Reasoning about Qualitative Temporal Information

Peter van Beek

Department of Computer Science[1]
University of Waterloo
Waterloo, Ontario, CANADA N2L 3G1
pgvanbeek@dragon.waterloo.edu

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 aid in the design of a backtracking algorithm for the full interval algebra that is useful in practice.

Introduction

Much temporal information is qualitative information such as "The Cuban Missile crisis took place during Kennedy's presidency," where only the ordering of the end points of the two events is specified. Allen [1] has proposed an interval algebra and Vilain & Kautz [20] have proposed a point algebra for representing such qualitative information. In this paper, we address two fundamental 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.
2. find all the feasible relations between every pair of intervals or points[2].

Specific applications of solutions to these tasks include natural language processing (Allen [2]), planning (Allen & Koomen [4]), and a knowledge representation language (Koubarakis et al. [10]). As well, the techniques developed here could be part of a specialist in a general temporal reasoning system 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. [7], Dean [5]), or combinations of qualitative and quantitative information (Allen & Kautz [3], Ladkin [11]).

The main results of the paper are as follows. For the point algebra and for a corresponding subset of the interval algebra, we give computationally efficient procedures for solving both tasks 1 & 2. Our algorithms are marked improvements over the previously known algorithms. In particular, we develop an $O(n^2)$ time algorithm for finding one consistent scenario that is an $O(n)$ improvement over the previously known algorithm [12], where n is the number of intervals or points, and we develop an algorithm for finding all the feasible relations that is of far more practical use than the previously known algorithm [18].

For the full interval algebra, Vilain & Kautz [20, 21] 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 aid in the design of 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 [1], van Beek & Cohen [18, 19]).

Background, Definitions, and 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. Interval algebra, **IA** (Allen [1]). There are thirteen basic relations (including inverses) that can hold between two intervals.

[1] Author's current address: Department of Computing Science, University of Alberta, Edmonton, Alberta, CANADA T6G 2H1.

[2] The terminology is from [7]. Other names for task 1 include consistent singleton labeling [18] and a satisfying assignment of values to the variables [12]. Other names for task 2 include deductive closure [21], minimal labeling [18] and, as it arises as a general constraint satisfaction problem, minimal network [15].

relation	symbol	inverse	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

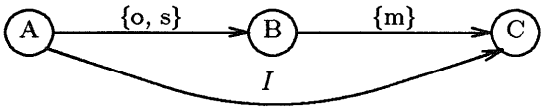
We want to be able to represent indefinite information so we allow the relationship between two intervals to be a disjunction of the basic relations. We use sets to list the disjunctions. Somewhat more formally, let I be the set of all basic relations, {eq, b, bi, m, mi, o, oi, d, di, s, si, f, fi}. **IA** is the algebraic structure with underlying set 2^I, the power set of I, unary operator inverse, and binary operators intersection and composition (denoted "constraints" in [1]; see that reference for the definition).

Definition. Point algebra, **PA** (Vilain & Kautz [20]). 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 inverse, and binary operators intersection and composition (denoted addition and multiplication in [20] where the operators are defined over bit vector representations of the underlying set; see that reference for the definitions). Note that $\leq$, for example, is an abbreviation of $\{<, =\}$, $\emptyset$ is the inconsistent constraint, and $?$ means there is no constraint between two points, $\{<, =, >\}$.

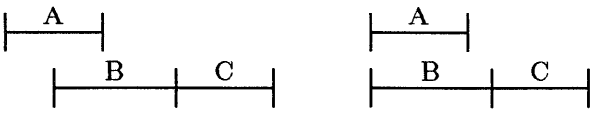
Vilain & Kautz show that a subset of the interval algebra can be translated into their point algebra. We denote as **SIA** the subset of the underlying set of the interval algebra that can be translated into relations between the endpoints of the intervals using the underlying set of **PA** (see [19] 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 basic 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.

Here is an example from the interval algebra of our two reasoning tasks. Suppose interval A either overlaps or starts interval B, but we are not sure which, and interval B meets interval 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 consistent scenario. Their mappings to a time line are shown below. In the mapping on the left, A overlaps B, in the one on the right, A starts B, and in both mappings B meets C and A is before C.



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

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**.

Finding Consistent Scenarios

Ladkin & Maddux [12] 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 [13, 15] before finding a consistent scenario.

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 [12, 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

Input: A **PA** network represented as a matrix C where element C_{ij} is the label 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 $\{S_1, S_2, \ldots, S_m\}$ be the SCCs we have found (the S_i partition the vertices in the graph in that each vertex is in one and only one of the S_i). We construct the condensed graph and its matrix representation, $\hat{C}$, as follows. Each S_i is a vertex in the graph. The labels on the edges between all pairs of vertices is given by

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

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 $<$ and perform a topological sort using only the edges in $\hat{C}$ labeled with $<$.

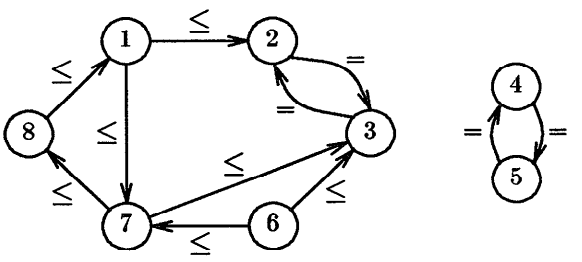
Fig. 1. Cons. Scenario Alg. for PA Networks

with $<$, $>$, or $?$ (see [9]). 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 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 Fig. 3) 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 every consistent scenario 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 cycle 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 [16]). 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. Condensing the strongly connected components is described in Step 1 of the algorithm shown in Fig. 1.



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

Fig. 2. Strongly Connected Components

Step 1: The $\emptyset$ relation. To rule out the $\emptyset$ 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 cycle 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 $=$. It turns out that 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 strongly connected components are condensed (Step 1 of Fig. 1). 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

$$\begin{aligned} \hat{C}_{11} &\leftarrow C_{17} \cap C_{18} \cap C_{71} \cap C_{78} \cap C_{81} \cap C_{87} \\ &\leftarrow \{<\} \cap \{>,=\} \cap \{>\} \cap \{<,=\} \cap \{<,=\} \cap \{>,=\} \\ &\leftarrow \emptyset \end{aligned}$$

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 $<$. This is valid because, as a result of Step 1, we know that a consistent scenario exists and that no remaining edge is forced to have $=$ as its label in all consistent scenarios. So, for any particular edge labeled with $\leq$ there exists a consistent scenario with $<$ as the singleton label. But, changing a $\leq$ to a $<$

can only force other labels to become $<$; it cannot force labels to become $=$. (Using the terminology of the algorithm, no new strongly connected components are introduced by this step; hence no new labels are forced to be equal and no new inconsistencies are introduced.) So, after all the changes, a consistent scenario will still exist.

Step 2: The $\neq$ relation. We can now perform topological sort to find one consistent scenario. It can be shown that, because of the previous steps of the algorithm, the $\neq$ relations will now be handled correctly (and implicitly) by top sort. The output of top sort is an assignment of numbers to the vertices (a mapping of the vertices to a time line) that is consistent with the information provided. As an example, consider the algorithm in Fig. 1 applied to the network in Fig. 3. Depending on the particular implementation of top sort, one possible result of the algorithm is the following assignment of numbers to vertices: $s \leftarrow 0$, $v \leftarrow 1$, $w \leftarrow 2$, and $t \leftarrow 3$.

Theorem 1. *The algorithm in Fig. 1 correctly solves the consistent scenario problem for* **PA** *and* **SIA** *networks in $O(n^2)$ time, where n is the number of points or intervals.*

Note that for **SIA** networks we must first translate the network into a **PA** network, solve, then translate back. For the time bound, finding the strongly connected components is $O(n^2)$ [16], condensing the graph looks at each edge only once, and topological sort is $O(n^2)$ [9]. 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 not be sure that the label on that edge is not involved in a contradiction by, for example, being part of a loop that causes a vertex to be less than itself.

Determining the Feasible Relationships

Ghallab & Mounir Alaoui [8] 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) [13, 15] can be used to find approximations to the sets of all feasible relations [1]. Much previous work are efforts at identifying classes of relations for which PC will give exact answers. Montanari [15] 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 [17] shows that PC is exact for the basic relations of **IA**. In [18, 21], 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 $O(n^4)$ strong four-consistency algorithm that is exact, where n is the number of intervals or points.

Here we give an algorithm for finding all feasible relations that, while still $O(n^4)$ in the worst case for **PA** networks with n points, is of far more practical use than our previous algorithm (that algorithm is still of importance as an approximation algorithm for instances of the problem from the full interval algebra; see [18, 21] for the details).

Our strategy for developing an algorithm for **PA** networks is to first identify why path consistency is sufficient if we exclude $\neq$ from the language and is not sufficient if we include $\neq$. 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 basic relation in the label between s and t is feasible. In particular, asserting $s = t$ forces v and w to also be equal to s and t. 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 $<$.

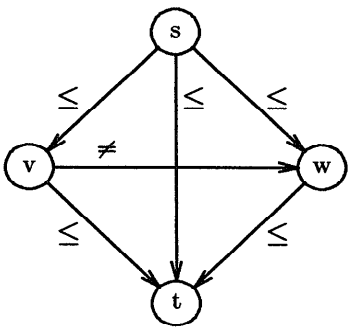


Fig. 3. "Forbidden" Subgraph

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 [19]). *The network in Fig. 3 is the smallest counter-example to the exactness of path consistency for* **PA** *networks 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

Input: A **PA** network represented as a matrix C where element C_{ij} is the label 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)
    begin
        select and delete a path (i, k, j) from Q
        t ← C_ij ∩ C_ik · C_kj
        if (t ≠ C_ij)
        begin
            C_ij ← t
            C_ji ← INVERSE (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) ≠ ∅
        for each s ∈ adj_≤(v)
            for each t ∈ adj_≤(v)
                if (adj_≤(s) ∩ adj_≠(v) ∩ adj_≥(t) ≠ ∅)
                begin
                    C_st ← '<'
                    C_ts ← '>'
                end
end
```

Fig. 4. Feas. Relations Alg. for PA Networks

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 '$\leq$'.

Changing the label on some 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 an edge (s, t), if we were to apply the path consistency algorithm, the set of possible triangles that would be examined is given by $\{(s, t, k), (k, s, t) \mid 1 \leq k \leq n, k \neq s, k \neq t\}$ (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.

Case 1: (s, t, k). Changing the label on the edge (s, t) from '$\leq$' to '$<$' would 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) also belongs to a "forbidden" subgraph and the label on that edge will have been found and updated.

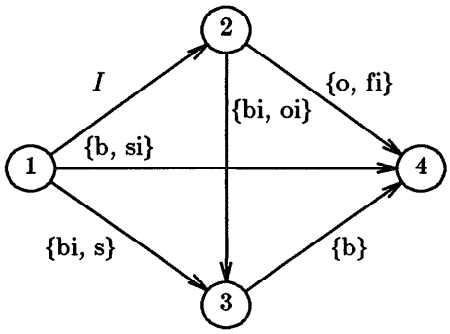
Case 2: (k, s, t). Similar argument as Case 1.

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

Note that for **SIA** networks we must first translate the network into a **PA** network, solve, then translate back. For a time bound, the path consistency procedure is $O(n^3)$ [14] and the find subgraphs procedure is easily shown to take $O(n^4)$ time in the worst case, where n is the number of points. This is the same as the previously known algorithm [18]. However, this comparison is misleading, as the algorithm in [18] always takes $O(n^4)$ time, no matter what the input is. A desirable feature of procedure FIND_SUBGRAPHS is that its cost is proportional to the number of edges labeled $\neq$. The worst cases for the algorithm are contrived and presumably would rarely occur. As experimental evidence, the algorithm was implemented in a straightforward way 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. Hence, the $O(n^3)$ path consistency procedure dominates the computation.

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 [20, 21] 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.



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

Fig. 5. Example IA Network

Finding Consistent Scenarios

Allen [1] proposes using simple backtracking search to find one consistent scenario of an **IA** network or report inconsistency. Valdés-Pérez [17] gives a dependency-directed backtracking algorithm. Both search through the alternative singleton labelings. As well, there has been much work on improving the performance of backtracking that could be applied to this problem (see [6] and references therein).

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 to decide whether a partial solution found so far is consistent (acceptable) and so might be part of a solution to the whole problem. The benefits go beyond a fast test for acceptability. Whereas Allen and Valdés-Pérez search through alternative singleton labelings, we can now reduce the cardinality of the domains we are searching through by decomposing the labels into the largest possible elements of **SIA**. For example, if the label on an edge is {b, bi, m, o, oi, si}, there are six possible ways to label the edge with a singleton label: {b}, {bi}, {m}, {o}, {oi}, {si}, but only two possible ways to label the edge if we decompose the labels into the largest possible elements of **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 (or equal) cardinality that is a superset of the singleton label. If the singleton label is consistent, so is the larger label. And, of course, there will be times when the larger label is consistent and the singleton label is not.

Recall that what we want to find is a labeling of the edges of the graph such that every label contains a single basic relation and it is possible to map the vertices to a time line and have the single relations between vertices hold. Finding a consistent scenario is now done in two stages: the output of the backtracking algorithm will be a consistent **SIA** network

Single:

(1,2)	(1,3)	(2,3)	(1,4)	(2,4)	(3,4)
{eq}					
	{bi}				
		{bi}			
			{b}		
				{o}	
				{fi}	
			{si}		
				{o}	
				{fi}	
		{oi}			
	{s}				
		{bi}			
		{oi}			
{b}					
	{bi}				
		{bi}			
			{b}		
				{o}	
					{b}

SIA:

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

Fig. 6. Backtrack Search

and the scenario algorithm for **SIA** networks (Fig. 1) is then used to find a consistent scenario of this network.

As an example, consider the network shown in Fig. 5. The backtrack search will look at the edges in the order (1,2), (1,3), (2,3), (1,4), (2,4), and (3,4). A record of both methods of search is shown in Fig. 6. Moving to the right and downward in the figure means a partial solution is being extended, moving to the left and downward means the search is backtracking. Note that, for example, when searching through alternative singleton labelings, much search is done before it is discovered that no consistent scenario exists with edge (1,2) labeled with {eq}, but when decomposing the labels into the largest possible elements of **SIA** and searching through the decompositions, no backtracking is necessary.

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 [4]). In planning, as formulated by Allen and Koomen [4], 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. 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.

Determining the Feasible Relationships

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 labelings of the edges, we decompose the labels into the largest possible elements of **SIA** and search through the decompositions. In the previous section when finding a consistent scenario we stopped the backtracking algorithm after one consistent **SIA** network was found. To determine the feasible relations we must find all such consistent **SIA** networks. For each such consistent **SIA** network we find the feasible relations using the algorithm of Fig. 4. 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 small instances of the problem, 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 better approach is to, if possible, accept approximate solutions to the problem (Allen [1], van Beek & Cohen [18, 19]).

Acknowledgements. Many thanks to my supervisor Robin Cohen and to Fahiem Bacchus, Charlie Colbourn, Fei Song, Bruce Spencer, and Paul van Arragon for help, advice, and encouragement and to Peter Ladkin for fruitful discussions over the internet.

References

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

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

[3] 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.

[4] Allen, J. F., and J. A. Koomen. 1983. Planning Using a Temporal World Model. *Proc. of the 8th IJCAI*, 741-747.

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

[6] Dechter, R., and I. Meiri. 1989. Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems. *Proc. of the 11th IJCAI*, 271-277.

[7] Dechter, R., I. Meiri, and J. Pearl. 1989. Temporal Constraint Networks. *Proc. of the 1st Int. Conf. on Principles of Knowledge Representation and Reasoning*, 83-93.

[8] Ghallab, M., and A. Mounir Alaoui. 1989. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. *Proc. of the 11th IJCAI*, 1297-1303.

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

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

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

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

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

[14] 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.

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

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

[17] Valdés-Pérez, R. E. 1987. The Satisfiability of Temporal Constraint Networks. *Proc. of the 6th National Conf. on AI*, 256-260.

[18] van Beek, P. 1989. Approximation Algorithms for Temporal Reasoning. *Proc. of the 11th IJCAI*, 1291-1296.

[19] van Beek, P., and R. Cohen. 1990. Exact and Approximate Reasoning about Temporal Relations. *Computational Intelligence.* To appear.

[20] Vilain, M., and H. Kautz. 1986. Constraint Propagation Algorithms for Temporal Reasoning. *Proc. of the 5th National Conf. on AI*, 377-382.

[21] 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.