Reasoning about Qualitative Temporal Information

Peter van Beek
Department of Computer Science
University of Waterloo
Waterloo, Ontario
CANADA N2L 3G1

Abstract

Interval and point algebras have been proposed for representing qualitative temporal information about the relationships between pairs of intervals and pairs of points, respectively. In this paper, we address two related reasoning tasks that arise in these algebras: Given (possibly indefinite) knowledge of the relationships between some intervals or points, (1) find one or more scenarios that are consistent with the information provided, and (2) find all the feasible relations between every pair of intervals or points. Solutions to these problems have applications in natural language processing, planning, and a knowledge representation language. We define computationally efficient procedures for solving these tasks for the point algebra and for a corresponding subset of the interval algebra. Our algorithms are marked improvements over the previously known algorithms. We also show how the results for the point algebra help us to design a backtracking algorithm for the full interval algebra that is useful in practice.

Topic: Qualitative representations of time.

Declaration of multiple submission: This paper was also submitted to AAAI-90.

1. Introduction

Qualitative temporal information is information such as "The Cuban Missile crisis took place during Kennedy's presidency," where only the ordering of the end points of the two events is specified. Allen [2] and Vilain & Kautz [29] have proposed algebras for representing such qualitative information about the relationships between pairs of intervals and pairs of points, respectively.

In this paper, we address two fundamental temporal reasoning problems that arise in these algebras: Given (possibly indefinite) knowledge of the relationships between some intervals or points,

1. find one or more scenarios that are consistent with the information provided.
2. find all the feasible relations between every pair of intervals or points†.

Specific applications of solutions to these problems include natural language processing (Allen [3], Song & Cohen [23]), planning (Allen & Koomen [5], Hogge [10]), plan recognition (Kautz [11]), and a knowledge representation language (Koubarakis et al. [13]), and history-based qualitative simulation (Weld & de Kleer [31]). As well, the techniques developed here could be part of a specialist in a general temporal reasoning system (see [20]) that would have other specialists for other kinds of temporal information such as quantitative information about the distances between intervals or points (Dechter et al. [8], Dean [6]), or combinations of qualitative and quantitative information (Allen & Kautz [4], Ladkin [14], Malik & Binford [19]).

The main results of the paper are as follows. For the point algebra and for a corresponding subset of the interval algebra, we define computationally efficient procedures for solving both of these tasks. Our algorithms are marked improvements over the previously known algorithms. In particular, for finding one consistent scenario we develop an $O(n^2)$ time algorithm (as opposed to the previously known $O(n^3)$ algorithm [15]) and for finding all the feasible relations we develop an algorithm that takes, under certain assumptions, $O(n^3)$ time (as opposed to the previously known $O(n^4)$ algorithm [26]), where n is the number of intervals or points.

For the full interval algebra, Vilain & Kautz [29, 30] show that both of these tasks are NP-Complete. This strongly suggests that no polynomial time algorithm exists. We show how the results for the point algebra help us to design a backtracking algorithm for finding one consistent scenario that, while exponential in the worst case, is shown to be useful in practice. A similar backtracking approach is given for finding all the feasible relations. The results here are less encouraging in practice and we conclude that a better approach in this case is to, if possible, accept approximate solutions to the problem (Allen [2], van Beek [26]).

---

† The terminology is from [8]. Other names for problem (1) include consistent singleton labeling [26] and a satisfying assignment of values to the variables [15]. Other names for problem (2) include deductive closure [30], minimal labeling [26] and, as it arises as a general constraint satisfaction problem, minimal network [21].

2. Background, Definitions, and An Example

In this section we review Allen's interval algebra and Vilain & Kautz's point algebra. We end with an example from the interval algebra of the two reasoning problems we want to solve.

Definition 1. Interval algebra, **IA** (Allen [2]). There are thirteen basic relations (including converses) that can hold between two intervals.

relation	symbol	converse	meaning
x before y	b	bi	xxx yyy
x meets y	m	mi	xxxyyy
x overlaps y	o	oi	xxx yyy
x during y	d	di	xxx yyyyy
x starts y	s	si	xxx yyyyy
x finishes y	f	fi	xxx yyyyy
x equal y	eq	eq	xxx yyy

We want to be able to represent indefinite or uncertain information so we allow the relationship between two intervals to be a disjunction of the basic relations. We use sets to list the disjunctions. Somewhat more formally, let I be the set of all basic relations, {b, bi, m, mi, o, oi, d, di, s, si, f, fi, eq}. **IA** is the algebraic structure with underlying set 2^I, the power set of I, unary operator converse, and binary operators intersection and composition (see [2] for the definition of the operators). Ladkin & Maddux [16] show that **IA** satisfies the definition of a relation algebra.

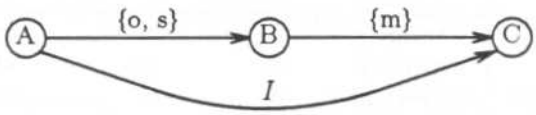
Definition 2. Point algebra, **PA** (Vilain & Kautz [29]). There are three basic relations that can hold between two points $<$, $=$, and $>$. As in the interval algebra, we want to be able to represent indefinite information so we allow the relationship between two points to be a disjunction of the basic relations. **PA** is the algebraic structure with underlying set $\{\emptyset, <, \leq, =, >, \geq, \neq, ?\}$, unary operator converse, and binary operators intersection and composition (see [29] for the definition of the operators). Note that $\leq$, for example, is an abbreviation of $\{<, =\}$, $\emptyset$ is the inconsistent constraint, and $?$ means there is no constraint between two points, $\{<, =, >\}$. Ladkin & Maddux [16] also show that **PA** satisfies the definition of a relation algebra.

Vilain & Kautz show that a subset of the interval algebra can be translated into their point algebra. We denote as **SIA** the subset of the underlying set of the interval algebra that can be translated into relations between the endpoints of the intervals using the underlying set of **PA** (see [15, 28] for an enumeration of **SIA**).

We will use a graphical notation where the vertices represent intervals or points and the directed edges are labeled with elements from the appropriate algebra representing the disjunction of possible relations between the two intervals or points. A **consistent scenario** is a labeling of the graph where every label is a singleton set (a set consisting of a single basic relation) and it is possible to map the vertices to a time

line and have the single relations between vertices hold. The set of **feasible relations** between two vertices consists of only the elements (basic relations) in that label capable of being part of a consistent scenario. Finding the feasible relations involves removing only those elements from the labels that could not be part of a consistent scenario.

An Example. Here is an example from the interval algebra of our two reasoning tasks. Suppose Event A either overlaps or starts Event B, but we are not sure which, and Event B meets Event C. We represent this as follows



where the label I, the set of all basic relations, shows we have no direct knowledge of the relationship between A and C. There are two possible answers to problem 1: find a scenario that is consistent with the information provided.

It remains to answer problem 2: find all the feasible relations between every pair of intervals. The only change is that the feasible relations between A and C are just the "before" relation. We see that this is true in the diagram above. No other relation between A and C can be part of a consistent scenario.

3. The Point Algebra and a Subset of the Interval Algebra

In this section we examine the computational problems of finding consistent scenarios and finding the feasible relations for the point algebra, **PA**, and the corresponding subset of the interval algebra, **SIA**.

3.1. Finding Consistent Scenarios

Review of previous solutions. Ladkin & Maddux [15] give an algorithm for finding one consistent scenario that takes $O(n^3)$ time for **PA** networks with n points. If no consistent scenario exists, the algorithm reports the inconsistency. Their algorithm relies on first applying the path consistency algorithm [17, 21] before finding a consistent scenario.

An improved solution. Here we give an algorithm for finding one consistent scenario that takes $O(n^2)$ time for **PA** networks with n points. Our starting point is an observation by Ladkin & Maddux [16, p.34] that topological sort alone will not work as the labels may be any one of the eight different **PA** elements, $\{\emptyset, <, \leq, =, >, \geq, \neq, ?\}$, and thus may have less information about the relationship between two points than is required. For top sort we need all edges labeled with $<$, $>$, or $?$ (see [12]). The "problem" labels are then $\{=, \emptyset, \leq, \geq, \neq\}$. The intuition behind the algorithm is that we somehow remove or rule out each of these possibilities and, once we have, we can then

Input: A **PA** network represented as a matrix C where element C_{ij} is the set of possible relations on edge (i, j).

Output: A consistent scenario (a linear ordering of the points).

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

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

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

If the empty label, $\emptyset$, results on any edge, then the network is inconsistent.

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

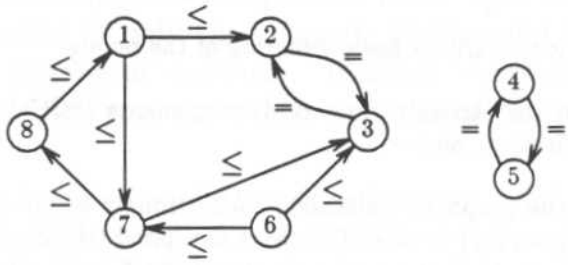
Fig. 1. Consistent Scenario Algorithm for PA Networks

apply top sort to give a consistent scenario. Much of the discussion to follow relies on the assumption that looking at paths (the transitivity information) is sufficient for deciding the label on an edge. The only exception to the truth of the assumption is that looking at paths will sometimes assign a label of $\leq$ instead of $<$ (see Sec. 3.2) but this will not affect the discussion.

Step 1: The = relation. To remove the $=$ relation from the network, we identify all pairs of points that are forced to be equal and condense them into one vertex. By forced to be equal, we mean that in all consistent scenarios, the vertices are equal so no other relation will result in a consistent scenario. More formally, we want to partition the vertices into equivalence classes S_i, $1 \leq i \leq m$, such that vertices v and w are in the same equivalence class if and only if they are forced to be equal. But, the vertices v and w are forced to be equal precisely when there is a path of the form

$$u \leq v \leq \cdots \leq w \leq u$$

where one or more of the $\leq$ can be $=$. This is the same as saying v and w are in the same equivalence class if and only if there is a path from v to w and a path from w to v using only the edges labeled with $\leq$ or $=$. This is a well-known problem in graph theory. Determining the equivalence classes is the same as identifying the strongly connected components of the graph and efficient algorithms are known (Tarjan [24], see also [1]). An example is shown in Fig. 2. Only $\leq$ and $=$ edges are shown except that self-loops are also omitted (each vertex is equal to itself). There are four strongly connected components.



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

Fig. 2. Strongly Connected Components

Step 1: The $\varnothing$ relation. To rule out the $\varnothing$ relation we must determine if the the network is inconsistent. The network is inconsistent if a vertex is forced to be $<$, $>$, or $\neq$ to itself. That is, when there is a path of the form

$$u = v = \cdots = w \neq u$$

or of the form

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

where all but one of the $<$ can be $\leq$ or $=$. We can identify these cases simply by also looking at edges labeled with $<$ when identifying the strongly connected components. The inconsistencies are then detected when the vertices are collapsed. For example, suppose the label on the edge (1, 7) in the graph shown in Fig. 2 was $<$ instead of the $\leq$ shown. Condensing the strongly connected component S_1 gives

$$\begin{aligned}\hat{C}_{22} &= C_{17} \cap C_{18} \cap C_{71} \cap C_{78} \cap C_{81} \cap C_{87} \\ &= \{<\} \cap \{>, =\} \cap \{>\} \cap \{<, =\} \cap \{<, =\} \cap \{>, =\} = \varnothing\end{aligned}$$

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

Step 2: The $\leq, \geq$ relations. To remove the $\leq$ relation from the network, we simply change all $\leq$ labels to $<$. As a result of Step 1, we know a consistent scenario exists and that no remaining edge is forced to have $=$ as its label in the consistent scenario. Changing $\leq$ to $<$ can only force other labels to become $<$; it cannot force labels to become $=$. So after the changes, a consistent scenario will still exist. We can also show that no new inconsistencies are introduced by this step.

Step 2: The $\neq$ relation. We can now perform topological sort to find one consistent scenario. The $\neq$ relations will be handled implicitly by top sort as all the vertices in $\hat{C}$ will be assigned a different number.

Theorem 1. *The algorithm in Fig. 1 correctly solves the consistent scenario problem for* **PA** *and* **SIA** *networks in $O(n^2)$ time.*

Note that for **SIA** networks we just first translate the network into a **PA** network, solve, then translate back. For the time bound, finding the strongly connected components is $O(n^2)$ [24], condensing the graph looks at each edge only once, and topological sort is $O(n^2)$ [12]. It is easy to see that the algorithm is asymptotically optimal as we must at least examine every edge in the network, of which there may be as many as $O(n^2)$. If we do not, we can never be sure that the label on that edge does not involve a contradiction by, for example, being part of a loop that causes a vertex to be less than itself.

Some applications of the results. Koubarakis et al. [13] use a subset of **SIA** in a knowledge representation language so the algorithm given in Fig. 1 is applicable. The decision portion of the algorithm, for example, could be used when adding new information to the network to detect if the new information is inconsistent with the old. **PA** may be useful in planning and scheduling where a common approach is to model actions or resources by points and there are precedence constraints and constraints on two actions co-occurring. Here a consistent scenario corresponds to a plan or a schedule. Finally, the algorithm will be shown to be useful in designing algorithms for **IA** (Sec. 4.1).

3.2. Determining the Feasible Relationships

Review of previous solutions. Ghallab & Mounir Alaoui [9] give an incremental procedure, based on a structure called a maximal indexed spanning tree, that is shown to work well in practice. The path consistency algorithm (PC) [17, 21] can be used to find approximations to the sets of all feasible relations [2]. Much previous work are efforts at identifying classes of relations for which PC will give exact answers. Montanari [21] shows that PC is exact for a restricted class of binary constraint relations. However, the relations of interest here do not all fall into this class. Valdés-Pérez [25] shows that PC is exact for the basic relations of **IA**. In [26, 30], we show that PC is exact for a subset of **PA** and a corresponding subset of **SIA**. The new point algebra differs from **PA** only in that $\neq$ is excluded from the underlying set. But we also give examples there that show that, earlier claims to the contrary, the path consistency algorithm is not exact for **PA** nor for **SIA** networks and we develop an $O(n^4)$ four-consistency algorithm that is exact.

An improved solution. Here we give an algorithm for finding all feasible relations that, under certain assumptions, takes $O(n^3)$ time for **PA** networks with n points. The algorithm is of more practical use than the previously known $O(n^4)$ algorithm.

Our strategy for developing an algorithm for **PA** is to first identify why path consistency is sufficient if we exclude $\neq$ from the language and is not sufficient if we include $\neq$. Fig. 3 gives the smallest counter-example showing that the path consistency algorithm is not exact for **PA**. The graph is path consistent. But it is easy to see that not every relation in the set of possible relations between s and t is feasible. In particular, asserting $s = t$ forces v and w to also be equal to s and t (pointer to previous section). But this is inconsistent with $v \neq w$. Hence, the $=$ relation is not

feasible as it is not capable of being part of a consistent scenario. The label between s and t should be $<$.

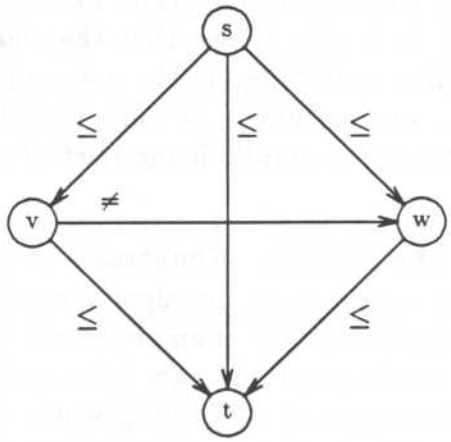


Fig. 3. "Forbidden" Subgraph

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

Theorem 2 (van Beek & Cohen [28]). *The network in Fig. 3 is the smallest counter-example to the exactness of path consistency for* **PA** *and, up to isomorphism, is the only counter-example of four vertices. Also, any larger counter-example must have a subgraph of four vertices isomorphic to the example.*

We shall solve the feasible relations problem by first applying the path consistency algorithm and then systematically searching for "forbidden" subgraphs and appropriately changing the labels (see Fig. 4; the path consistency algorithm is slightly simplified because of properties of the algebras). The algorithm makes use of adjacency lists. For example, $adj_{\leq}(v)$ is the list of all vertices, w, for which there is an edge from v to w that is labeled with '$\leq$'. We assume $w \in adj_{\neq}(v)$ only if $v < w$ so that we only process each such vertex once.

Changing the label on an edge (s, t) from '$\leq$' to '$<$' may further constrain other edges. The question immediately arises of whether we need to again apply the path consistency algorithm following our search for "forbidden" subgraphs to propagate the newly changed labels? Fortunately, the answer is no. Given a new label on the edge (s, t), if we were to apply the path consistency algorithm, the set of possible triangles that would be examined is given by $\{ (s, t, k), (k, s, t) \mid 1 \leq k \leq n, k \neq s, k \neq t \}$ (see procedure RELATED_PATHS in Fig. 4). Thus there are two cases. For both, we can show that any changes that the path consistency algorithm would make will already have been made by procedure FIND_SUBGRAPHS.

Input: A **PA** network represented as a matrix C where element C_{ij} is the set of possible relations on edge (i, j).
Output: The set of feasible relations for C_{ij}, $i, j = 1, \ldots, n$.

```
procedure FEASIBLE
begin
    PATH_CONSISTENCY
    FIND_SUBGRAPHS
end

procedure PATH_CONSISTENCY
begin
    Q ← ⋃_{1 ≤ i < j ≤ n} RELATED_PATHS (i, j)
    while Q is not empty do begin
        select and delete a path (i, k, j) from Q
        t ← C_ij ∩ C_ik · C_kj
        if (t ≠ C_ij) then begin
            C_ij ← t
            C_ji ← CONVERSE (t)
            Q ← Q ∪ RELATED_PATHS (i, j)
        end
    end
end

procedure RELATED_PATHS (i, j)
    return { (i, j, k), (k, i, j) | 1 ≤ k ≤ n, k ≠ i, k ≠ j }

procedure FIND_SUBGRAPHS
begin
    for each v such that adj_≠(v) ≠ ∅ do
        for each s ∈ adj_≥(v) do
            for each t ∈ adj_≤(v) do
                if (adj_≤(s) ∩ adj_≠(v) ∩ adj_≥(t) ≠ ∅) then begin
                    C_st ← '<'
                    C_ts ← '>'
                end
end
```

Fig. 4. Feasible Relations Algorithm for PA Networks

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

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

In both, the label on (s, k) will become '$<$'. For (s, t) to change we must have the situation depicted in Fig 3., for some v and w. But $v \leq t$ and $w \leq t$ together with $t \leq k$ (or $t = k$) imply that $v \leq k$ and $w \leq k$ (we can assume the relations were propagated because we applied the path consistency algorithm before the procedure for

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

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

Theorem 3. *The algorithm in Fig. 4 correctly solves the feasible relations problem for* **PA** *and* **SIA** *networks.*

Note that for **SIA** networks we just first translate the network into a **PA** network, solve, then translate back. A desirable feature of procedure FIND_SUBGRAPHS is that its cost is proportional to the number of edges labeled $\neq$. For a worst case bound, the path consistency algorithm is $O(n^3)$ [18], so the algorithm is $O(n^3)$ if we assume that the adjacency lists are represented as bit vectors and that intersecting them takes one unit of time. This is a reasonable assumption since even for n up to 100 this only takes a few instructions on most computers. The algorithm was implemented and tested on random problems up to size 100. It was found that about 90% of the time was spent in the path consistency algorithm and only about 2% in FIND_SUBGRAPHS.

Some applications of the results. Song & Cohen [23] and Nökel [22] both use a subset of **SIA** so the algorithm given in Fig. 4 is applicable (although both restrict themselves to the subset of **SIA** that does not need $\neq$, the results here show that the expressive power of their temporal language could be expanded without compromising efficiency or exactness). Finally, the algorithm will be shown to be useful in designing algorithms for **IA** (Sec. 4.2).

4. The Full Interval Algebra

In this section we examine the computational problems of finding consistent scenarios and finding the feasible relations between intervals for the full interval algebra, **IA**. Vilain & Kautz [29, 30] show that both of these problems are NP-Complete for the interval algebra. Thus the worst cases of the algorithms that we devise will be exponential and the best we can hope for is that the algorithms are still useful in practice. We discuss to what extent this is achieved below.

4.1. Finding Consistent Scenarios

Review of previous solutions. Allen [2] proposes using simple backtracking search to find one consistent scenario of an **IA** network or report inconsistency. Valdés-Pérez [25] gives a dependency-directed backtracking algorithm. Both search through the alternative singleton edge labelings. As well, there has been much work on improving the performance of backtracking that could be applied to this problem (see [7] and references therein).

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

The key idea is that the $O(n^2)$ decision procedure for **SIA** networks (Step 1 of Fig. 1) can be used effectively to decide whether a partial solution found so far is consistent and so might be part of a solution to the whole problem. Instead of searching through alternative singleton labelings, as Allen and Valdés-Pérez do, we can now decompose the labels into elements of **SIA**. For example, if the label on an edge is {b, bi, m, o, oi, si }, there are six alternative singleton edge labelings but only two when decomposed into elements of **SIA**: {b, m, o} and {bi, oi, si}. It is easy to see that this is guaranteed to be better since, for any choice of a singleton label, we can choose a label of larger cardinality that is a superset of the singleton label. If the singleton label is consistent, so is the larger label. And, of course, there will be times when the larger label is consistent and the singleton label is not. The output of the backtracking algorithm will be a consistently labeled **SIA** network and the scenario algorithm for **SIA** (Fig. 1) is then used to find a consistent scenario.

The algorithm was implemented and tested on random instances from a distribution designed to approximate planning applications (as estimated from a block-stacking example in [5]). For a problem size of $n = 20$, the average time to find a solution was about seven seconds of CPU time (25 tests performed). For $n = 40$, it was 74 seconds (average over 21 tests). This seems surprisingly fast. However, it should be noted that four of the tests for $n = 40$ were not included as they were stopped before completion as a limit on the number of consistency checks was exceeded. We are currently examining the best order in which to search through the labels (for example, perhaps the most restrictive relations should be tried first) to further avoid this occasional thrashing behavior†.

Some applications of the results. In planning, as formulated by Allen and Koomen [5] and Hogge [10], actions are associated with the intervals they hold over and the full interval algebra is used. Finding one consistent scenario corresponds to finding an ordering of the actions that will accomplish a goal. Hence, the results here are directly applicable.

4.2. Determining the Feasible Relationships

A solution. A similar backtracking algorithm as in the previous section can be designed for finding all the feasible relations. Again, instead of searching through the alternative singleton (or basic) labelings of the edges, we search through the alternative decompositions of the labels into elements of **SIA**. For each labeling of the network with elements of **SIA** that is consistent (determined using the decision portion of the algorithm in Sec 3.1), we find the feasible relations (using the algorithm in Sec 3.2). The feasible relations for the **IA** network is then just the union of all such solutions. Initial experience, however, suggests this method is practical only for very small instances of the problem (but there exists methods of decomposing large problems into smaller ones [3, Dechter]), or for instances where only a few of the relations between intervals fall outside of the special subset **SIA**. We conclude that in most cases a

† It would be interesting to compare with the dependency-directed backtracking approach of Valdés-Pérez. To the best of our knowledge no practical experience has been reported and no implementation made (Valdés-Pérez, personal communication).

better approach is to, if possible, accept approximate solutions to the problem (Allen [2], van Beek [26]).

Some applications of the results. The approach given above may be useful when much preprocessing time is available or exact answers are important. One example is the construction of a plan library for plan recognition. Here the construction is done once and the library is used many times to recognize plans (see [11] for the use of **IA** in this setting).

References

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

[2] Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. *Comm. ACM* **26**, 832-843.

[3] Allen, J. F. 1984. Towards a General Theory of Action and Time. *Artificial Intelligence* **23**, 123-154.

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

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

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

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

[8] Dechter, R., I. Meiri, and J. Pearl. 1989. Temporal Constraint Networks. *Proc. of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ont., 83-93.

[9] Ghallab, M., and A. Mounir Alaoui. 1989. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. *Proc. of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Mich., 1297-1303.

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

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

[12] Knuth, D. E. 1973. *Sorting and Searching.* Addison-Wesley, 258-265.

[13] Koubarakis, M., J. Mylopoulos, M. Stanley, and A. Borgida. 1989. Telos: Features and Formalization. Knowledge Representation and Reasoning Technical Report KRR-TR-89-4, Department of Computer Science, University of Toronto.

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

[15] Ladkin, P. B., and R. Maddux. 1988. On Binary Constraint Networks. Technical Report, Kestrel Institute, Palo Alto, Calif.

[16] Ladkin, P. B., and R. Maddux. 1988. The Algebra of Constraint Satisfaction Problems and Temporal Reasoning. Technical Report, Kestrel Institute, Palo Alto, Calif.

[17] Mackworth, A. K. 1977. Consistency in Networks of Relations. *Artificial Intelligence* **8**, 99-118.

[18] Mackworth, A. K., and E. C. Freuder. 1985. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence* **25**, 65-74.

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

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

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

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

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

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

[25] Valdés-Pérez, R. E. 1987. The Satisfiability of Temporal Constraint Networks. *Proc. of the Sixth National Conference on Artificial Intelligence*, Seattle, Wash., 256-260.

[26] van Beek, P. 1989. Approximation Algorithms for Temporal Reasoning. *Proc. of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Mich., 1291-1296.

[27] van Beek, P. 1990. Reasoning about Qualitative Temporal Information (Extended version). Dept. of Computer Science Technical Report, University of Waterloo.

[28] van Beek, P., and R. Cohen. 1989. Approximation Algorithms for Temporal Reasoning (Extended version). Department of Computer Science Technical Report CS-89-12, University of Waterloo.

[29] Vilain, M., and H. Kautz. 1986. Constraint Propagation Algorithms for Temporal Reasoning. *Proc. of the Fifth National Conference on Artificial Intelligence*, Philadelphia, Pa., 377-382.

[30] Vilain, M., H. Kautz, and P. van Beek. 1989. Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report. In *Readings in Qualitative Reasoning about Physical Systems*, D. S. Weld and J. de Kleer (eds.), Morgan-Kaufman, 373-381.

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