The Satisfiability of Temporal Constraint Networks[1]

Raúl E. Valdés-Pérez

Computer Science Department
Carnegie-Mellon University
Pittsburgh PA 15213

Abstract

A popular representation of events and their relative alignment in time is James Allen's intervals and algebra. Networks of disjunctive interval constraints have served both to assimilate knowledge from ambiguous sentences, and to hold partial solutions in a planner. The satisfiability of these networks is of practical concern, and little has been achieved beyond proving that determining satisfiability is NP-hard. This paper scrutinizes the interval representation and its mechanisms. We make explicit the unstated assumptions of the mechanisms, introduce several useful theorems regarding interval networks, distinguish three types of inconsistency exhibited by these networks, and point out under what conditions these inconsistencies are detected. Finally the theorems, observations, and distinctions regarding inconsistency are exploited to design a practical algorithm to determine the satisfiability of an interval network. The extension of our results to two-dimensional spatial reasoning is under investigation.

1. Introduction

One way to represent events extending over time is by the use of the interval algebra, popularized by Allen [Allen 83], and incorporated into the planner in [Allen & Koomen 83]. Some problems accompanying its use have been cited [Vilain & Kautz 86], notably the lack of a suitable practical algorithm to determine the satisfiability of a set of assertions in the interval algebra.

This paper mathematically characterizes Allen's interval algebra and makes explicit the assumptions that underlie it. We treat the issue of satisfiability in the light of two new theorems regarding networks of intervals. The insight provided by these theorems and other observations is exploited to state a practical algorithm to determine the satisfiability of a given interval network. Finally, the development here should suggest a way to analyze disjunctive constraint networks that use a different algebra.

2. Events as Intervals

Simple intervals are convenient to represent events that began and ended, and that occurred continuously between those two times. An example of such an event is a visit paid to a friend on a previous day. The use of intervals to depict the temporal extent of such events leads to a temporal ordering of these events by comparing the interval endpoints. By considering all alignments of the four endpoints, one arrives at Allen's thirteen possible orderings between two intervals, shown in the following Table.

Allen's 13 Interval Orderings			
<	before	>	after
m	meets	mi	met-by
o	overlaps	oi	overlapped-by
s	starts	si	started-by
f	finishes	fi	finished-by
d	during	di	contains
=	equals		

So, for example, the Carter presidency "meets" the Reagan presidency, because the end of the one event coincides with the beginning of the other.

However, the meaning of certain linguistic assertions is not captured by any *single* ordering. A statement such as

> She telephoned my friend during my visit yesterday at his home.

requires a disjunction of orderings, to express the ignorance of whether the telephone call ended before, after, or at the same time as the visit. We denote the disjunctive relation between two intervals by a set or list of orderings. These intervals are treated as *unknowns*, because if the positions of two events along the time dimension were known precisely, then the relation between them would of course be a single ordering.

$$A \xrightarrow{\{<\ >\}} B \qquad \begin{array}{c} A_l \xrightarrow{<} A_u \xrightarrow{<} B_l \xrightarrow{<} B_u \\ \boxed{\text{or}} \\ B_l \xrightarrow{<} B_u \xrightarrow{<} A_l \xrightarrow{<} A_u \end{array}$$

Figure 1: Equivalent Representations

An advantage of depicting events as intervals, versus an equivalent endpoint-based representation, is the concise way that a disjunctive relation between two events is expressed by a single relation. This conciseness has favorable computational consequences, as discussed below. In Figure 1, disjointness {< >} is shown as an interval relation on the left, and as an equivalent disjunction of endpoint relations on the right.[2] We remark that while the interval relation is along a *directed* edge, the relation in the reverse direction obtains by simply inverting each ordering, according to the lines in the Table above.

3. Inference Through Transitivity

It is possible to obtain a relation between two intervals despite the lack of an assertion directly mentioning the intervals. [Allen 83] gives a table for calculating a relation between two intervals A and C by combining the known relation of each with interval B. Given AR_1B and BR_2C, the relation $R_1 \bullet R_2$ between A and C

[1]This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, monitored by the Air Force Avionics Laboratory under contract F33615-84-K-1520.

[2]The subscripts 'l' and 'u' denote respectively the lower and upper endpoints of an interval.

follows by forming the cross product of the sets R_1 and R_2, composing each resulting ordered pair by looking up the result in the transitivity table, and taking the union of the resulting sets. For example,

{o d s} • {s si =} =

tt(o,s) ∪ tt(o,si) ∪ tt(o,=) ∪ tt(d,s) ∪ tt(d,si) ∪
tt(d,=) ∪ tt(s,s) ∪ tt(s,si) ∪ tt(s,=) =

{o} ∪ {di fi o} ∪ {o} ∪ {d} ∪ {> oi mi d f} ∪
{d} ∪ {s} ∪ {s si =} ∪ {s} =

{o di fi d > oi mi f s si =}.

The relations in the first line are interpreted linguistically as 'ended while the other occurred', and 'started at the same time as.' The transitive relation means 'ended after the other started.'

Newly inferred relations, such as the above, have to be reconciled with the current direct relation. For the interval algebra, the reconciliation involves conjoining the old relation with the new, because both must be true. For example, we convert each relation to its equivalent disjunction:

reconcile[{r_1 r_2 ...}, {s_1 s_2 ...}] =
$r_1 \wedge s_1 \vee r_1 \wedge s_2 \ldots \vee r_2 \wedge s_1 \vee r_2 s_2 \ldots$

Since the 13 orderings are mutually exclusive, $r_i \wedge s_j$ is *false* unless $r_i = s_j$. Therefore, the reconciliation of two interval relations is just their intersection.

Depending on the application, relations generated externally are either hypothesized by a planner or input as domain facts. Before the new relations are assimilated, we imagine a group of intervals each related by the tautological relation:

{< > m mi o oi d di s si f fi =}

consisting of all possible orderings. When the first new relation is input directly or inferred through transitivity, it is reconciled with the tautological relation.

4. Networks of Intervals

When there are many intervals, the relation between two intervals of interest may be affected by the transitive relations involving any third interval. One may desire to know the relations between several pairs of intervals, so the need arises to make explicit, or explicate, all the implicit relations obtained by transitivity. It is also useful to know whether the relations are globally consistent; if they are not, then the validity of inferences obtained by transitivity is suspect. Moreover, if the interval representation is part of a planner, as in [Allen & Koomen 83], then inconsistent hypothesized relations should be noticed, in order that the plan be realizable.

The need to compute the consequences of a set of relations is met by casting intervals and relations as the nodes and edges of a (directed) graph, or network, in order to apply known graph algorithms. The transitive closure algorithm (TCA) explicates all the relations contained in paths through the network, reconciles the different paths between each pair of nodes, and places the result in the edge connecting the pair. The algorithm will often detect inconsistencies, as discussed below. As formulated in [Aho et al. 74], the TCA terminates in time cubic in the number of nodes, at which point we say the network is closed. [Allen 83] presents an incremental version of the algorithm to use when a few new relations are added to an already closed network.

A closed network has an edge between each pair of nodes, i.e. is a complete graph. If one of the edges is the null relation, then the network is unsatisfiable. If the network has no null relations, then we ask: Does there exist a globally consistent assignment of a single ordering to each edge? If there is such a labelling, then the network is satisfiable, and we call the edge assignment a solution.

[Aho et al. 74] states sufficient conditions on a network algebra that guarantee that any closed network within the algebra is minimal, meaning that there is no smaller graph having the same solutions. If any closed network is minimal, then a closed non-null network has a solution; otherwise the null network would be smaller and have the same number of solutions, namely none. It is a desirable property that any closed network be minimal, because then the satisfiability of any network is found by the polynomial-time TCA.

Unfortunately, the interval algebra fails to meet the sufficient conditions, so that a closed network in this algebra is not necessarily minimal. [Montanari 74] also discusses conditions that guarantee minimality, which do not hold here either; this is discussed fully in [Valdes-Perez 86]. [Vilain & Kautz 86] finally proved that the question of satisfiability in these networks is NP-hard, so that the closure cannot in general be minimal.[3]

To summarize, it is desirable to explicate all the relations in a constraint network for two reasons: First, to find the tightest relation between all node pairs, and second, to detect inconsistencies of the type discussed in the next section. A closed non-null network generally is not minimal, hence possibly unsatisfiable. Further computation is needed to determine satisfiability, and to construct a solution in the favorable case. This paper shall propose a solution to this problem.

5. Sources of Unsatisfiability

We distinguish three types of network unsatisfiability.

Type 1: If the algebraic domain of the nodes is insufficiently large, then there are too few values to satisfy the network relations.

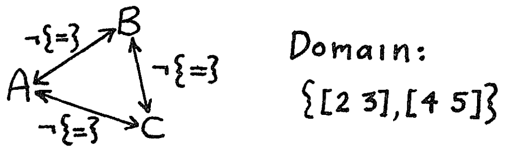

Figure 2: Insufficient Domain

For example, the network relations in Figure 2 require that all intervals be distinct, but the intervals' domain has only two members. This type of inconsistency depends on both the network relations and the algebraic domain.

In the literature on the interval representation, the usual (unstated) assumption is that the domain of possible values for intervals is large enough so that inconsistencies of type 1 do not arise. In practical applications it may be otherwise.

Type 2: To state the second type of inconsistency, we first present a theorem proved in [Valdes-Perez 86]:

> **Theorem 1:** If an interval network is closed and non-null, then '=' is a member of the composition of the relations along any loop in the network.

Therefore, if there is a loop for which the ordering '=' is *not* a member, then the network is unsatisfiable, whether unclosed or already null. Figure 3 on the following page illustrates such a loop, which we call an absurdity. Traversing this loop yields the contradiction that an interval is less than itself. This type of unsatisfiability is exactly what is detected by the TCA; its nature is characterized by Theorem 1.

[3] Our notion of closed differs from that in [Vilain & Kautz 86], for whom 'closed' corresponds to our meaning of 'minimal.' Our use is consistent with that of [Montanari 74] and [Valdes-Perez 86].

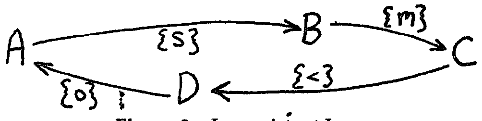

Figure 3: Inconsistent Loop

Type 3: The third and final type of unsatisfiability is that which remains after a network is closed. We interpret this type as follows. When attempting a labelling of the network, an already labelled subnetwork may require a label L1 for an edge E to avoid a loop contradiction of type 2. Another subnetwork may require a different label L2 for E for the same reason. This situation makes the network unsatisfiable, but is disguised in the closed disjunctive network by the relation {L1 L2 ...} for E.

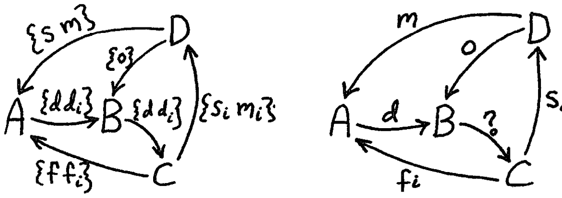
Figure 4: A Closed Unsatisfiable Network

An unsatisfiable closed network from Figure 5 in [Allen 83] is shown on the left of Figure 4. The attempt at labelling in Figure 4 on the right stalls, because either available label for the edge BC causes an absurdity at ABCA or BCDB.

6. The Closure of a Singleton is Minimal

A singleton is an interval network having a single label for each edge. Since a singleton uses the same transitivity algebra as above, and similarly reconciles the relations obtained through different paths by set intersection, there is no reason to expect that its closure is minimal. However, a closed singleton is indeed minimal, a fact needed for our satisfiability algorithm below.

> **Theorem 2:** A closed non-null interval network having a single disjunct at each edge is satisfiable. In the solution, each edge is labelled with its single ordering.[4]

We note that the purpose of the theorem is not to suggest using the TCA to find the satisfiability of a singleton; this is done more efficiently by separating intervals into their endpoints and translating the interval orderings into precedences and coincidences between these endpoints. Solving the result is quadratic in the number of intervals.

Theorem 2 shows that the difficulty of determining satisfiability arises from *disjunction*, not strictly from the vocabulary of interval orderings nor from the transitivity table. Hence, a less expressive but still disjunctive representation may nevertheless possess type-3 inconsistencies that remain undetected by the closure. One way to reduce expressiveness and eliminate type-3 inconsistency is discussed below in section 8.

7. A Remark on the Transitive Closure

Before presenting the algorithm to construct a solution, if one exists, of a general disjunctive interval network, we need the following observation.

> **Observation 1:** The transitive closure algorithm at each step examines only some three nodes i,j,k and their joining edges (i.e. a triangle). This step replaces the relation ik by reconciling it with the relation ij • jk obtained transitively. Therefore, if all of the n(n-1)(n-2)/3! triangles of a network are stable, in the sense that the mentioned replacement does not change the existing relation ik, then the network is already closed.

This fact is used by our algorithm below as an iteration invariant; at a certain step, the current labelled subnetwork is always minimal.

8. Current Approaches

Given the exponential nature of the satisfiability problem, [Vilain & Kautz 86] lists several options. One option is to limit the problem to small (sub)networks, which could be done hierarchically, as in [Allen 83]. However, the resulting subnetworks still need to be solved efficiently. [Vilain & Kautz 86] discusses other problems with hierarchization.

A second option is to resign oneself to not knowing whether a given network has a solution. One can still compute new, possibly invalid, relations through transitivity, and be content with detecting inconsistencies of type 2.

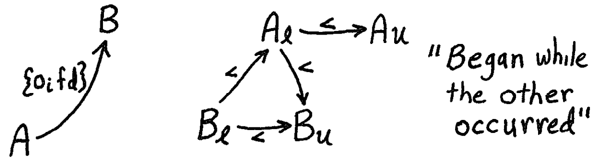

Figure 5: Interval Relation = Endpoint Relations

A third way is to trade off some expressiveness for a gain in tractability, in the style of [Brachman & Levesque 84]. Consider that there are 2^{13}-1 possible non-null relations between two intervals, of which merely thirteen are non-disjunctive. However, [Vilain & Kautz 86] points out that some of the disjunctive relations are expressed without disjunction by simple precedences among endpoints, as shown in Figure 5.

It is easy to enumerate systematically the disjunctive relations that are nondisjunctive in the corresponding endpoint network. We may then use these relations as our representation language, and formulate coherent linguistic interpretations of them. Evidently this language is considerably less expressive than the full disjunctive interval relations; the gain is a quadratic execution time, via the search for certain cycles, versus an exponential.

In the remainder of this paper we introduce an alternative algorithm that tests satisfiability by constructing a solution, does not sacrifice expressiveness, and is intended for practical use.

9. A Satisfiability Algorithm

The algorithm shown on the next page terminates and reports correctly either a consistent labelling of the network or unsatisfiability.[5] The asymptotic complexity remains, of course, exponential; the gain in practice arises from quick pruning and clever backtracking.

The algorithm was conceived using the theorems presented earlier as insight; the theorems also justify several of the steps. The search framework is a variant of the dependency-directed backtracking (DDB) introduced in [Stallman & Sussman 77] and further developed in [Steele 80].

[4] This theorem is also proved in [Valdes-Perez 86].

[5] As is usual, type-1 unsatisfiability is disregarded, meaning that the algebraic domain of the intervals is assumed large enough so that the intervals of any consistent network can be assigned values that fulfill the network relations. One such domain is the positive real numbers.

```
;;; A Constructive Satisfiability Algorithm
;;; read 'btl' as 'backtracklist'
Totally order in O the graph's edges; an edge
nearer the tail of O is more recent.[6]
∀e∈edges: btl(e) ← {}.
E←∅.  ;;; let the first edge in O succeed ∅
while ∃ an edge E' succeeding E in O begin
  E←E'.
  label E with its first candidate label.
  while ∃ an absurd triangle (E,e_i,e_j) ;;; TEST
        or[7] ∃ a nogood NG that is a subset
           of the current labelled network begin
    case absurd triangle : btl(E)←btl(E)∪{e_i,e_j}
         nogood NG : btl(E)←btl(E)∪NG-{E}.
    while there is no next candidate for E begin
      If btl(E) is empty then return(Failure).
      Assert btl(E) as a nogood.  ;;; ASSERT
      E_r←most recent edge in btl(E).
      btl(E_r)←btl(E)-{E_r} ∪ btl(E_r).
      ∀e∈edges: if e was labelled after E_r do
         unlabel(e).
         btl(e)←{}.
         reset the next candidates for e to
          its original set.
      E←E_r.
    end.
    label E with its next candidate.
  end.
end.
return(Success).
```

Abstractly, the algorithm proceeds by repeatedly selecting an edge E and testing its edge labels; backtracking - to choices made before E - is done only when no label for E is consistent with the currently labelled subnetwork.

A key aspect of the [Stallman & Sussman 77] approach to DDB is the use of nogoods. A nogood, depicted either as a list or as the negation of a conjunction (NAND), is a set of choices at choice-points that *cannot* be jointly present in any solution. The purpose of a nogood is therefore to enable abandonment of a search path as fruitless. Nogoods are normally discovered by analyzing inconsistent states to find those choices that were jointly responsible for an inconsistency (there may be several). Nogoods can also be derived by the resolution rule of inference of propositional logic [Nilsson 80], as explained in the Appendix. Our algorithm needs to save only nogoods created by resolution, for reasons discussed below.

Each edge E has a backtracklist that makes available backtrack destinations whenever the candidate labels at E are exhausted. backtracklist collects those edge-labels less recent than E that were jointly contradictory with an edge-label for E.[8]

Each time that an edge-label at E fails, before another label is tried for E, the **case** statement updates the backtracklist btl[E]. Each time that an edge-label at E fails, and there is no other label for E, the search backtracks to the most recent edge E_r in btl[E], and updates btl[E_r] by adding to it btl[E] - minus E_r itself. If E_r has no more candidates, then backtracking recurs, which explains the guard of the most deeply nested **while**.

[6]We assume the order that results from starting with a complete graph of three nodes, then at each step forming the complete graph of n+1 nodes by adding n edges from an arbitrary new node. The desired order is the order in which the edges are introduced.

[7]We assume in further discussion that the first clause is evaluated first.

[8]The joint contradiction could also have involved edges more recent than E, but they would already have been discarded by backtracking to E.

10. Properties

The algorithm is theoretically interesting because it is conducted entirely within the original interval network representation; it makes no use, for example, of endpoint graphs.

The clean separation between type-2 and type-3 inconsistencies in the algorithm is remarkable. Clause 1 of the TEST in the 2nd **while** handles type 2, by intercepting any potential triangular absurdity, two edges of which are then recorded in the backtracklist for the edge. The second clause of TEST encounters those contradictions already catalogued at ASSERT, which we examine next.

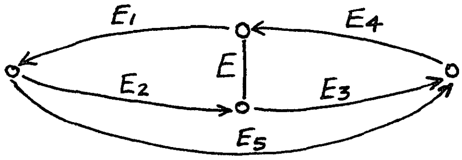

Figure 6: Type-3 Inconsistency

When the candidates at an edge are exhausted in the body of the 2nd **while**, the contents of the backtracklist are asserted as a nogood at ASSERT, as justified in the Appendix. To illustrate that this nogood represents a type-3 inconsistency, we consider the case of two candidate edge-choices at E that contradict previous choices E_1E_2 and E_3E_4, as shown in Figure 6. Our goal is to show that the loop $E_1E_2E_3E_4$ contains the '=' ordering. By assuming the contrary, and using that $E_5 \in E_2 \bullet E_3$, we deduce that there is the triangular absurdity $E_1E_5E_4$.[9] However, the first clause of step TEST intercepts all such triangles, and we arrive at a contradiction.

Theorem 1 justifies clause 1 of the TEST: no satisfiable network can forbid an interval to equal itself. Theorem 2 and Observ. 1. provided the insight that by designing an algorithm with an iteration invariant of a triangularly stable singleton network, the network is always minimal. Therefore there is no need to test the global consistency of the current labelled subnetwork.

We have used a variant of DDB in order to ensure completeness and termination. The standard DDB as described in [Stallman & Sussman 77] and [Steele 80] is apparently incomplete, because a backtrack destination is chosen arbitrarily, which does not ensure a systematic and finite traversal of the search space. In any case, our algorithm could instead use this DDB, by sacrificing completeness for the efficiency, during backtracking, of not resetting those choice-points more recent than the backtrack destination.

11. Extensions

We are currently examining an extension of the interval algebra and our algorithm to architectural layout [Baykan & Fox 87]. The objects to be laid in this application are two-dimensional rectangles, so that binary constraints between objects are expressed as a *pair* of interval orderings. The same problem of satisfiability of a completed layout plan arises here. Some differences are, for example, the desire to incorporate ternary and higher constraints into the satisfiability tester. Ternary constraints are not expressible in a network, but they are easily integrated into our algorithm in clause 1 of TEST, which checks all triangles about to be completed. Another change is needed because architects prefer to generate all solutions, if feasible, in order to let the practicing architect choose from among them.

[9]E_5 was already labelled, because the algorithm constructs a complete network before adding a new node, as specified by the order we imposed on the edges.

12. Conclusion

This paper has examined a simple but noteworthy knowledge representation language used in AI, explicated assumptions that underlie it, characterized its properties, proved several theorems concerning it where few had existed, and used these theorems and observations to design an algorithm that finds the satisfiability of a set of assertions in the language. Our work follows the analytical approach of others that have systematically characterized domains such as inheritance systems [Touretzky 86] and frames [Brachman & Levesque 84].

Acknowledgments

The author thanks Mark Derthick, Oren Etzioni, and the reviewers for helpful comments on drafts of this paper, and Dave Touretzky for suggesting the final form of the satisfiability algorithm; responsibility for errors remains with the author. He also thanks Danny Sleator and the MIT Hardware Troubleshooting group headed by Randall Davis for many fruitful discussions.

Appendix. Propositional Resolution

From the first two propositions in:

$$\neg A \vee P_1,\ A \vee P_2 \ \rightarrow P_1 \vee P_2$$

one can infer the third.

Imagine a search problem in which some choice-point CP has available only the choices A and B. Then any solution to the problem must include A or B. At a certain point, suppose that choice A is tried, which proves inconsistent with C, and then choice B is tried and proves inconsistent with D. The statement of the feasibility of CP, and the two nogoods, look like this:

$$A \vee B$$
$$\neg A \vee \neg C$$
$$\neg B \vee \neg D$$

from which follows the proposition and nogood $\neg C \vee \neg D$. Therefore, while trying the choices c_k at a choice-point, the union of the k nogoods obtained, minus the c_k elements, is itself a legitimate nogood.[10]

[10] [Nilsson 80] describes resolution in detail. Resolution of nogoods is mentioned in [Steele 80]; it and the search regimen in this paper was also used in [Valdes-Perez 87] and is more fully explained there.

References

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

J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 1983, *26(11)*, 832-843.

J.F. Allen and J.A. Koomen. Planning using a temporal world model. *Proceedings of IJCAI-8*, pages 741-747, 1983.

C.A. Baykan and M.S. Fox. An investigation of opportunistic constraint satisfaction in space planning. To appear in IJCAI Proceedings, 1987.

R.J. Brachman and H.J. Levesque. The tractability of subsumption in frame-based description languages. *Proceedings of IJCAI-84*, pages 34-37, 1984.

R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 1980, *14*, 263-313.

A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 1977, *8*, 99-118.

U. Montanari. Networks of constraints: fundamental properties and applications to picture processing. *Information Sciences*, 1974, *7*, 95-132.

N.J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, 1980.

R.M. Stallman and G.J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 1977, *9*, 135-196.

G.L. Steele Jr. *The Definition and Implementation of a Computer Programming Language Based on Constraints.* Ph.D. thesis, Massachusetts Institute of Technology, 1980.

D.S. Touretzky. *The Mathematics of Inheritance Systems.* Morgan Kaufmann Publishers, 1986.

R.E. Valdes-Perez. *Spatio-Temporal Reasoning and Linear Inequalities.* Memo 875, MIT Artificial Intelligence Laboratory, 1986.

R.E. Valdes-Perez. Knowledge-Based Schematics Drafting: Aesthetic Configuration as a Design Task. MIT AI Lab Working Paper 292, 1987.

M.B. Vilain. A system for reasoning about time. *Proceedings of AAAI*, pages 197-201, 1982.

M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. *Proceedings of AAAI*, pages 377-382, 1986.