of G, and computing the *functions* $\mathcal{S}$ *and* $\mathcal{A}$ for the vertices of G. A planar embedding of G is an assignment to each vertex v of a co-ordinate (x_v, y_v) on the integer plan such that, for any other vertex w, there is a path from v to w if and only if $x_v < x_w$ and $y_v < y_w$. Figure 8.13 shows an example of planar embedding. In this example, since the co-ordinates of a are both lower than the corresponding co-ordinates of t, we can derive that there is a $\leq$-path from a to t.

By using the planar embedding, we can answer queries of type $\leq$ in constant time, but we cannot answer queries of type $<$, because the planar embedding does not distinguish $<$-paths and $\leq$-paths in the SP-graph. The $\mathcal{S}$ and $\mathcal{A}$ functions of G have this purpose: given two vertices x, y such that in G there is a $\leq$-path from x to y, we have that G contains a $<$-path from x to y if and only if $\mathcal{A}(x) < \mathcal{S}(w)$. For example, since in Figure 8.13 $\mathcal{A}(a) < \mathcal{S}(t)$, the SP-graph under consideration has a $<$-path from a to t, entailing $a < t$.



$\mathcal{S}(s) = 0$	$\mathcal{A}(s) = 0$
$\mathcal{S}(t) = 2$	$\mathcal{A}(t) = 2$
$\mathcal{S}(a) = 0$	$\mathcal{A}(a) = 0$
$\mathcal{S}(b) = 0$	$\mathcal{A}(b) = 2$
$\mathcal{S}(c) = 1$	$\mathcal{A}(c) = 2$
$\mathcal{S}(d) = 1$	$\mathcal{A}(d) = 1$

Figure 8.13: A planar embedding and the $\mathcal{S}, \mathcal{A}$ values for the SP-graph of Figure 8.12.

For every vertex v of G, $\mathcal{S}(v)$ is defined as the maximum number of $<$-edges on any path from the source vertex of G to v, while $\mathcal{A}(v)$ is defined as follows. If v is the sink of G or there exists a vertex w such that G has an $<$-edge from v to w, then $\mathcal{A}(v) = \mathcal{S}(v)$; otherwise $\mathcal{A}(v)$ is the minimum value over values in the set $\{\mathcal{A}(w) \mid G$ contains a $\leq$-edge from v to $w\}$. Figure 8.13 gives an example of $\mathcal{S}$ and $\mathcal{A}$.

Given a (consistent) acyclic TL-graph G, SPMG constructs a *series-parallel metagraph* (SP-metagraph) G' for G as follows. First G is partitioned into a set of maximal series-parallel subgraphs, and each of these SP-graphs is collapsed into a single metaedge of G'. A metaedge from u to v represents a SP-graph with source u and vertex v, and its label is the intersection of the labels of all paths from u to l in G. Any $\neq$-edge in G connecting two vertices x, y in the same SP-subgraph may be replaced with a $<$-edge (if there is a path from x to y); otherwise the edge is a $\neq$-metaedge of the metagraph.

Then, each metaedge in the metagraph thus derived, is processed to compute a planar embedding for the corresponding SP-graph and its $\mathcal{A}$ and $\mathcal{S}$ functions. This allows SPMG to answer queries involving vertices "inside" the same metaedge in constant time. In order to answer queries between vertices of the metagraph, SPMG uses a path-search algorithm that, like in TL-graphs and timegraphs, can exploit ranks for pruning the search. The time complexity of such an algorithm is linear in the number of vertices and edges forming the metagraph. $\neq$-edges are handled at query time, as described for TL-graphs. To compute the strongest entailed relation between two vertices that are internal to two different metaedges, SPMG combines path-search on the metagraph and lookup inside the two SP-graphs associated with the metaedges. For more details on constructing and querying a SP-metagraph, the interested reader may see [Delgrande *et al.*, 2001].

An experimental analysis conducted in [Delgrande *et al.*, 2001] shows that SPMG is very efficient for sparse TL-graphs that are suited for being compiled into SP-metagraph representation. Furthermore, the performance of SPMG degrades gracefully for the randomly generated (without forcing any structure) data sets considered in the experimental analysis.

8.3 Tractable Interval Algebra Relations

IA is a relation algebra [Tarski, 1941; Ladkin and Maddux, 1994; Hirsh, 1996] in which the operators converse, intersection and compositions are defined as we have described for

PA in Section 8.2. In the context of IA, the main reasoning tasks are intractable. However, several tractable subclasses of IA have been identified. Among them, the most studied in terms of algorithm design are the convex simple interval algebra (SIA^c), the simple interval algebra (SIA) and the ORD-Horn algebra. In terms of the set of relations contained in these subalgebras of IA, we have that

$$\text{SIA}^c \subset \text{SIA} \subset \text{ORD-Horn}.$$

ORD-Horn is a maximal tractable subclass of IA formed by the relations in IA that can be translated into conjunctions of (at most binary) disjunctions of interval endpoint constraints in $\{\leq, =, \neq\}$ (or simply in $\{\leq, \neq\}$, given that any $=$-constraint can be expressed by two $\leq$-constraints), where at most one disjunct (PA-constraint) is of type "$=$" or "$\leq$". For example, the IA-constraint

$$I\ \mathbf{B} - \{starts\}\ J$$

can be translated into the following set of constraints between of I and J:

$$\{I^- < I^+,\ J^- < J^+,\ I^- \neq J^- \vee J^+ \leq I^+\},$$

where I^- and I^+ are the starting point and the end point of I, respectively (analogously for J^- and J^+).

Other maximal tractable subclasses of IA have been identified [Drakengren and Jonsson, 1997b; Krokhin *et al.*, 2003]. However, such classes are probably less interesting than ORD-Horn, because they do not contain all the basic relations of IA, and hence they cannot express definite information about the ordering of two temporal intervals. (For more details on these classes, the interested reader may see the chapter on computational complexity of temporal constraint problems in this book).

Finally, Golumbic and Shamir studied some interesting tractable subclasses of IA, which are restrictions of the set of relations forming the intractable class $\mathcal{A}_3$ [Golumbic and Shamir, 1993]. In particular, they show that, for some of these tractable subclasses, the problem of deciding consistency is equivalent to some well-known polynomial graph-theoretic problems; while for the other tractable subclasses they present new polynomial graph-based techniques. Golumbic and Shamir illustrate also some interesting applications of temporal reasoning involving their classes of relations.

In the rest of this section we will focus on the main techniques for processing constraints in SIA^c, SIA and ORD-Horn.

8.3.1 Consistency Checking and Finding a Solution

Consistency checking (finding a consistent scenario/solution) for a set of constraints in SIA^c and SIA can be easily reduced to consistency checking (finding a consistent scenario/solution) for an equivalent set of constraints in PA^c and PA respectively. Hence, these tasks can be performed by using the method described in Section 8.2.1, which requires $O(n^2)$ time.

Concerning ORD-Horn constraints, it has been proved that path consistency guarantees consistency [Nebel and Bürckert, 1995]: given a set Ω of constraints in ORD-Horn, the path

$\circ$	b	bi	d	di	o	oi	m	mi	s	si	f	fi
b	b	**B**	b d o m s	b	b	b d o m s	b	b d o m s	b	b	b d o m s	b
bi	**B**	bi	bi d oi mi fi	bi	bi d oi mi fi	bi	bi d oi mi fi	bi	bi d oi mi fi	bi	bi	bi
d	b	bi	d	**B**	b d o m s	bi d oi mi f	b	bi	d	bi d oi mi f	d	b d o m s
di	b di o m fi	bi di oi mi si	d di s si f fi o oi eq	di	di o fi	di oi si	di o fi	di oi fi	di o fi	di	di oi si	di
o	b	bi di oi mi si	d o s	b di o m fi	b o m	eq d di o oi s si f fi	b	di oi si	o	di oi fi	d o s	b o m
oi	b di o m fi	bi	d oi f	bi di oi mi si	eq d di o oi s si f fi	bi oi mi	di o fi	bi	d oi f	bi oi mi	oi	di oi si
m	b	bi di oi mi si	d o s	b	b	d o s	b	eq f fi	m	m	d o s	b
mi	b di o m fi	bi	d oi f	bi	d oi f	bi	eq s si	bi	d oi f	bi	mi	mi
s	b	bi	d	b di o m fi	b o m	d oi f	b	mi	s	eq s si	d	b o m
si	b di o m fi	bi	d oi f	di	di o fi	oi	di oi fi	mi	eq s si	si	oi	di
f	b	bi	d	bi di oi mi si	d o s	bi oi mi	m	bi	d	bi oi mi	f	eq f fi
fi	b	bi di oi mi si	d o s	di	o	di oi s	m	di oi s	o	di	eq f fi	fi

Table 8.3: Allen's composition table for IA (the basic relation "eq" is omitted). The composition of eq and any basic relation r is r.

consistency algorithm of Figure 8.2.2 is a complete procedure for deciding the consistency of Ω in cubic time.[*]

Other path consistency algorithms for processing IA-constraints have been proposed in the temporal reasoning literature. For a comparison of some of the most representative see [Bessière, 1996].

In practice, the efficiency of a path consistency algorithm applied to a set of constraints in IA may significantly depend on the time spent for constraint composition. Allen [Allen, 1983] originally proposed calculating the $2^{13} \times 2^{13}$ possible compositions of IA-relations dynamically, using a table storing the 13×13 compositions of the basic relations of IA (see Table 8.3). The composition of two arbitrary relations R_1 and R_2 in IA is the union of all the compositions $r_i \circ r_j$, such that r_i and r_j are basic relations in R_1 and R_2, respectively.

Other significantly improved methods have been proposed since then by Hogge [Hogge, 1987] and by Ladkin and Reinefeld [Ladkin and Reinefeld, 1997]. Ladkin and Reinefeld showed that their method of storing all the possible compositions in a table (requiring about 64 megabytes of memory) is much faster than any alternative previously proposed.

Van Beek and Manchak [van Beek and Manchak, 1996] studied other efficiency improvements for a path consistency algorithm applied to a set of IA-constraints obtained by using some techniques that reduce the number of composition operations to be performed, or by using particular heuristics for ordering the constraints to be processed (e.g., the pairs on the list L of the algorithm in Figure 8.2.2). Finally, another useful technique for improving path consistency processing in IA-networks is presented by Bessière in [Bessière, 1996].

Concerning the problem of finding a scenario/solution for a set of ORD-Horn constraints,

[*]We have introduced this algorithm in the context of PA, but the same algorithm can be used also for constraints in IA. Of course, for IA-constraints the algorithm uses a different composition table and a different INVERSE function.

two different approaches have been proposed. Given a path consistent set Ω involving variables $x_1, \ldots, x_n$, Ligozat proved that we can find a solution for Ω in the following way [Ligozat, 1996]: iteratively choose instantiations of x_i, for $1 \leq i \leq n$, in such a way that for each i, the interval assigned to x_i has the maximal number of endpoints distinct from the endpoints of the intervals assigned to x_k, $k = 1, \ldots i-1$, allowed by the constraints between x_i and x_k. Operatively, from this result we can derive the following simple method for finding a scenario: iteratively refine the relation R of a constraint xRy to a basic relation among those in R, preferring relations that impose a maximal number of different endpoints for x and y; each time a relation is refined, we enforce path-consistency to the resulting set of constraints. Each refinement transforms the network to a tighter equivalent network, and the method is guaranteed to be backtrack free.[*] Although Ligozat in his paper does not give a complexity analysis, it can be proved that his method takes $O(n^3)$ time [Bessière, 1997; Gerevini, 2003a].

The second method was proposed by Gerevini and Cristani [Gerevini and Cristani, 1997]. Their technique is based on deriving form a path-consistent set Ω of constraints in ORD-Horn a particular set Σ of constraints over PA involving the endpoints of the interval variables in Ω. From a consistent scenario (solution) for Σ we can then easily derive a consistent scenario (solution) for Ω. The time complexity of this method is $O(n^2)$, if the input set of constraints is known to be path-consistent, while in the general case it is $O(n^3)$, because before applying the method we need to process Ω with a path consistency algorithm.

8.3.2 Minimal Network and One-to-all Relations

A path-consistent network of constraints over SIA^c is minimal [Vilain and Kautz, 1986; Vilain *et al.*, 1990]. However, path consistency is not sufficient to ensure minimality when the constraints of the set are in SIA, and thus neither when they are in ORD-Horn, which is a superclass of SIA.[†]

Regarding ORD-Horn, Bessière, Isli and Ligozat proved that path consistency is sufficient to compute the minimal network representation for two subclasses of ORD-Horn, one of which covers more than 60% of the relations in ORD-Horn [Bessière *et al.*, 1996]. Actually, their results in [Bessière *et al.*, 1996] are stronger than this. They show that, for these subclasses of ORD-Horn, path consistency ensures *global consistency* [Dechter, 1992]. A globally consistent set of constraints implies that its constraint network is minimal and, furthermore, that a consistent scenario/solution can be found in a backtrack free manner [Dechter, 1992].

The minimal network representation for a set S of constraints in SIA can be computed in three main steps:

1. Translate S into an equivalent set S' of interval endpoint constraints over PA;
2. Compute the minimal network representation of S' by using the method described in Section 8.2.2;
3. Translate the resulting PA-constraints back into the corresponding interval constraints over SIA.

[*] A constraint network $\mathcal{N}$ is tighter than another equivalent network $\mathcal{N}'$ when every constraint represented by $\mathcal{N}$ is stronger than the corresponding constraint of $\mathcal{N}'$.

[†] An example of path-consistent set of constraints in IA that is not minimal is given in [Allen, 1983].

The time complexity of this procedure is dominated by step 2, which takes $O(n^3 + m \cdot n^2)$ time (see Section 8.2.2).

Unfortunately, this method cannot be applied to a set Ω of constraints over ORD-Horn, because the translation of Ω can include *disjunctions* of PA-constraints. It appears that at the time of writing the following simple method, based on solving a set of consistency checking instances, is the most efficient way for computing the minimal network of Ω (in terms of worst-case time complexity).

Minimal Network Representation for ORD-Horn. For each basic relation $r \in R$ involved in each constraint xRy of Ω, we refine xRy to xry and we check the consistency of the resulting set Ω'. r belongs to the relation of the constraint between x and y in the minimal network of Ω if and only if Ω' is consistent.

Since this method performs $O(n^2)$ consistency checks, it is clear that its worst-case time complexity is $O(n^5)$. It has been conjectured that this is the best that we can do for ORD-Horn [Nebel and Bürckert, 1995] (in terms of worst-case complexity), and it appears that at the time of writing no proof exists for this claim yet.

Concerning the problem of computing one-to-all relations, the algorithm OAC that we have presented in the context of PA (see Figure 8.8) is complete for SIA^c, but it is incomplete for SIA.

When the input constraints belong to SIA, or to a larger class, the problem of computing one-to-all relations can be solved by using the algorithm OAC-2 (see Figure 8.9). It is easy to see that, since consistency checking for a set of constraints in SIA can be accomplished in $O(n^2)$ time, OAC-2 applied to a set of SIA-constraints requires $O(n^3)$ time. Similarly, OAC-2 applied to a set of ORD-Horn constraints requires $O(n^4)$ time, because consistency checking requires $O(n^3)$ time.

Finally, in Section 8.4.2 we will consider an extension of the timegraph approach called *disjunctive timegraphs*. This extension adds a great deal of expressiveness power, including the ability to represent constraints in ORD-Horn. A disjunctive timegraph representing a set of constraints in ORD-Horn can be polynomially processed for solving the problems of consistency checking, finding a scenario/solution and computing one-to-one relations.

8.4 Intractable Interval Algebra Relations

In this section we present some techniques for processing intractable classes of IA-relations, i.e., classes for which the main reasoning problems are NP-hard, and hence that cannot be solved in polynomial time (assuming P $\neq$ NP).

The ORD-Horn class, though computationally attractive, is not practically adequate for all AI applications because it does not contain disjointness relations such as *before or after*, which are important in planning and scheduling. Figure 8.14 gives a list of the disjointness relations in IA, together with their translation in terms of PA-constraints between interval endpoints.

For example, these relations can be useful to express the constraint that some planned actions cannot be scheduled concurrently, because they contend for the same resources (agents, vehicles, tools, pathways, and so on). In the context of the trains example given in Section 8.1, the constraint

I {before,after} J	$\Leftrightarrow$	$I^+ < J^- \lor J^+ < I^-$
I {before,met-by} J	$\Leftrightarrow$	$I^+ < J^- \lor J^+ = I^-$
I {meets,after} J	$\Leftrightarrow$	$I^+ = J^- \lor J^+ < I^-$
I {meets,met-by} J	$\Leftrightarrow$	$I^+ = J^- \lor J^+ = I^-$
I {before,after,meets} J	$\Leftrightarrow$	$I^+ \leq J^- \lor J^+ < I^-$
I {before,after,met-by} J	$\Leftrightarrow$	$I^+ < J^- \lor J^+ \leq I^-$
I {before,meets,met-by} J	$\Leftrightarrow$	$I^+ \leq J^- \lor J^+ = I^-$
I {meets,after,met-by} J	$\Leftrightarrow$	$I^+ = J^- \lor J^+ \leq I^-$
I {before,after,meets,met-by} J	$\Leftrightarrow$	$I^+ \leq J^- \lor J^+ \leq I^-$

Figure 8.14: The nine disjointness relations of IA and the corresponding translation into disjunctions of PA-constraints between interval endpoints. (The complete translation of each interval relation between I and J into a set of point relations also contains the endpoint constraints $I^- < I^+$ and $J^- < J^+$.)

$$\texttt{at}(\texttt{T2}, \texttt{S2})\ \{\text{before, after}\}\ \texttt{at}(\texttt{T1}, \texttt{S2}),$$

which is used to express the information that two trains cannot stop at station S2 at the same time, cannot be expressed using only ORD-Horn relations. Moreover, as indicated in [Gerevini and Schubert, 1994b], reasoning about disjoint actions or events is important also in natural language understanding. Finally, Golumbic and Shamir [Golumbic and Shamir, 1993] discuss an application of reasoning about interval relations to a problem in molecular biology, where disjointness relations are used to express that some pairs of segments of DNA are disjoint [Benzen, 1959].

Unfortunately, adding any of the disjunctive relation of Figure 8.14 to ORD-Horn (as well as to SIA, SIA^c and to the simple class formed by only the thirteen relations of IA) leads to intractability.*

In general, the problem of deciding the satisfiability of a set of constraints in IA (called ISAT for IA in [Golumbic and Shamir, 1993]) is NP-complete [Vilain and Kautz, 1986; Vilain *et al.*, 1990]. The hard to solve instances of ISAT for IA appear around a *phase transition* concerning the probability of satisfiability that was identified and investigated by Ladkin and Reinefeld [Ladkin and Reinefeld, 1992] and by Nebel [Nebel, 1997].

A phase transition is characterized by some critical values of certain order parameters of the problem space [Cheeseman *et al.*, 1991]. Specifically, Ladkin and Reinefeld observed a phase transition of ISAT for IA in the range $6 \leq q \times n \leq 15$ for $q \geq 0.5$, where q is the ratio of non-universal constraints (i.e., those different from the disjunction of all the thirteen basic relations), and n is the number of the interval variables involved in the constraints.

Nebel characterized the phase transition in terms of the average degree (d) of the constraint network representing the input set of constraints. For example, when the average number of basic relations forming a constraint (i.e., its "label size") is 6.5, the phase transition shown in Figure 8.4 is centered in $d = 9.5$. (For more details, the interested reader may

*It should be noted that there exist some tractable classes containing {before, after} [Krokhin *et al.*, 2003; Golumbic and Shamir, 1993]. However, these classes do not contain all the basic relations of IA, and hence they are limited in terms of the definite temporal information that can be expressed. This may significantly affect the applicability of these classes.

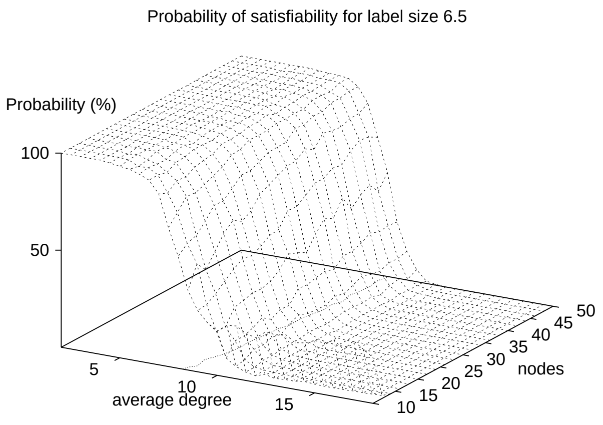


Figure 8.15: Nebel's Phase transition of ISAT for IA when the label size is 6.5.

see [Nebel, 1997; Ladkin and Reinefeld, 1992]).

8.4.1 Backtracking and Path Consistency

Typically, the algorithms for processing intractable classes of relations in IA are based on search methods that use backtracking [Bitner and Reingold, 1975; Shanahan and Southwick, 1989]. Ladkin and Reinefeld [Ladkin and Reinefeld, 1992] proposed a method for determining the consistency of a set of constraints in IA that is based on chronological backtracking, and that uses path consistency algorithms as a forward checking and pruning technique. At the time of writing their algorithm, which has also been investigated by van Beek and Manchak [van Beek and Manchak, 1996] and by Nebel [Nebel, 1997], appears to be the known fastest (complete) method for handling constraints in the full IA, using the constraint network representation.*

Figure 8.16 gives Ladkin and Reinefeld's algorithm as formulated by Nebel in [Nebel, 1997]. The input set of constraints is represented by a $n \times n$ matrix, where each entry C_{ij} contains the IA-relation between the interval variables i and j $(1 \leq i, j \leq n)$. Split is a tractable subset of IA (e.g., ORD-Horn). The larger is Split, the lower is the average branching factor of the search.

Nebel proved that the algorithm is complete when Split is SIA^c, SIA, or ORD-Horn [Nebel, 1997]. He also analyzed experimentally the backtracking scheme of Figure 8.16 when Split is chosen to be one of the above tractable sets, showing that the use of ORD-Horn leads to better performance on average. However, it turned out that there are other algorithmic features that affect the performance of the backtracking algorithm more significantly than the choice of which kind of Split to use. These features regard the heuristics for ordering the constraints to be processed, and the kind of path consistency algorithm used at step 1 (this can be either based on a weighted queue scheme, such as the algorithm of

*Another recent powerful approach is the one proposed in [Thornton *et al.*, 2002; Thorthon *et al.*, 2004], that uses local search techniques for fast consistency checking. This method can outperform backtracking-based methods. However, as any local search approach, it is incomplete.

Algorithm: IA-CONSISTENCY(C)
Input: A matrix C representing a set Θ of constraints in IA
Output: *true* if Θ is satisfiable, *false* otherwise

```
1.  $C \leftarrow$ PATH-CONSISTENCY($C$)
2.  if $C = false$ then
3.      return false
4.  else choose an unprocessed relation $C_{ij}$ and
5.      split $C_{ij}$ into $R_1, \ldots, R_k$ s.t. all $R_l \in \mathsf{Split}$  $(1 \leq l \leq k)$
6.      if no relation can be split then
7.          return true
8.      for $l \leftarrow 1$ to $k$ do
9.          $C_{ij} \leftarrow R_l$
10.         if IA-CONSISTENCY($C$) then
11.             return true
12.     return false
```

Figure 8.16: Ladkin and Reinefeld's backtracking algorithm for consistency checking of IA-constraints [Ladkin and Reinefeld, 1992; Nebel, 1997].

Figure 8.2.2, or an algorithm based on an iterative scheme which uses no queue, such as the algorithm PC-1 [Montanari, 1974; Mackworth, 1977]).[*]

As shown by Nebel, from these design choices and the kind of tractable set used as Split, we can derive different search strategies that on some problem instances have complementary performance. These strategies can be orthogonally combined to obtain a method that can solve (within a certain time limit) more instances than those solvable using the single strategies. (For more details, the interested reader may see [Nebel, 1997]).

Concerning the problems of computing the minimal network representation, one-to-all relations and one-to-one relations, at the time of writing no specialized algorithm is known. However, each of these problems can be easily reduced to a set of instances of the consistency checking problem, which can be solved by using the backtracking algorithm illustrated above. For example, in order to determine the one-to-one relation between two intervals I and J, we can check the feasibility of each basic relation r contained in the stipulated relation R between I and J in the following way: first we replace R with r, and then we run IA-CONSISTENCY to check the consistency of the modified network.

The problem of computing a consistent scenario (solution) can be reduced to the problem of consistency checking. In fact, a consistent scenario (solution) for an input set of constraints exists only if the set is consistent and, in such a case, IA-CONSISTENCY has the "side effect" of reducing it to a set of *tractable* constraints (depending on the value of Split, these constraints can be, for example, in SIA^c, SIA, or ORD-Horn). A consistent scenario (solution) for this tractable set is also a scenario (solution) for the input set of constraints, and can it be determined by using the techniques described in Section 8.3.1.

[*]PC-2 [Mackworth, 1977] is another important path-consistency algorithm that has been used in the context of temporal reasoning (e.g., [van Beek and Cohen, 1990]). A disadvantage of PC-2 is that it requires $O(n^3)$ space, while the other algorithms that we have mentioned requires $O(n^2)$ space, where n is the number of the variables in the input set of (qualitative) temporal constraints.

8.4.2 Disjunctive Timegraphs

We now consider an alternative method for representing and processing intractable relations in IA, which is based on an extension of the timegraph approach illustrated in Section 8.2.4.

A *disjunctive timegraph* ($\mathcal{D}$-timegraph) is a pair $\langle T, D\rangle$, where T is a timegraph and D a set of constraints in PA (PA-disjunctions) involving only point-variables in T (see Figure 8.17). Considering our trains example in Section 8.1, each of the temporal constraints 1–7 can be expressed using a $\mathcal{D}$-timegraph. More in general, the disjunctions of a $\mathcal{D}$-timegraph add a great deal of expressive power to a timegraph, including the ability to represent relations in ORD-Horn, disjointness of temporal intervals (see Figure 8.14). Moreover, A $\mathcal{D}$-timegraph can represent other relations not belonging to IA or PA, such as Vilain's point-interval relations [Vilain, 1982],* point-interval disjointness relations [Gerevini and Schubert, 1994b] and some 3-interval and 4-interval relations [Gerevini and Schubert, 1995a] such as

$$I\ \{\text{before}\}\ J\ \vee\ K\ \{\text{before}\}\ H.^{\dagger}$$

The current algorithms for processing the disjunctions of a $\mathcal{D}$-timegraph are specialized for binary disjunctions, and hence not every relation in IA is representable (because there are some relations that require ternary disjunctions). In principle, the techniques presented in [Gerevini and Schubert, 1995a] can be extended to deal with arbitrary disjunctions.

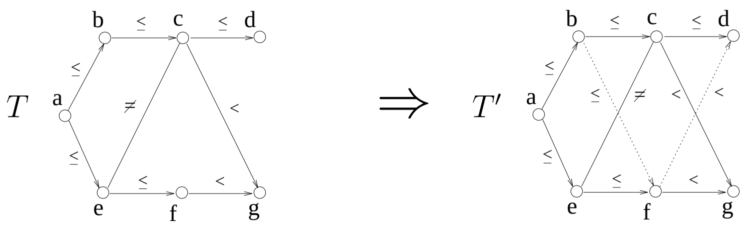
A $\mathcal{D}$-timegraph $\langle T, D\rangle$ is *consistent* if it is possible to select one of the disjuncts for each PA-disjunction in D in such a way that the resulting collection of selected PA-constraints can be consistently added to T. This set of selected disjuncts is called an *instantiation* of D in T, and the task of finding such a set is called *deciding* D relative to T.

Once we have an instantiation of D, we can easily solve the problem of finding a consistent scenario by adding the instantiation to T and using a topological sort algorithm [Cormen *et al.*, 1990]. In order to check whether a relation R between two time points x and y is entailed by a $\mathcal{D}$-timegraph $\langle T, D\rangle$, we can add the constraint $x\overline{R}y$ to T (where $\overline{R}$ is the negation of R), obtaining a new timegraph T', and then check if (a) T' is consistent, and (b) D can be decided relative to T' (if T' is consistent). The original $\mathcal{D}$-timegraph entails xRy just in case one of (a), (b) does *not* hold. This gives us a method for computing one-to-one relations, one-to-all relations, as well as the minimal network representation using the $\mathcal{D}$-timegraph approach.

Deciding a set of binary PA-disjunctions is an NP-complete problem [Gerevini and Schubert, 1994b]. However, in practice this task can be efficiently accomplished by using a method described in [Gerevini and Schubert, 1994a; Gerevini and Schubert, 1995a]. Given a disjunctive timegraph $\langle T, D\rangle$, the method for deciding D relative to T consists of two phases:

*Vilain's class of point-interval relations is formed by 2^5 relations obtained by considering all the possible disjunctions of five basic relations between a point and an interval.

†Point-interval disjointness can be used to state that a certain time point (perhaps an instantaneous action, or the beginning or end of an action) must not be within a certain interval (another action, or the interval between two actions). This kind of constraints is fundamental, for example, in nonlinear planning (e.g., [Chapman, 1987; Pednault, 1986b; Sacerdoti, 1975; Tate, 1977; Weld, 1994; Yang, 1997]), where an earlier action may serve to achieve the preconditions of a later one, and no further action should be inserted between them which would subvert those preconditions.



$$D : \begin{cases} D(1) : f < a \vee f < d \\ D(2) : b < f \vee a < g \\ D(3) : g \leq a \vee b \leq f \\ D(4) : d < g \vee b < d \\ D(5) : b \neq e \vee b < d \end{cases} \Rightarrow \quad D' : \; D(5) : b \neq e \vee b < d$$

Figure 8.17: An example of $\mathcal{D}$-timegraph $\langle T, D\rangle$ from [Gerevini and Schubert, 1995a]. The $\mathcal{D}$-timegraph $\langle T', D'\rangle$ is equivalent to $\langle T, D\rangle$, and is obtained by applying the pruning rules to the disjunctions in D using T.

- a preprocessing phase, which prunes the search space by reducing D to a subset D' of D, producing a timegraph T' such that D has an instantiation in T if and only if D' has an instantiation in T';
- a search phase, which finds an instantiation of D' in T' (if it exists) by using backtracking.

Preprocessing uses a set of efficient *pruning rules* exploiting the information provided by the timegraph to reduce the set of the disjunctions to a logically equivalent subset. For example, the “T-derivability” rule says, informally, that if the timegraph entails one of the disjuncts of a certain disjunction, then such a disjunction can be removed without loss of information; the “T-resolution” rule says that if the timegraph entails the negation of one of the two disjuncts of a disjunction, then this disjunction can be reduced to the other disjunct (called “T-resolvent”), which can then be added to the timegraph (provided that the timegraph does not entail also the negation of the second disjunct); finally the “T-tautology” rule can be used to detect whether a disjunction can be eliminated because it is a tautology with respect to the information entailed by the timegraph. (For more details on these and other rules, the interested reader may see [Gerevini and Schubert, 1995a].)

Various strategies are possible for preprocessing the set of disjunctions using the pruning rules. The simplest strategy is the one in which the rules are applied to each disjunction once, and the set of T-resolvents generated is added to the timegraph at the end of the process. For example, the $\mathcal{D}$-timegraph $\langle T', D'\rangle$ of Figure 8.17 can be obtained from $\langle T, D\rangle$ by following this simple strategy.[*]

A more complete strategy, though more computationally expensive, is to add the T-resolvents to the graph as soon as they are generated, and to iterate the application of the

[*] D(1) and D(3) are eliminated by T-resolution, D(2) by T-derivability and D(4) by T-tautology.

rules till no further disjunction can be eliminated. This strategy is still polynomial, and it is complete for the class of PA-disjunctions translating a set of interval relations in ORD-Horn [Gerevini and Schubert, 1995a]. In general, the choice of the preprocessing strategy depends on how much effort one wants to dedicate to the preprocessing step and how much to the search step.

Once the initial set of disjunctions has been processed by applying the pruning rules, if this processing has not been sufficient to decide consistency, then the search for an instantiation of the remaining disjunctions is activated. Gerevini and Schubert proposed a search algorithm specialized for binary disjunctions of strict inequalities, which can express the practically important relation *before or after*, as well as point-interval disjointness (i.e., exclusion of a point form an interval). The algorithm is based on a "partially selective backtracking" technique, which combines chronological backtracking and a form of selective backtracking [Gerevini and Schubert, 1995a; Bruynooghe, 1981; Shanahan and Southwick, 1989].

The experimental results presented in [Gerevini and Schubert, 1995a] show that the $\mathcal{D}$-timegraph approach is very efficient especially when the timegraph is not very sparse (i.e., "enough" non-disjunctive temporal information is available), and the number of disjunctions is relatively small with respect to the number of input PA-constraints represented in the timegraph. For more difficult cases (sparse timegraphs with few PA-constraints and numerous PA-disjunctions) a "forward propagation" technique can be included into the basic search algorithm. Such a technique can dramatically reduce the number of backtracks.

8.5 Concluding Remarks

The ability to efficiently represent and process temporal information is an important issue in AI, as well as in other discipline of computer science (e.g., [Song and Cohen, 1991; Lascarides and Oberlander, 1993; Hwang and Schubert, 1994; Snodgrass, 1990; Kline, 1993; Kline, 1993; Özsoyŏglu and Snodgrass, 1995; Orgun, 1996; Golumbic and Shamir, 1993]). In this chapter we have surveyed a collection of techniques for processing qualitative temporal constraints, focusing on fundamental reasoning tasks, such as consistency checking, finding a solution (or consistent scenario), and deducing (or querying) new constraints from those that are explicitly given.

We believe that this style of temporal reasoning is relatively mature and has much to offer to the development of practical applications. However, at the time of writing there are still some important aspects that deserve further research. These include the following:

- The design and experimental evaluation of efficient methods for *incremental* qualitative temporal reasoning, both in the context of the general constraint network approach and of specialized graph-based representations like timegraphs or series-parallel graphs. In fact, in many applications we are interested in *maintaining* certain properties (e.g., consistency, the minimal network representation or the time chain partition of a TL-graph), rather then recomputing them from scratch each time a new constraint is asserted, or an existing constraint is retracted. Some studies in this direction for metric constraints are presented in [Bell and Tate, 1985; Cervoni *et al.*, 1994; Gerevini *et al.*, 1996], while other more recent studies focusing on qualitative constraints are described in [Delgrande and Gupta, 2002; Gerevini, 2003a; Gerevini, 2003b].

- The study of alternative algorithms for dealing with intractable classes of temporal constraints, such as *anytime* algorithms (e.g., [Boddy and Dean, 1994; Hansen and Zilberstein, 1996; Zilberstein, 1996]), and algorithms based on local search techniques. As we have already mentioned, an interesting example of such techniques for qualitative temporal reasoning is given in [Thorthon *et al.*, 2004].

- The study of new methods for representing and managing qualitative relations involving non-convex intervals, which, for example, can be useful in the representation of periodic events (e.g., [Leban *et al.*, 1986; Poesio and Brachman, 1991]).[*]

- The design of new efficient representations and algorithms for managing combined qualitative and metric temporal information. In particular, the integration of metric constraints involving deadlines, durations and absolute times into the timegraph representation is a promising research direction for addressing scalability in temporal reasoning with qualitative and metric information.[†]

- The study of new algorithms for handling Interval Algebra relations extended with qualitative relations about the relative duration of the involved intervals (e.g, I overlaps J and the duration of I is shorter that the duration of J). An interesting calculus for dealing with this type of temporal constraints has been proposed and studied in [Pujari *et al.*, 1999; Kumari and Pujari, 2002; Balbiani *et al.*, 2003]. Hoewever, it appears that this calculus has not yet been fully investigated from an algorithmic point of view.

- The integration of qualitative temporal reasoning and other types of constraint-based reasoning, such as qualitative spatial reasoning, into a uniform framework. For instance, the Spatio-Temporal Constraint Calculus is a recent approach for spatio-temporal reasoning integrating Allen's Interval Algebra and the Region Connection Calculus RCC-8 [Randell *et al.*, 1992; Bennett *et al.*, 2002a; Gerevini and Nebel, 2002]. The development of efficient reasoning algorithms for such a combined calculus is an important direction for future research.

[*] In this chapter we have not treated this type of qualitative constraints. The interested reader can see other chapters in this book.

[†] These metric constraints were handled in the original implementation of timegraphs [Schubert *et al.*, 1987; Miller and Schubert, 1990], but $\neq$ and PA-disjunctions were not handled, and also $<$ and $\leq$ relations entailed via metric relations were not extracted in a deductively complete way.