Artificial Intelligence

AN INTERNATIONAL JOURNAL

Subscription Information

Artificial Intelligence (ISSN 0004-3702) is published in three volumes (9 issues) a year.
The subscription price for 1987 (Volumes 31–33) is Dfl. 690.00 + Dfl. 75.00 p.p.h. = Dfl. 765.00.
Our p.p.h. (postage, packing and handling) charge includes surface delivery of all issues, except to subscribers in the following countries where air delivery (S.A.L. - Surface Air Lifted) is ensured: Argentina, Australia, Brazil, Canada, China, Hong Kong, India, Israel, Japan, Malaysia, Mexico, New Zealand, Pakistan, Singapore, South Africa, South Korea, Taiwan, Thailand, U.S.A. For the rest of the world airmail charges are available upon request.
Reduced rates for individuals are available under certain conditions; for more information write directly to the Publisher.
Claims for missing issues will be honoured free of charge within three months after the publication date of the issues. Mail orders and inquiries to:
Elsevier Science Publishers B.V., Journals Department, P.O. Box 211, 1000 AE Amsterdam, The Netherlands.

Abstracted in

ABSTRACTS OF COMPUTER LITERATURE
BEHAVIOR AND PHYSIOLOGY INDEX
BEHAVIORAL SCIENCES
COMPUTER ABSTRACTS
COMPUTING REVIEWS
COMPUTER AND CONTROL ABSTRACTS
CYBERNETICS ABSTRACTS
ELECTRONICS AND COMMUNICATIONS ABSTRACTS
ENGINEERING INDEX
INFORMATION PROCESSING JOURNAL
LANGUAGE AND LANGUAGE BEHAVIOR ABSTRACTS
MATHEMATICAL REVIEWS
REFERATIVNYĬ ŽURNAL MATEMATIKA
U.S.S.R. Abstracts Journal, Series "Mathematics"
ZENTRALBLATT FÜR MATHEMATIK

Published bimonthly 0004-3702/87/$3.50 Printed in The Netherlands

Constraint Propagation with Interval Labels

Ernest Davis

Courant Institute of the Mathematical Sciences, New York, NY 10012, U.S.A.

Recommended by Johan de Kleer and Ken Forbus

ABSTRACT

Constraint propagation is often used in AI systems to perform inference about quantities. This paper studies one particular kind of constraint propagation, where quantities are labelled with signs or with intervals, and these labels are propagated through recorded constraints. We review the uses of such inference schemes in AI systems of various kinds, and evaluate their strengths and weaknesses. In particular, we determine the completeness and running time of constraint propagation for various kinds of labels and constraints.

1. Quantity Knowledge Bases

Much of the knowledge in many AI reasoning systems can be expressed as mathematical relations on real-valued quantities. For example, temporal reasoning involves relations on times and durations; spatial reasoning involves relations on lengths and angles; physical reasoning involves relations on masses, temperatures, energies, etc. Frequently, the maintenance of these relationships and the performing of inferences on them can be separated into a distinct module called a *quantity knowledge base*. A quantity knowledge base interacts with the rest of the system's inference mechanism in two ways. The system provides the quantity knowledge base with mathematical relations, either perceived or derived from nonmathematical inferences; and the quantity knowledge base uses these to infer new relations which it can report to the rest of the system.

The problems of combinatorial explosion and search, which plague inference engines generally, take particularly fierce form in quantitative domains. Starting with any quantitative constraint, it is possible to deduce innumerable others. For instance, starting with the constraint $0 < a < b$, we can infer further constraints such as $a < \sqrt{ab} < \frac{1}{2}(a+b) < b$; $e^a < e^b$; $\int_0^a e^{-x^2}\,dx < \int_0^b e^{-x^2}\,dx$ .... Moreover, it is known that extremely complex and convoluted paths of inference are sometimes needed to draw desired conclusions from

input constraints, even if both the conclusions and constraints have simple forms. Thus, an effective quantity knowledge base must exert strong control to prevent the inference process from searching a large or infinite space, and, in so doing, it must generally give up some degree of completeness. The designer of the knowledge base must manage this tradeoff so that the knowledge base produces useful information in reasonable time.

One approach to this problem has been to implement the knowledge base as a *constraint network* which performs inference by *propagating interval labels*. This architecture has been used, with varying degrees of success, in physical reasoning, temporal reasoning, and spatial reasoning. This paper discusses how constraint networks are used in these domains; how the efficiency and effectiveness of constraint propagation varies with the types of constraints and labels involved; and how the basic architecture of constraint networks may be extended to achieve greater power.

2. Constraint Networks

A constraint network is a declarative structure which expresses relations among parameters. It consists of a number of *nodes* connected by *constraints*. A node represents an individual parameter, which has some particular *value*, known or unknown. A constraint represents a relation among the values of the nodes it connects. Typically, each constraint is connected to only a few nodes. For example, in Waltz's program for interpreting line drawings, each node is an image element; its values are the various possible interpretations; and constraints are rules establishing coherence of interpretation [1]. In MOLGEN [2], nodes are elements of an genetic experiment, such as bacteria, chromosomal elements, or antibiotics, whose value must be specified in designing the experiment, and constraints are limitations on the ways these can be chosen, given the objective of the experiment. In an electronics program, nodes are voltages and currents, and the constraints are physical laws that relate them [3]. Constraints which are universally applicable across all nodes of a given type or structure within a given domain, such as Kirchhoff's current and voltage laws in an electronic system, may sometimes be left *implicit*, imbedded in code rather than explicitly represented by a data structure.

Forward inference on constraint networks, called *assimilation*, is generally done using *constraint propagation*, shown in Algorithm 2.1. In constraint propagation, information is deduced from a local group of constraints and nodes, and is recorded as a change in the network. Further deductions will make use of these changes to make further changes. Thus, the consequences of each datum gradually spread throughout the network.

Algorithm 2.1. Constraint propagation:

repeat
- take some small group of constraints and nodes in some connected section of the network,
- update the information in this section of the network, given the information in the constraints and the nodes;

until no more updating occurs (the network is *quiescent*) or some other termination condition is reached.

Constraint networks may also support *query answering*, where the user may ask for the value of a term, or the truth-value of a proposition, based on the state of the network. Generally, constraint networks are designed so that most of the inferencing is done in assimilation, and queries can be answered quickly from the final state of the network. Query answering generally does not modify the state of the network.

All constraint propagation systems share a number of valuable properties:

(1) Constraint propagation consists of a simple control structure applied to a simple updating module. Hence it is easy to code, to analyze, and to extend. It is easy to understand its actions, to explain them to a user, and to monitor them by some external intelligent module.

(2) It degrades well under time limitations; interrupting the process in the middle gives useful information already deduced.

(3) It is easily implemented in parallel, since updating can be performed all over the network simultaneously. Except in constraint inference systems, it is possible to implement each constraint as a single processor, which updates its arguments whenever they are changed, or at regular intervals.

(4) It is well suited to incremental systems. A constraint may be added incrementally by incorporating it in the network, updating its arguments, and propagating the effects.

(5) The concept of a constraint network as consisting of a collection of local connections works well with similar locality assumptions made in AI systems. For example, physical reasoning programs assume that physical effects propagate across connections between components. Hence, constraints on physical values will relate only the parameters of a few closely connected components [4].

We may distinguish six different categories of constraint propagation based on the type of information which is updated.

- In *constraint inference*, new constraints are inferred and added to the constraint network. Kuipers' ENV [5], Simmons' Quantity Lattice [6], and Brooks' CMS [7] are all constraint inference systems (see also [8–10]).
- In *label inference*, each node is labelled with a set of possible values. In assimilation, the constraints are used to restrict these sets. Label inference is the subject of this paper.
- In *value inference*, nodes are labelled with constant values, and the constraints are used to find values of unlabelled nodes from labelled nodes. SKETCHPAD [11] and THINGLAB [12] use value inference.

– *Expression inference* is a generalization of value inference, in which nodes may be labelled with a value expressed as terms over the values of other nodes. When a node is given two different labels, these are equated, and the resultant equation solved. Expression inference is used in CONSTRAINTS [13].

– In *relaxation*, all nodes are given exact values, which may not be consistent with the constraints. The assimilation process "pushes" these values around, so as to satisfy each constraint. Relaxation has been used extensively outside AI to solve problems such as partial differential equations [14].

– In *relaxation labelling*, nodes are assigned with assignments of probabilities to various values. Updating involves changing probabilities by combining the previous probability on the node with the probabilities indicated by the other nodes [15].

Constraint propagation systems may be applied in a *wholesale* or *incremental* fashion. In a wholesale system, all input constraints are available at the beginning of processing, and are fixed thereafter. In an incremental system, accepting new constraints alternates with answering queries. (In algorithm theory, these are called "off-line" and "on-line" systems, respectively.)

There are two important differences between quantitative and nonquantitative constraint propagation. Firstly, the range of a nonquantity variable is usually a fairly small finite set. Therefore, when constraint inference provides no further pruning, one can always resort to exhaustive search using generate-and-test: pick a possible value at random, and determine the consequences of your choice. Quantities have an infinite range of possible values. Thus, exhaustive search is generally infeasible, and the power of generate-and-test much diminished.[1] Secondly, the ranges of nonquantity variables are often sets with little or no internal structure; sets of possible labels are generally designated by enumeration of elements. In contrast the real line has a lot of internal structure, and all kinds of unary predicates are available for our use.

Each of the six categories of constraint propagation has particular weaknesses and limitations. Relaxation and relaxation labelling are not sound deductive techniques; the labels derived are not logically related to the previous labels. Therefore, they are difficult to analyze, and difficult to interface with general inference systems. Value inference and expression inference can only be used if the constraints are equations; they cannot be used with inequalities. Constraint inference and label inference are hard to control; it may be difficult to prevent them from going into infinite loops. In constraint inference, it may be difficult to insure that new constraints derived will be useful in query answering. The appeal of label inference is that it is deductively sound, it can be used with

[1] Generate-and-test is used in a number of quantitative applications: either in the context of Monte Carlo search, as in SPAM [16] and MERCATOR [17]; or where the real line is broken into a finite number of subintervals as in ENVISION [3]; or where the constraints involved have a disjunctive form, and one disjunct must be chosen for exploration, as in CONSTRAINTS [13].

constraints of arbitrary form, and that it is much easier to control than general constraint inference.

3. Label Inference

In label inference systems, each node is labelled with the set of its possible values. In quantitative applications, this set is generally an interval on the real line. In query answering, the term to be evaluated is expressed as a function of some of the nodes, and the system must calculate the range of value of the term given the label sets of the component nodes. In assimilation, node values are updated through *label refinement*, in which the label set of one node is restricted, based on a constraint and on the labels of all the other nodes in the constraint. For example, if we have the constraint "$X + Y = Z$," and we have the label sets "$X \geq 3$, $Y = 1$," then we can deduce that $Z \geq 4$, and add this to the description of the label set of Z. The general form of refinement can be expressed in the following definition:

Definition 3.1. Let C be a constraint on nodes $X_1, \ldots, X_k$. Let S_i be the label set for X_i. Then

$$\text{REFINE}(C, X_j) = \{a_j \in S_j \mid \exists(a_i \in S_i,\ i = 1, \ldots, k,\ i \neq j)\ C(a_1, \ldots, a_j, \ldots, a_k)\}\ .$$

That is, REFINE(C, X_j) is the set of values for X_j which is consistent with the constraint C and with all the labels S_i. A value a_j is in REFINE(C, X_j) if a_j is in S_j and it is part of some k-tuple $a_1, \ldots, a_k$ which satisfies C and all the S_i. Note that refinement is sound deductively; if a tuple satisfies the constraint and the starting labels, then it satisfies the refined label.

Applying the updating function REFINE within the constraint propagation control structure (Algorithm 2.1) gives the Waltz algorithm, first presented in [1]. The Waltz algorithm entails using refinement on each constraint and each node over and over until refinement produces no more changes. When this stage is reached, the network is said to have *reached quiescence*. Since refinement is sound deductively, so is the Waltz algorithm, which simply iterates refinement. That is, if a given assignment of values to the quantities satisfies all the constraints and all the starting labels, then it will satisfy the labels calculated by the Waltz algorithm. If the algorithm halts with assigning some parameter the null set, then the input state was inconsistent. Algorithm 3.2 is an efficient implementation of the Waltz algorithm. Example 3.3 shows the Waltz algorithm in action.

Algorithm 3.2.
/* The set S_i is the current label set of quantity X_i. */

/* REVISE refines all the parameters $X_1, \ldots, X_k$ of a given constraint C, and returns the set of all parameters whose set was changed. */

```
procedure REVISE(C(X_1, ..., X_k))
  begin CHANGED ← ∅
    for each argument X_i do
      begin S ← REFINE(C, X_i)
        if S = ∅ then halt /* the original constraints were inconsistent */
        else if S ≠ S_i then
          begin S_i ← S;
            add X_i to CHANGED
          end
      end
    return CHANGED
  end

procedure WALTZ
  begin Q ← a queue of all constraints
    while Q ≠ ∅ do
      begin remove constraints C from Q;
        CHANGED ← REVISE(C)
        for each X_i in CHANGED do
          for each constraint C' ≠ C which has X_i in its domain do
            add C' to Q
      end
  end
```

Example 3.3 (Executing the Waltz algorithm). Suppose we start with the following bounds

$$x \in [1, 10], \qquad y \in [3, 8], \qquad z \in [2, 7]$$

and relations

$$x + y = z, \qquad y \leq x.$$

This would be implemented in a data structure like the one shown in Fig. 1.

The algorithm would proceed as follows: The constraint queue would begin with both constraints (CON1, CON2) on it.

CON1 $(x + y = z)$ is popped from the queue.
- Since $x \geq 1$ and $y \geq 3$, CON1 gives $z \geq 4$, so reset the bounds of z to $[4, 7]$.
- Since $z \leq 7$ and $y \geq 3$, CON1 gives $x \leq 4$, so reset the bounds of x to $[1, 4]$.

Since x and z have been changed, add CON2 to the queue.

CON2 $(y \leq x)$ is popped from the queue.
- Since $x \leq 4$, CON2 gives $y \leq 4$, so reset the bounds of y to $[3, 4]$.
- Since $y \geq 3$, CON2 gives $x \geq 3$, so reset the bounds of x to be $[3, 4]$.

Since x and y have changed, add CON1 to the queue.

CON1 $(x + y = z)$ is popped from the queue.
- Since $x \geq 3$, $y \geq 3$, CON1 gives $z \geq 6$, so reset the bounds of z to $[6, 7]$.

Since only z has been changed and z has no other constraints besides CON1, nothing is added to the queue.

Since the queue is empty, quit.

Algorithm 3.2 may be modified to handle the addition of a single constraint by changing the first line of procedure WALTZ to read, "$Q \leftarrow$ a queue containing the new constraint."

The running time of the Waltz algorithm has been extensively analyzed in the case where the range of the parameter is a finite set [18–20]. In particular, Mackworth and Freuder have shown that the algorithm halts after $O(ae)$ calls to REVISE where a is the number of possible values per parameter, and e is the number of constraints. However, this halting result does not apply when the set of possible labels is infinite, as is often the case with quantitative labels. In such cases, the analysis depends critically on the nature of the constraints involved.

The restriction that all information pass from the constraints to the queries via the label sets makes it almost certain that some information will be lost. For example, if we start with two nodes A and B labelled "$A \in \{1, 2, 3\}$" and "$B \in \{2, 3, 4\}$" and the constraint "$A = B$," then the Waltz algorithm will deduce the new labels "$A \in \{2, 3\}$," "$B \in \{2, 3\}$," and stop. If we now pose the query "What is the value of the vector $\langle A, B \rangle$," the system will answer that there are four possibilities: $\langle 2, 2 \rangle$, $\langle 2, 3 \rangle$, $\langle 3, 2 \rangle$, and $\langle 3, 3 \rangle$, since these are all consistent with the labels, though not with the constraint. Similarly, if we query the system "Is A equal to B," the best answer we can get is "Possibly yes, or possibly not."

One may view this information in the following geometrical terms. The set of tuples of values which are consistent with the constraints and the starting labels define some volume in k-space (k is the number of parameters). Label inference finds the projection of this volume onto each separate quantity axis.

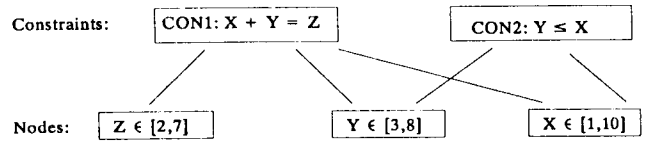

FIG. 1. Constraint network.

Any query operation which uses just the labels is considering the entire cross-product of all these projections. Thus, the effect of making the labels intermediaries between assimilation and query answering is to replace the constraints by a "box" in k-space which circumscribes them. The difference between the volume defined by the constraints and the circumscribing box corresponds to information lost.

It is sometimes possible to avoid this information loss by using nodes corresponding to complex terms. For instance, the above problem can be fixed by adding a node $A - B$ labelled 0. If the query answerer now consults this new labelling, it can answer both of the above questions accurately. In the geometrical view of the previous paragraph, introducing these terms corresponds to transforming the axes, so that boxes in the new axes fit the constraints. However, this approach increases the complexity of both assimilation and query answering (see Section 10).

An alternative approach, pursued in an early version of SPAM [21] is to use labels such that all selections of values from the labels satisfy the constraints. (McDermott calls these "conservative" labels.) That is, the cross-product of the labels is a subset of the solution set of the constraints, rather than being a superset, as in systems which use refinement: an inscribed box, rather than a circumscribed box. In the above example, we would use either the set of labels $A \in \{2\}$, $B \in \{2\}$, or the set of labels $A \in \{3\}$, $B \in \{3\}$. If we had the starting labels $A \in \{1,2,3\}$, $B \in \{2,3,4\}$, and the constraint $A \neq B$ then there would be four possibilities for reasonable conservative labellings: $A \in \{1\}$, $B \in \{2,3,4\}$; $A \in \{1,2\}$, $B \in \{3,4\}$; $A \in \{1,3\}$, $B \in \{2,4\}$; and $A \in \{1,2,3\}$, $B \in \{4\}$. Boggess [22] uses an extreme form of this approach in which each quantity is given a single value such that the quantities satisfy the constraints. Thus, in the second example, she might use the labellings $A = 1$, $B = 4$ or $A = 3$, $B = 2$, etc.

This approach, however, is problematic. It is not deductively sound; that is, it permits inferences which are not warranted by the constraints and which consequently could be mistaken. It is not unique; there are many different sets of conservative labels. Finally, it is much harder to implement than a refinement algorithm, since there is no analogue of the "REFINE" operator which allows one variable to be set at a time. Rather, all variables must be considered together.

For instance, if, as above, we start with the labels $A \in \{1,2,3\}$, $B \in \{2,3,4\}$, and we add the constraint $A \neq B$, then we must choose between one of the four conservative labellings mentioned above. Finding these conservative labelling involves assigning A and B simultaneously; they cannot be considered separately. None of these labellings is a logically necessary consequence of the constraint; any of them could turn out later to be wrong. For example, if we choose $A \in \{1,2\}$, $B \in \{3,4\}$, then we arbitrarily rule out the consistent possibility $A = 3$, $B = 2$. The next constraint added may contradict

our labels, even if it is consistent with the first constraint; in our example, the constraint $A = 3$ would violate the labelling we have chosen. If this happens, the system must backtrack, and choose some other set of conservative values. Typically, contradictions of this kind will be frequent, and will involve many previous choices; hence backtracking will be extensive and expensive.

Label inference techniques can rarely be complete, because of the "narrow bandwidth" of labels as an interface between the constraints and the queries. A more modest requirement is that assimilation and query answering are each separately complete. Assimilation is complete if the label assigned to a node represents accurately the range of values it can attain given the constraints. That is, if assimilation is complete, then we can consistently assign to any node any value within its label, and then can pick values for all the other nodes so that all constraints are satisfied. If the actual range of values is a set which is not described by any label in the label language, then we can say that assimilation is complete up to the label language if the labels assigned are the best labels possible. By definition, the refinement operation is always complete for a single constraint; however, as we shall see, this does not imply that the Waltz algorithm is complete for a set of constraints.

Query answering is complete if the range of values returned for the queried term is exactly the range permitted by the label sets. That is, if we query the system for the value of term T, and it returns "T is in set S," then, for any value V in S, we may assign values $V_1, \ldots V_n$ to the nodes so that all implicit constraints are satisfied, and so that, if the nodes have these values, then T will have value V. For example, suppose we have nodes A, B, and $A - B$, with labels $\{2,3\}$, $\{2,3\}$ and $\{0\}$, respectively, as above, and we query the system whether $A = B$. A query system that looks only at the first two labels will answer "Possibly true; possibly false;" a system which looks at the third label will say "Definitely true." The first system is incomplete, while the second is complete.

The basic characteristics of a label inference system are the constraint language, the query language, the query answering language, the kinds of terms used as nodes, and the kinds of predicates used as labels. Once these are fixed, programming and evaluating the assimilator involves the following considerations:

- How is an input constraint expressed as a constraint on node terms?
- How is refinement performed on explicit constraints?
- What implicit constraints are used? How is refinement performed on them?
- Under what circumstances does the Waltz algorithm reach quiescence? If it does not, how is it halted?
- In what order does the Waltz algorithm choose constraints and nodes to refine?
- Under what circumstances are nodes introduced and removed from the system?

- Is the assimilator complete?

Designing the query answerer involves the following problems:

- How is a query expressed as a function of the terms?
- How is a query term evaluated?
- Is the query answerer complete?

4. Applications of Labels Inference

In this section, we discuss the use of label inference in three particular programs: de Kleer and Brown's ENVISION [3], Dean's TMM [23], and McDermott's SPAM [16]. We will not, by any means, try to summarize these programs in their totality; our presentation is limited to their application of label inference.

4.1. ENVISION

ENVISION [3] performs qualitative reasoning about the behavior of physical systems over time. Given the description of a physical system in physical terms, ENVISION first creates a mathematical model of the system, by identifying a number of physical quantities as determining the state of the system, and by identifying constraints which relate these quantities and their time derivatives. It then uses these constraints to predict the behavior of the system over time. ENVISION uses constraint propagation and label inference to enforce the physical constraints at any given instant of time. To predict the behavior of the system over extended time, it solves *qualitative differential equations*, which are beyond the scope of this paper.

ENVISION is largely concerned with the response of systems to small changes in one system parameter. It characterizes these changes, and the resultant changes in other system quantities, purely in terms of their sign without any indication of magnitude. Therefore, nodes represent quantity changes, and they are labelled with one of four signs: POS = $(0, \infty)$, NEG = $(-\infty, 0)$, ZERO = $[0, 0]$, and IND = $(-\infty, \infty)$.

Consider, for example, the simple electronic circuit shown in Fig. 2. The

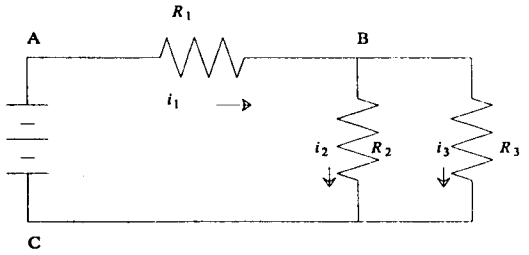

FIG. 2. Electronic circuit.

resistances of the resistors are assumed to be positive, but otherwise unspecified.

ENVISION begins by identifying seven state variables: V_{AC}, the voltage drop from A to C; V_{BC}, the voltage drop from B to C; V_{AB}, the voltage drop from A to B; i_1, the current through R_1; i_2, the current through R_2; i_3, the current through R_3; and i_4, the current through the battery. These quantities are related by Kirchhoff's voltage law (the sum of the voltage drops around a cycle is zero), Kirchhoff's current law (the total current flowing into any node is zero), and Ohm's law (the current through a resistor is proportional to the voltage across it). Symbolically, we can express these laws as follows:

Kirchhoff's voltage law:
$$-V_{AC} + V_{AB} + V_{BC} = 0 .$$

Kirchhoff's current law:
$$i_1 - i_2 - i_3 = 0 , \qquad i_2 + i_3 - i_4 = 0 , \qquad i_4 = i_1 .$$

Ohm's law:
$$V_{AB} = R_1 i_1 , \qquad V_{BC} = R_2 i_2 , \qquad V_{BC} = R_3 i_3 .$$

Since ENVISION is primarily interested in changes to these quantities, rather than the quantities themselves, we differentiate these rules. For any quantity Q, let $\delta(Q)$ represent an infinitesimal change to Q. Then, differentiating the above rules, we get:

Kirchhoff's voltage law:
$$\delta(-V_{AC}) + \delta(V_{AB}) + \delta(V_{BC}) = 0 .$$

Kirchhoff's current law:
$$\delta(i_1) - \delta(i_2) - \delta(i_3) = 0 , \qquad \delta(i_2) + \delta(i_3) - \delta(i_4) = 0 , \qquad \delta(i_4) = \delta(i_1) .$$

Ohm's law:
$$\delta(V_{AB}) = R_1 \delta(i_1) , \qquad \delta(V_{BC}) = R_2 \delta(i_2) , \qquad \delta(V_{BC}) = R_3 \delta(i_3) .$$

Finally, since ENVISION only uses the signs of these changes, the above equations are discretized to be relations on signs. Following de Kleer, we will denote the sign of a quantity Q as $[Q]$, and the sign of the change of Q as ∂Q; thus $\partial Q = [\delta(Q)]$. The last four equations above all have the form $\delta(P) = \alpha\delta(Q)$, where α is positive. Thus $\delta(P)$ and $\delta(Q)$ have the same sign, and $\partial P = \partial Q$. The first three equations all have the form $\delta(P) + \delta(R) + \delta(Q) = 0$. This implies that either all these derivatives are zero, or at least one is positive, and at least one is negative. Thus, the sum of the corresponding intervals must either be $[0, 0]$ or $(-\infty, \infty)$. We may write this $\partial P + \partial R + \partial Q \supseteq \text{ZERO}$.[2] The resultant constraints are as follows:

[2] De Kleer [3] and others use the notation, "$\partial P + \partial Q + \partial R = 0$."

$$-\partial V_{AC} + \partial V_{AB} + \partial V_{BC} \supseteq 0 . \quad \text{(C1)}$$

$$\partial i_1 - \partial i_2 - \partial i_3 \supseteq 0 . \quad \text{(C2)}$$

$$\partial i_2 + \partial i_3 - \partial i_4 \supseteq 0 . \quad \text{(C3)}$$

$$\partial i_1 = \partial i_4 . \quad \text{(C4)}$$

$$\partial V_{AB} = \partial i_1 . \quad \text{(C5)}$$

$$\partial V_{BC} = \partial i_2 . \quad \text{(C6)}$$

$$\partial V_{BC} = \partial i_3 . \quad \text{(C7)}$$

In practice, ENVISION does not go through the steps of differentiating and discretizing equations. Rather, physical laws are stated in discretized differential form, so that ENVISION goes directly from a physical description to a final form. The original form of the equations is therefore irrelevant; all that matters is the discretized form. Thus, ENVISION can handle nonlinear and linear systems with equal ease.

Given a change in one or more system parameters, ENVISION can use label propagation around these constraints to infer changes in others. Suppose that the current through R_2 increases; that is, $\partial i_2 = \text{POS}$. From (C6), we can deduce that $\partial V_{BC} = \text{POS}$; that is, V_{BC} is increasing. From (C7), we can deduce that $\partial i_3 = \text{POS}$; that is, i_3 is increasing. From (C2), we can now deduce that $\partial i_1 = \text{POS}$. From either (C3) or (C4), we can deduce that $\partial i_4 = \text{POS}$. From (C5), we can deduce that $\partial V_{AB} = \text{POS}$. From (C1), we can deduce that $\partial V_{AC} = \text{POS}$. Thus, we can determine that, if i_2 increases, so do all the other system parameters.

Sometimes constraint propagation by itself is not adequate to make these predictions. For example, suppose we specify that $\partial V_{AC} = \text{POS}$, the voltage across the battery is increasing. The only constraint associated with V_{AC} is (C1), and (C1) is not sufficiently powerful to allow us to propagate any labels. It implies that either ∂V_{BC} or ∂V_{AB} must be positive, but there is no way to tell which. In this case, ENVISION uses a "generate-and-test" strategy. It guesses a particular value for ∂V_{BC}. First, try $\partial V_{BC} = \text{NEG}$. From (C1) we can deduce $\partial V_{AB} = \text{POS}$. From (C5), we can deduce $\partial i_1 = \text{POS}$. From (C6), we can deduce $\partial i_2 = \text{NEG}$. From (C7), we can deduce $\partial i_3 = \text{NEG}$. But these contradict constraint (C2): ∂i_1, $-\partial i_2$, $-\partial i_3$ are all positive. Thus, the guess that $\partial V_{BC} = \text{NEG}$ must have been wrong. A similar contradiction is reached if we try $\partial V_{BC} = \text{ZERO}$. The only possibility is that $V_{BC} = \text{POS}$. Making this assumption allows us to propagate labels quickly to deduce that all quantities increase.

4.2. TMM

TMM [23] maintains a *time map*, a knowledge structure which records changes

to the world over time. The input to the program consists of a number of *time tokens*, denoting particular instants of interest; assertions that a particular event takes place, or a particular fact is true, at some particular time token, or between two particular time tokens; rules that describe necessary relations among events and facts; and constraints, which relate the times at which the various time tokens take place. The main concern of TMM is to keep track of how facts become true and false over time; however, we focus entirely on its use of temporal constraints.

Temporal constraints in TMM are all interval bounds on the time difference between two tokens. For example, if the token "light(gas)" and "boiled(eggs)" represent the time when the gas is lit under a pot of eggs, and the time when the eggs are hard boiled, respectively, then the fact, "The eggs will take from 10 to 15 minutes to boil," would be expressed, "boiled(eggs) − light(gas) ∈ [10, 15]." (The units involved—minutes in this case—are constant throughout the program.) TMM maintains a constant network where all pairs of differences are maintained as nodes, with interval labels.[3] The constraints on these nodes are all implicit, determined by the form of the node quantities. If N_1 is a node representing $a - b$, N_2 is a node representing $b - c$, and N_3 is a node representing $a - c$, where a, b, and c are tokens, then N_1, N_2, and N_3 are related by the implicit constraint "$N_1 + N_2 = N_3$." TMM performs inference on an input constraint by setting the label for the node involved, and then propagating the effects of this new label through these implicit constraints. TMM also allows constraints to be deleted, and their consequences to be withdrawn (see Section 11).

For example, suppose we are given the following facts. Dinner preparations last night consisted of shopping, followed by a break, followed by cooking. Shopping took between an hour and an hour and a half; the break took between half an hour and an hour; cooking took between one and a quarter and two hours; and the whole preparation time took between two and three hours.

TMM would analyze this using four time tokens: B, the time when preparations were begun; S, the time when shopping was completed, C, the time when cooking began, and D, the time when dinner was ready. The input constraints are therefore

$$S - B \in [1.0, 1.5] , \qquad C - S \in [0.5, 1.0] , \qquad D - C \in [1.25, 2.0] ,$$
$$D - B \in [2.0, 3.0] .$$

The constraint network has six nodes with labels derived from the input constraints, and four implicit constraints.

[3] Strictly speaking, TMM records all the differences only between the most important tokens, called the kernel. Tokens outside the kernel are more weakly bound. See Section 10.

Nodes:

$$N_1 = S - B \in [1.0, 1.5], \quad N_2 = C - B \in (-\infty, \infty),$$
$$N_3 = D - B \in [2.0, 3.0], \quad N_4 = C - S \in [0.5, 1.0],$$
$$N_5 = D - S \in (-\infty, \infty), \quad N_6 = D - C \in [1.25, 2.0].$$

Constraints:

$$N_1 + N_4 = N_2, \quad \text{(C1)}$$
$$N_1 + N_5 = N_3, \quad \text{(C2)}$$
$$N_2 + N_6 = N_3, \quad \text{(C3)}$$
$$N_4 + N_6 = N_5. \quad \text{(C4)}$$

(In general, for n quantities, there will be $\binom{n}{2}$ nodes and $\binom{n}{3}$ implicit constraints.)

We now propagate these labels through these constraints.

- From (C1), $N_1 \in [1.0, 1.5]$, $N_4 \in [0.5, 1.0]$, deduce $N_2 \in [1.5, 2.5]$.
- From (C2), $N_1 \in [1.0, 1.5]$, $N_3 \in [2.0, 3.0]$, deduce $N_5 \in [0.5, 2.0]$.
- From (C3), $N_2 \in [1.5, 2.5]$, $N_6 \in [1.25, 2.0]$, $N_3 \in [2.0, 3.0]$, deduce $N_3 \in [2.75, 3.0]$, $N_2 \in [1.5, 1.75]$, $N_6 \in [1.25, 1.5]$.
- From (C4), $N_4 \in [0.5, 1.0]$, $N_6 \in [1.25, 1.5]$, $N_5 \in [0.5, 2.0]$, deduce $N_4 \in [0.5, 0.75]$, $N_5 \in [1.5, 2.0]$.
- From (C1), $N_1 \in [1.0, 1.5]$, $N_4 \in [0.5, 0.75]$, $N_2 \in [1.5, 1.75]$, deduce $N_1 \in [1.0, 1.25]$.
- No more deductions can be made.

4.3. SPAM

SPAM [16] maintains an incremental knowledge base for information about the relative positions of objects. The program modifies the knowledge base in response to user input constraints, and consults the knowledge base to answer user queries. It handles a broad range of constraints and queries.

SPAM associates a Cartesian reference frame with each object. Each reference frame has its own unit scale, its own orientation, and its own origin. The knowledge base of SPAM consists of a constraint network whose nodes denote four basic binary relationships between frames, A and B: their relative scale, $s(B, A)$; their relative orientation, $o(B, A)$; and the x- and y-coordinates of the origin of B in the frame of A, $x(B, A)$ and $y(B, A)$. These nodes are labelled with interval bounds. Nodes are constrained by implicit constraints, dependent on the geometry of the terms involved, as well as by user input constraints.

For example, Fig. 3 shows three frames, A, B, and C. The program might choose to represent the relations between these frames in terms of the

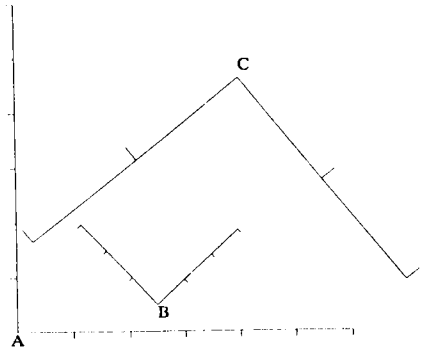

FIG. 3. Three frames of reference.

following nodes and labels:

$$x(B, A) \in [2.0, 3.0], \quad y(B, A) \in [0.0, 1.0],$$
$$s(B, A) \in [0.5, 1.0], \quad o(B, A) \in [30°, 90°];$$
$$x(C, A) \in [0.0, 6.0], \quad y(C, A) \in [4.0, 5.0],$$
$$s(C, A) \in [2.0, 3.0], \quad o(C, A) \in [180°, 270°].$$

When SPAM is asked a query, it first transforms the query to be a function over the node terms, and then evaluates this function using either Monte Carlo or hill-climbing techniques. For example, if the system is queried, "How far is C from B in terms of the units of B?" this expression is transformed to

$$\frac{[(x(C, A) - x(B, A))^2 + (y(C, A) - y(B, A))^2]^{1/2}}{s(B, A)}.$$

It then calculates that, given the bounds on the quantities, the value of this expression lies somewhere in the range [3.0, 12.8].

When SPAM is given a new constraint to assimilate, it first transforms it to a constraint over node terms, as above, and then uses a hill-climbing technique to perform refinement on the nodes involved. For example, suppose that we start with the constraint network as above, and we input the constraint, "C is at least twelve B units from B." This is translated into the constraint

$$[(x(C, A) - x(B, A))^2 + (y(C, A) - y(B, A))^2]^{1/2} \geq 12s(B, A).$$

The program then determines the maximum and minimum values of each of

the node terms involved, given this constraint and the starting labels. It arrives at the results

$$x(B, A) \in [2.0, 2.6], \quad y(B, A) \in [0.0, 0.5], \quad s(B, A) \in [0.5, 0.53],$$
$$x(C, A) \in [5.4, 6.0], \quad y(C, A) \in [4.5, 5.0].$$

Note that if the previous query, "How far is B from C in B units?" is raised after this assimilation, the answer will be "In the range [9.2, 12.8]." Thus, information has been lost. The assimilation program estimates the degree of loss by determining how frequently tuples chosen at random within the refined labels satisfy the input constraint. If the information loss is unacceptably high, the program may choose to rectify this introducing new nodes, and, possibly, new frames (see Section 10).

SPAM does not use the Waltz algorithm. Rather, refinement is applied to each constrant when it is entered, and then the constraint is forgotten. This is due to time limitations; since we are using a hill-climber to perform refinement, each refinement takes so long that propagation would be wholly unworkable.

5. Label Languages

We may now proceed to a more general analysis. As stated at the end of Section 3, there are five essential determinants of a label inference system: the types of node terms, the language of labels, the language of constraints, the language of queries, and the language of answers. However, it turns out that the languages of labels and constraints are key for categorization. (For the present, we will assume that all nodes are simple quantities; we will consider more complex nodes in Section 10. The language of queries is almost always equal either to the label language or to the constraint language. The language of query answers is typically the same as the language of term labels.) We will therefore begin by separately considering varieties of label language and constraint language, and then describe how they combine.

There are two main label languages in use: signs and intervals. Sign labelling uses the four labels POS, NEG, ZERO, and IND.[4] This crude categorization is often sufficient for particular applications, particularly, as in ENVISION, in categorizing directions of change.

Interval labelling uses arbitrary intervals as labels.[5] It is used in spaces where

[4] Strictly speaking, we should, perhaps, include the labels NONNEG(x) ($x \geq 0$), NONPOS(x) ($x \leq 0$) and NONZERO(x) ($x \neq 0$). This would allow refinements such as NONZERO(xy) implies NONZERO(x), NONZERO(y). However, the slight increase in completeness does not seem to be worth the substantial increase in complexity.

[5] Throughout this paper we use the notation $T \in [a, b]$. Note that this may be taken as an abbreviation for $a \leq T \leq b$. In describing the first-order language of these systems, it is not necessary either to use set theory or to use intervals as individuals.

sign labelling is too coarse a measurement, or which have no absolute zero. Intervals have a number of valuable properties as a language of labels. Intervals are flexible enough to represent both an approximate value and a degree of uncertainty. An interval can be represented by two real numbers and two booleans (to indicate closedness or openness at each end.) All connected sets of real numbers are intervals, and vice versa. Hence, the image of an interval, or a collection of intervals, under a continuous function is itself an interval. Likewise, the intersection of two intervals is an interval. Finally, it is generally reasonably easy to work out the refinement operator for intervals in particular cases.

Three questions are often raised with respect to intervals. The first is whether to use open, closed, or half-open intervals. Clearly, for maximal flexibility, one would want to use all of these. However, this usually complicates the case analysis for little practical gain, and it is better to stick either to closed intervals or to open intervals. As between closed and open intervals, closed intervals are probably preferable, for three reasons. Firstly, given any bounded set of values, there is a unique minimal closed set which contains it; hence, a best labelling. Secondly, exact values are a special case of closed intervals, so exact values and interval values can be handled uniformly. However, the programmer who wishes to exploit this fact must be somewhat careful. He will find that code which is correct and works perfectly well for intervals of finite length may bomb on point intervals because of round-off error. Finally, the image of a closed bounded interval under a continuous function is a closed bounded interval; this is not true of open intervals. However, this advantage may be lost, since most systems also use intervals which are infinite on the right or the left. Closed intervals are also typically used in interval analysis [24].

The second question about intervals relates to the treatment of disconnected sets. One may find that the set of values consistent with a constraint is disconnected, because the constraint makes one quantity involved either a discontinuous or a multi-valued function of the other quantities. For example, given the constraint $xy = z$, and the labels $y \in [-1, 1]$, $z \in [4, 5]$, the solution set for x is $x \in (-\infty, -4] \cup [4, \infty)$. The problem is that $x = z/y$ is a discontinuous function of y at $y = 0$. For another example, given the constraint $x^2 = z$ and the label $z \in [4, 9]$, the solution set for x is $x \in [-3, -2] \cup [2, 3]$. Here, the problem is that the function $x = \sqrt{z}$ is multi-valued.

There would seem to be two possible approaches. The first is to use sets of intervals, rather than just single intervals. The second is to use the convex hull of the solution set; that is, the smallest interval containing the entire solution set. I have never heard of any program using the first approach. The simplicity of using a single interval would apparently outweigh the occasional loss of information involved. It seems unlikely that this would ever be the major source of information loss in a program of this kind.

The final question regarding intervals is that of psychological plausibility. Is

it reasonable to treat quantitative knowledge as a step function, to say that one is sure that the length of a stick is between 5.11 and 6.34 feet, and nothing more? Do not one's beliefs correspond more to a probability distribution with a likelihood gradually diminishing on either side of a central value? There is something plausible about these arguments; unfortunately, the alternatives to intervals are worse. Probability functions require more arbitrary assumptions and introducing more fudge numbers than simple intervals; their semantics are less clear; and, if done right, they require much more complex computations. Representing uncertain values by (say) a Gaussian probability curve is no less arbitrary than representing them by fixed bounds; and it is very difficult to define what the probability means and how to combine different curves. My feeling is that it is better to pick some safe bounds, to do computations with them, and then to introduce probability at the end by treating the results with a grain of salt. If the computation tells you that the length of the stick is between 5.11 and 6.34 feet long, this should be read as "About 5.7 ± 0.6." If some rule requires that the stick be at most 6.5 feet long, the system should not consider that the condition has been met with any great certainty.

6. Constraint Languages

There are eight major classes of constraint languages which have emerged so far as important in AI systems. In increasing order of complexity these are:

(a) unary predicates;
(b) order languages, consisting simply of the order relationship;
(c) systems of equations of the form "$x - y > c$" or "$x - y \geq c$";
(d) linear equations and inequalities with unit coefficients (i.e. all coefficients are -1, 0, or 1);
(e) linear equations and inequalities with arbitrary coefficients;
(f) boolean combinations of constraints;
(g) algebraic equations;
(h) transcendental equations.

In this section, we will discuss the AI applications of each of these.

The simplest kind of constraints are unary predicates, i.e. labels. We have already looked at two possible classes of unary predicates: signs and intervals. Other predicates which are often used include exact values and fixed ranges. Exact values are used in systems where quantities are known without uncertainty. This was the practice in old style blocks-world programs, such as SHRDLU [25], which represented the positions of blocks by the exact coordinates of the front lower left-hand corner. Fixed ranges are used in measure spaces which are naturally divided into a finite set of exhaustive, disjoint intervals. For example, the measure space "water temperature" is naturally divided into the ranges $(-\infty, 32)$ (ice), $[32, 32]$ (melting), $(32, 212)$ (liquid), $[212, 212]$ (boiling), $(212, \infty)$ (gas). The space of voltages across a transistor is

divided into the cutoff region, the linear region, and the saturation region. Such divisions are extensively used in naive physics systems [3, 26].

Order relations arise in systems where comparative magnitudes are the only quantitative relations of interest. This occurs, for example, in systems which plan the order of events without consideration of their time duration, such as NOAH [27]. Each level in planning in NOAH specifies a partial ordering on the actions in that level. Order relations are also a large part of the quantitative information used in physical reasoning systems [3, 5, 26].[6]

Inequalities of the form $x - y \geq c$ are useful in dealing with quantity spaces in which the relative values of two quantities may be known well though their absolute quantities are much more uncertain. This happens particularly in trying to place events on a timeline. For example, one might know that Michaelangelo was an older contemporary of Raphael, and that the end of the Thirty Years War occurred about thirty years after its beginning, while having only the vaguest notion of either the dates of these events, or the relation of the first pair to the second pair. Such knowledge can be represented in equations like the following:

$$\text{date(birth(Raphael))} - \text{date(birth(Michaelangelo))} \in [0, 30],$$

$$\text{date(end(30\_years\_war))} - \text{date(start(30\_years\_war))} \in [25, 35],$$

$$\text{date(birth(Raphael))} - \text{date(start(30\_years\_war))} \in [-100, 200],$$

$$\text{date(birth(Michaelangelo))} \in [1400, 1600],$$

$$\text{date(start(30\_years\_war))} \in [1500, 1700].$$

Such constraints are also used in TMM [23] and other planning programs [28].

In measure spaces with scalar multiplication, a bound on the quotient of two quantities is often more useful than a bound on their difference. For example, you might know that both your kitchen table and your house were roughly 2 to 1 rectangles without knowing at all precisely the ratio of their sizes. You might then express your knowledge in the inequalities:

$$\text{length\_of(table)}/\text{width\_of(table)} \in [1.8, 2.2],$$

$$\text{length\_of(house)}/\text{width\_of(house)} \in [1.5, 2.5],$$

$$\text{length\_of(house)}/\text{length\_of(table)} \in [10.0, 50.0].$$

A system of bounds on quotients of positive quantities is isomorphic to a system of bounds on differences; the logarithm function is the isomorphism.

[6] Note that any constraint on specific quantities derived from the monotonicity of functions, as in [5, 26], is always a simple order relationship. The actual derivation of these relations is outside the scope of this paper, since I am not dealing with constraints on functions.

Allen and Kautz [50] used a constraint network with both quotient bounds and boolean combinations of order relations for temporal reasoning.

The class of linear relations with unit coefficients is important in common sense reasoning, because such relations are adequate to express conservation laws. Conservation laws assert that the change in a value over time is equal to the sum of its increments minus the sum of its decrements. For instance, the change in a bank account is the sum of the deposits plus the interest minus the sum of the withdrawals. The change of the amount of liquid in a cup is the total amount poured in minus the total spillage and evaporation. If we are only interested in reasoning about discrete changes, and we can identify all relevant increments and decrements, then we can write any such rule in the form

$$Q_f - Q_i = \sum_k \mathrm{Inc}_k - \sum_k \mathrm{Dec}_k .$$

Linear inequalities with nonunit coefficients may arise from combining ratio information with difference information. For example, suppose that I know that painting the table will take somewhat longer than painting the chair, I intend to start the table about ten minutes after finishing the chair, and I want the whole process to take under two hours. Using T_1, T_2, T_3, and T_4 for the times when I start the chair, end the chair, start the table, and end the table, respectively, I can express these constraints in the linear inequalities,

$$\frac{T_4 - T_3}{T_2 - T_1} \in [2, 3], \qquad T_3 - T_2 \in [8, 12], \qquad T_4 - T_1 < 120 .$$

General linear inequalities also arise in analyzing the differential behavior of nonlinear systems. For example, in studying small perturbations to an ideal gas, which obeys the law $PV = nkT$ (P = pressure, V = volume, T = temperature, n and k are constants), we may use the differential rule $V\,\mathrm{d}P/\mathrm{d}t + P\,\mathrm{d}V/\mathrm{d}t = nk\,\mathrm{d}T/\mathrm{d}t$. For small perturbations from a starting state, V and P may be treated as constants. ENVISION [3] uses such relations. General linear inequalities have also been used in analyzing the stability of a tower of polyhedral blocks [29], and in geometric reasoning on polygons and polyhedra [8]. (Note that, though Malik and Binford [30], talk about arbitrary linear inequalities, all of the examples given are either absolute bounds, order relations, or bounded differences.)

Boolean combinations of constraints have been most extensively studied in physical reasoning [3, 26] in electronic circuit design [13], and in planning. They take three major forms. Firstly, there are gated constraints of the form "if P then Q," where P and Q are constraints. These arise in electronics and in physics in considering systems which have several states, each with different rules, such as the state of a transistor, or the phase of a quantity of material.

Secondly, there are reductions of complex constraints. As we have seen in ENVISION, when a linear equation of the form $X + Y + Z = 0$ is expressed as a relation over the signs of X, Y and Z, it takes the form "Either $X = Y = Z = 0$ or at least one of them is positive and at least one is negative." In fact, if we start with any relationship on quantities, and we discretize the quantities to a finite set of possibilities, then the relationship reduces to some table of allowable tuples, and can be expressed as a boolean combination of simple constraints. Thirdly, in planning, a very common constraint is that the actor can only do one thing at a time. This can be expressed as the disjunction, "Either the end of ACT1 comes before the beginning of ACT2 or vice versa." Likewise, the relations on time intervals used by Allen [31] are essentially boolean combinations of order relations on their endpoints, as shown by Vilain and Kautz [32].

Physical reasoning may involve nonlinear constraints. Some of these arise from the geometric properties of physical systems; others have their origin in the basic physics. For example, Williams [33] uses multiplicative equations in analyzing MOS transistors.

Geometric reasoning generally involves both algebraic and transcendental equations, since distance function is algebraic, and the trigonometric functions are transcendental. (For the purposes of complexity analysis, these transcendental functions can be eliminated by replacing constraints on angles with equivalent algebraic constraints on their sines and cosines.) The exponential and logarithmic functions have been unimportant in AI systems.

7. Complexity Results

In evaluating the effectiveness of label inference for solving a network of constraints of a particular type, it is important to know how difficult, in principle, is the solution of such constraints. For each type, there are three problems to consider: (i) the problem of evaluating the bounds on a term of a particular kind, given labels on its arguments; (ii) the problem of performing refinement on a constraint of a given kind; and (iii) the problem of solving a set of such constraints.

((i) and (ii)) Evaluating order relationships, linear sums, or boolean combinations of these, on labels is straightforward. Evaluating an algebraic expression on a set of labels is NP-hard, even if the expression is no more than fourth-order. (Yemini [34] shows that solving a set of equations of the form $(x_i - x_j)^2 + (y_i - y_j)^2 = c_{ij}$ is NP-hard. This problem can be reduced to the problem of whether 0 is in the range of $\Sigma_{i,j}[(x_i - x_j)^2 + (y_i - y_j)^2 - c_{ij}]^2$.) However, the bounds on an algebraic expression can be computed in exponential space, and hence doubly exponential time [35]. Evaluating a transcendental function is, in the worst case, uncomputable. (It is uncomputable whether a

given transcendental expression is identically equal to zero [36].) The same bounds apply to performing refinement, except for boolean combinations. Since boolean combinations include arbitrary conjunctions, solving a single boolean combination is equivalent to solving a system of them (see below).

(iii) A set of order relationships or bounds on differences may be solved in time n^3 (see [32] and Appendix A). A set of linear relations can be solved using Karmakar's algorithm in time $O(n^{3.5}E^2)$, where n is the number of variables and E is the size of the problem [37]. Solving a set of boolean combinations of constraints is in general NP-complete, even if all the constraints have the form "$X=0$" or "$X=1$," unless the boolean combinations have very restricted form. (This is just the satisfiability problem [38].) It should be noted in particular that solving a system of constraints of the form $[X] \pm [Y] \pm [Z] \geq 0$ is NP-complete. Thus, discretizing a set of linear equations to a constraint on signs has the effect of changing a computationally tractable problem to an intractable one. Similarly, Vilain and Kautz [32] have shown that solving the interval relations used by Allen [31] is NP-complete. Solving a set of algebraic relations is NP-hard, and solving a set of transcendental equations is uncomputable, as discussed above.

8. Propagating Signs

We may now evaluate the power of a variety of label inference systems, organizing them by label language and constraint language. We will first consider the simpler label language of signs and then the language of interval bounds.

For the purposes of sign computation, any function or constraint can be reduced to a finite table or relation over the sign of its arguments. All constraint languages thus look pretty much the same to a network with sign labels. Figure 4 gives the tables for some common functions and relations.

Complex arithmetic expressions can be evaluated by combining evaluation rules. For instance, if POS(x), POS(y), NEG(z), and NEG(w) then we can evaluate the expression $xy + wz$ by first evaluating the products as positive and then evaluating the sum as positive. They can be refined by evaluating all other parts of the relation and applying the refinement rules hierarchically. For example, given the relation $xy + wx(t + z) = -6$, we can evaluate xy as POS and -6 as NEG, giving $wx(t + z)$ as NEG. We next evaluate wx as NEG, giving $t + z$ as POS. Finally, we use sum refinement to deduce that, since z is NEG, t is POS.

When sign labels are used, the Waltz algorithm is guaranteed to quiesce quickly. Since each quantity can change its value only once (from IND to a sign) in the course of execution, it can only once be responsible for putting a constraint on the constraint queue. The number of times a single constraint is

Order relations

Evaluation: $X < Y$ if [NEG(X) and (ZERO(Y) or POS(Y))] or [POS(Y) and (ZERO(X) or NEG(X))]

Refinement: $X < Y$ implies:
if (NEG(Y) or ZERO(Y)), then NEG(X)
if (POS(X) or ZERO(X)), then POS(Y)

Arithmetic expressions

+	NEG	ZERO	POS	IND
NEG	NEG	NEG	IND	IND
ZERO	NEG	ZERO	POS	IND
POS	IND	POS	POS	IND
IND	IND	IND	IND	IND

×	NEG	ZERO	POS	IND
NEG	POS	ZERO	NEG	IND
ZERO	ZERO	ZERO	ZERO	ZERO
POS	NEG	ZERO	POS	IND
IND	IND	ZERO	IND	IND

X	NEG	ZERO	POS	IND
$-X$	POS	ZERO	NEG	IND

X	NEG	ZERO	POS	IND
$1/X$	NEG	UND	POS	IND

Constraint: $X - Y \geq C$

$C > 0$: either POS(X) or NEG(Y)
$C = 0$: either POS(X) or NEG(Y) or (ZERO(X) and ZERO(Y))
$C < 0$: no constraint on the signs of X or Y.

FIG. 4. Sign tables for simple functions and relations.

put on the queue is thus L, the number of variables in the constraint. Summing over all constraints, the total number of times that some constraint is put on the queue is E, the size of the constraint set, the sum of the lengths of the constraints. The maximal number of calls to REFINE is thus EL.

If, as is often the case, a refinement rule can be applied only when only one parameter is unknown, there is a particularly efficient implementation. With each constraint, keep a count field of the number of unknown variables, which is updated each time a variable is set, and maintain a data structure which allows quick access to those constraints with count field 1. Always choose a constraint with count field 1 to refine; if there are none, then terminate. In this way, you never look at any constraint twice with the idea of refinement. Thus, quiescence is reached after e calls to REFINE, where e is the number of constraints. The algorithm runs in time $O(E)$. (This is similar to the use of count fields in [39].)

Propagation of sign labels is not complete overall for any interesting class of constraints. It is complete for assimilation for pure order constraints, but not for any more complex class of constraints. Nor is this surprising; the language of signs is simply too crude to capture much of what is going on in constraints

of this kind. All that is available to a reasoner which uses signs are the tables of sign relationships implied by the constraints, as discussed above. The most we could expect is that the algorithm might be complete for the sign tables of some interesting class of constraints. It is, in fact, complete for the tables of bounded difference constraints. That is, given a collection of constraints on signs of the form in Fig. 4, and a number of starting sign labels, the Waltz algorithm gives the correct range possible to each quantity. For any more complex system of constraints, the assimilation problem is NP-complete, so a quick algorithm like the Waltz algorithm has no chance of being complete. As mentioned, ENVISION [3] gets around this incompleteness using "generate-and-test" techniques, in which particular signs are assigned to indeterminate quantities, and constraint propagation is used to check whether these assignments lead to contradiction. However, this technique can be exponentially explosive when applied to large systems.

9. Propagating Intervals

The properties of label inference vary widely across different kinds of constraints. Throughout the following discussion, n represents the number of quantities, e, the number of constraints, and E, the total size of the constraint system.

If the constraints are order relationships, then the Waltz algorithm is complete for assimilation. Note that, when the high bound of a variable is reset, its new value is one of the original high bounds on one of the other quantities, and likewise for the lower bound. Thus, each variable can take on at most $2n$ different labels over the course of propagation. Therefore, by the theorem of Mackworth and Freuder [18], the running time of the Waltz algorithm is $O(ne)$. Cases can be constructed where this worst case is achieved.

Bounds on quantity differences, of the form $x - y \in [a, b]$, are easily analyzed, because the problem can be mapped onto the well-known problem of finding a minimum-cost path in a directed graph. The analysis is carried out in Appendix A. The major results are as follows: If we use a network with nodes of the form $x - y$, then the Waltz algorithm is complete for the whole inference process (assimilation together with query answering). Moreover, if we perform refinement in the proper order, then, for consistent sets of constraints, the system reaches quiescence in time $O(n^3)$. If we use a network with nodes for quantities, rather than their differences, then the Waltz algorithm is complete for assimilation, though not for inference as a whole. For consistent starting states, if constraints are chosen in the right order, the Waltz algorithm terminates in time $O(n^3)$. If all the constraints and labels have a positive upper bound and a negative lower bound, then constraints can be chosen in an order which gives convergence in time $O(n^2)$. In either case, if the starting state is inconsistent, then the system will either detect the inconsistency by finding a lower bound greater than an upper bound, or go into an infinite loop. Such infinite loops can be detected by monitoring the system and halting it when it has performed more than the maximum number of refinements needed for a consistent system. With either differences as nodes or simple quantities as nodes, there are pathological cases in which choosing nodes in a poor order causes the system to do exponential number of refinements before it quiesces. It is therefore important in these systems to choose the order of refinements carefully.

The analysis of unit coefficient constraints is substantially more complicated, and is carried out in Appendix B. The results may be summarized as follows. Like systems of bounded differences, propagation around unit coefficient constraints necessarily quiesces if the starting state is consistent. If we use either a FIFO queue for constraints in the propagation algorithm, or we choose constraints to refine in a fixed sequential order, then the Waltz algorithm will quiesce in time $O(nE)$, where n is the number of variables and E is the size of the constraint system (the sum of the lengths of all the constraints). On the other hand, if constraints are chosen in a poor order, then the Waltz algorithm may take exponential time to quiesce, and if the starting state is inconsistent, then it may go into an infinite loop. Unlike systems of bounded differences, however, the Waltz algorithm is not complete for assimilation.

Once we get past unit coefficient inequalities, to arbitrary linear relations, or nonlinear relations, the Waltz algorithm starts to break down. Not only is it incomplete; it tends to go into infinite loops even for well-behaved sets of constraints. Consider, for example, the simple pair of equations $\{x = y, x = 2y\}$ with the starting ranges $\{x \in [0, 100], y \in [0, 100]\}$. The system is consistent, with the unique solution $x = y = 0$. However, the Waltz algorithm goes into an infinite loop. We begin by deducing from the second equation that $y \in [0, 50]$; then, from the first equation, that $x \in [0, 50]$; then that $y \in [0, 25]$; then that $x \in [0, 25]$; . . . Indeed, for practically any system of arbitrary linear relations, or nonlinear relations, there are starting labellings which go into infinite loops in this way.

Therefore, some rule other than quiescence must be chosen to terminate constraint propagation. The termination criterion can be a simple time limit, or a cutoff on the number of times any single constraint is invoked, or a cutoff on the amount of change that is considered significant. (Ultimately, the machine will enforce this last cutoff by itself, when the change goes below floating point number accuracy.)

The analysis of unit coefficient constraints in Appendix B suggests some heuristics that may be useful even for nonlinear constraints. There is reason to believe that, in general, it is a good heuristic strategy to choose constraints off the queue of Algorithm 2.1 in FIFO order, or in fixed sequential order, rather than, say, LIFO order or best-first order. The intuitive justification for this rule is that, in any situation, there will generally be a few best refinements to apply,

and FIFO or fixed order ensures that these will be applied reasonably soon. The analysis of Appendix B makes this more precise. It also supports a somewhat stronger heuristic: that if we have been n times through the queue and have not reached a quiescent state, then we should probably quit, since we are likely to be in a very long or infinite loop.

The evaluation of complex terms and the refinement of complex constraints are issues in themselves. It is always possible to evaluate complex terms by applying each operator in sequence, but this may give bounds which are much too large. For example, to evaluate the range of the term $(X+1)/X$, given the label $X \in [1,2]$, we can apply the operators in order as follows:

$$\frac{[1,2]+1}{[1,2]} = \frac{[2,3]}{[1,2]} = [1,3].$$

However, by simplifying the expression to $1 + 1/X$, we can see that the true bounds are $[1.5, 2.0]$. In constraint refinement, even crude techniques like this are lacking.

Closed-form formulas exist for all linear constraints, and for particularly simple nonlinear constraints (see Tables 1 and 2). Where closed-form formulas cannot be obtained, a variety of techniques can be applied. The theory of interval arithmetic provides a number of formulas which may be useful for terms and constraints of special forms [24]. Complex terms may be evaluated

TABLE 1. Refinement over linear constraints

Constraint: $\sum_{i \in PC} c_i x_i - \sum_{i \in NC} c_i x_i \in [p, q]$.

/* PC and NC are, respectively the sets of indices with positive and negative coefficients. The c_i are assumed to be positive. */

Labels: $x_i \in [a_i, b_i]$.

Refined labels:

For $j \in PC$, $x_j \in \left[\max\left(a_j, \frac{1}{c_j}\left(p + \sum_{i \in NC} c_i a_i - \sum_{i \in PC, i \neq j} c_i b_i\right)\right), \min\left(b_j, \frac{1}{c_j}\left(q + \sum_{i \in NC} c_i b_i - \sum_{i \in PC, i \neq j} c_i a_i\right)\right)\right]$.

For $j \in NC$, $x_j \in \left[\max\left(a_j, \frac{1}{c_j}\left(\sum_{i \in PC} c_i a_i - q - \sum_{i \in NC, i \neq j} c_i b_i\right)\right), \min\left(b_j, \frac{1}{c_j}\left(\sum_{i \in PC} c_i b_i - p - \sum_{i \in NC, i \neq j} c_i a_i\right)\right)\right]$.

TABLE 2. Refinement rule for a quadratic constraint (this rule was used in the MERCATOR system [17])

Constraint: $x^2 + yx + z = 0$.

Labels: $x \in [x_l, x_h]$, $y \in [y_l, y_h]$, $z \in [z_l, z_h]$.

Refined labels for x:

If $y_l > 0$ then let $a_l = y_l$, $a_h = y_h$,

else if $y_h < 0$ then let $a_l = -y_h$, $a_h = -y_l$,

else let $a_l = 0$, $a_h = \max(-y_l, y_h)$.

/* a_l, a_h are the lower and upper bounds on abs(y). */

Let $D_l = \max(a_l^2 - 4z_h, 0)$, $D_h = a_h^2 - 4z_l$.

If $D_h < 0$ then the system is inconsistent.

/* D_l, D_h are bounds on $y^2 - 4z$. */

Let $v_1 = \frac{1}{2}(-y_h - \sqrt{D_h})$, $v_2 = \frac{1}{2}(-y_l - \sqrt{D_l})$,
$v_3 = \frac{1}{2}(-y_h + \sqrt{D_l})$, $v_4 = \frac{1}{2}(-y_l + \sqrt{D_h})$.

If $x_l > v_2$ then $x \in [\max(x_l, v_3), \min(x_h, v_4)]$,

else if $x_h < v_3$ then $x \in [\max(x_l, v_1), \min(x_h, v_2)]$,

else $x \in [\max(x_l, v_1), \min(x_h, v_4)]$.

If this result makes the upper bound on x less than the lower bound, then the system was inconsistent.

using Monte Carlo techniques. For each quantity in the term, a value within its label is picked at random, and the term is evaluated on these particular values. This gives one possible value of the term. By iterating random choices, a range of possible values of the term is found, which may be assumed to be close to the true range of the term. Alternatively, a hill-climbing technique may be used, where the maximum and minimum values of the term are sought, within the range of the quantity labels. Hill-climbing techniques can also be used for label refinement; the maximum and minimum values of each quantity are sought, given the constraint being refined and labels on the other quantities. Techniques for building such hill-climbers are given in [40].

Neither Monte Carlo search nor hill-climbing is a sound inferential technique. They generally return a subset of the true range, and thus arbitrarily exclude legitimate possibilities. Furthermore, our experience with hill-climbers in SPAM [16] was discouraging; we found them to be slow and unreliable.

Table 3 summarizes the results of Sections 6, 7, and 8.

TABLE 3. Summary of results (n = number of variables, E = size of constraint set (sum of lengths of constraints), L = maximum length of any constraint)

	Theoretical bounds[a]	Sign propagation[b]	Interval propagation[b]
Unary predicates	Trivial	Trivial	Trivial
Order relation	$O(n^3)$	Complete: $O(E)$	Complete: $O(n^3)$
Bounded difference	$O(n^3)$	Complete: $O(E)$	Complete: $O(n^3)$
Unit coefficients	As hard as linear programming	Incomplete: $O(E)$	Incomplete: $O(En)$
Linear programming	$O(n^{3.5}E^2)$	Incomplete: $O(EL)$	May not quiesce
Algebraic relations	Doubly exponential	Incomplete: $O(EL)$	May not quiesce
Transcendental relations	Unsolvable	Incomplete: $O(EL)$	May not quiesce

[a]Times given for "bounds" are the best known times for a complete solution.
[b]Times given for sign and interval propagation are the times to reach quiescence.

10. Complex Nodes

The previous sections focused on constraint networks in which each node corresponds to a simple quantity. However, AI systems frequently use nodes which represent a complex term. As we have seen, TMM [23] uses nodes representing the difference between two dates, and SPAM [16] uses nodes representing the relative scales, orientations, and positions of two reference frames. Such complex nodes greatly increase the power of label propagation; they also complicate the problems of inference.

Complex nodes are generally used when information about the relative values of quantities is much more available than information about their absolute values. In the world of TMM, there is very little information about absolute times of events, so a node for the time of a single event would need a very wide label, and would express very little information. Likewise, in SPAM, very little is known about the parameters of objects in any absolute scale; what is known is their positions relative to one another.

To repeat the example of Section 6, suppose that one's knowledge of dates in early modern history is partially expressed in the following constraints:

$$\text{date(birth(Raphael))} - \text{date(birth(Michaelangelo))} \in [0, 30],$$
$$\text{date(end(30\_years\_war))} - \text{date(start(30\_years\_war))} \in [25, 35],$$
$$\text{date(birth(Raphael))} - \text{date(start(30\_years\_war))} \in [-100, 200],$$
$$\text{date(birth(Michaelangelo))} \in [1400, 1600],$$
$$\text{date(start(30\_years\_war))} \in [1500, 1700].$$

If we are limited to use the dates of the events themselves as nodes, then our labels will be very broad:

$$\text{date(birth(Michaelangelo))} \in [1400, 1600],$$
$$\text{date(birth(Raphael))} \in [1400, 1630],$$
$$\text{date(start(30\_years\_war))} \in [1500, 1700],$$
$$\text{date(end(30\_years\_war))} \in [1525, 1735].$$

The precise information relating the birth of Raphael to the birth of Michaelangelo, and the end of the Thirty Years War to its beginning are irrecoverably lost. If, however, we may use nodes like

$$\text{date(birth(Raphael))} - \text{date(birth(Michaelangelo))},$$
$$\text{date(end(30\_years\_war))} - \text{date(start(30\_years\_war))},$$

then the constraints may be expressed exactly as labels on the nodes.

A constraint network with complex nodes must indicate the structure of the nodes and their relation to the basic quantities. Thus, such networks have three levels: the quantities (unlabelled); the nodes, with interval labels; and the explicit constraints, connected to the nodes (Fig. 5).

The price of this flexibility is increased complexity. Unlike simple quantities, complex terms cannot be assigned values arbitrarily; their values are related by implicit constraints. For example, terms of the form $D_{ij} = x_j - x_i$ obey the constraint $D_{ij} + D_{jk} = D_{ik}$. Label inference should be carried out through these implicit constraints, as well as through the explicit constraints.

To achieve the greatest possible inferential power, we should use a *complete* set of implicit constraints: that is, a set of implicit constraints such that, if all

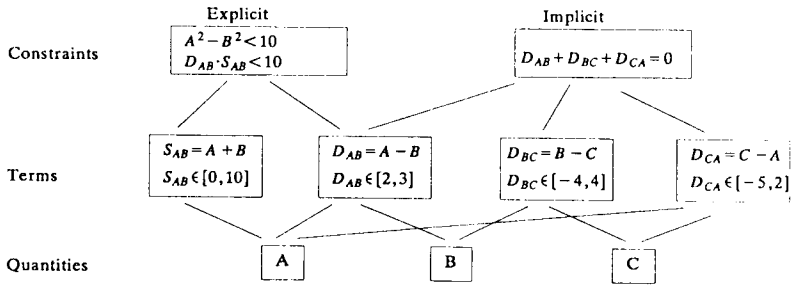

FIG. 5. Three-level constraint network.

the constraints are satisfied for a given set of term values, then that set of values is, indeed, possible for the terms. For terms $D_{ij} = x_j - x_i$, the family of constraints $D_{ij} + D_{jk} = D_{ik}$ is complete; given any numbers a_{ij} such that $a_{ij} + a_{jk} = a_{jk}$, we can find numbers b_i such that $a_{ij} = b_i - b_j$. However, finding a complete set of implicit constraints for more complex terms may be difficult or uncomputable.

A second, related problem is that explicit constraints and queried terms are dependent on complex node terms in multiple ways. Therefore, the value returned by a query depends critically on which node terms are used to compute it; and the refinements of a constraint depend on how it is construed as a relation over nodes.[7] Suppose, for example, our system has nodes for the quantities X_i, for terms of the form $D_{ij} = X_i - X_j$, and for terms of the form $S_{ij} = X_i + X_j$. We now query the system for the value of $T = X_1^2 - X_2^2$. This can be evaluated, either as a function of X_1 and X_2, as in the above expression, or as the function $S_{12}D_{12}$. Suppose we start with the labels $X_1 \in [1, 2]$, $X_2 \in [1, 2]$, $D_{12} \in [-1, 1]$, $S_{12} \in [2, 4]$. Considering the term as a function over X_1 and X_2, we can calculate that its range is $[-3, 3]$. Considering it as a function over D_{12} and S_{12}, we can calculate its range as $[-4, 4]$. Here, the X nodes give better bounds than the complex nodes. However, if the label on D_{12} were $[0, 0]$, then using $T = S_{12}D_{12}$, we can calculate $T \in [0, 0]$, while the calculation using the labels on X_1 and X_2 still gives $[-3, 3]$. Thus, which expression is better depends on the starting states of the labellings.

A final problem is that the number of possible complex nodes is much greater than the number of simple quantities. If our nodes have the form $D_{ij} = x_i - x_j$, then n quantities give rise to $\frac{1}{2}n(n-1)$ possible nodes. In general, if complex nodes are functions of k quantities, there will be $\binom{n}{k}$ possible nodes. If all of these must be represented in the system, we will waste a lot of space and inference time. There is no point in sustaining "date(IJCAI-85) − date(death(Caesar))" as a node term, and in reducing its range by a week when we get more precise information about the date of the conference. To avoid combinatorial explosion, therefore, there must be rules that specify which nodes to use.

One approach to these problems is to insist that the constraint network always use an *independent* set of nodes. A set of nodes is independent if they may all be assigned values independently; that is, given any assignment of values to the nodes, it is possible to assign corresponding values to the quantities. For example, if the quantities are W, X, Y, Z, then the nodes $D_{XW} = X - W$, $D_{YX} = Y - X$, and $D_{ZX} = Z - X$ are independent. Given any values for these nodes, we can find corresponding values for the quantities; for

[7] This problem can arise computationally with nodes which are simple labels, since it may not be clear how the algebraic expression can be simplified. (It may not even be computable.) However, in principle, the functional dependence of a term on basic quantities is unique; the problem is merely one of computing this dependence over interval labels.

example, we may choose $W = 0$, $X = D_{XW}$, $Y = D_{YX} + D_{XW}$, and $Z = D_{ZX} + D_{XW}$. Likewise, the nodes $S_{WX} = W + X$, $D_{WX} = W - X$, $S_{YZ} = Y + Z$, and $D_{YZ} = Y - Z$ are independent; given any values for the nodes, we can choose $W = \frac{1}{2}(S_{WX} + D_{WX})$, $X = \frac{1}{2}(S_{WX} - D_{WX})$, $Y = \frac{1}{2}(S_{YZ} + D_{YZ})$, $Z = \frac{1}{2}(S_{YZ} - D_{YZ})$. However, the set of nodes $D_{WX} = W - X$, $D_{XY} = X - Y$, $D_{YW} = Y - W$ is not independent, since corresponding values of W, X, Y, and Z can only be found if $D_{WX} + D_{XY} + D_{YW} = 0$.

In many cases, it will be possible to give rules for constructing independent sets of nodes. For example, if all nodes are quantity differences of the form $D_{ij} = x_i - x_j$, then the following rule hold: Consider a graph in which each quantity x_i is a vertex and each node D_{ij} is an arc. A set of nodes D_{ij} is independent just if they form a tree in the graph. To construct independent sets of spatial quantities, SPAM [16] uses the following technique: The reference frames were organized into three trees, called the "size tree," the "orientation tree," and the "position tree." The relative scale of two frames was recorded in a node of the constraint network if the frames were adjacent in the size tree; the relative orientation was recorded if the frames were adjacent in the orientation tree; the relative position was recorded if they were adjacent in the position tree. This constituted a sufficient, though not necessary, set of conditions that the set of nodes in the network was independent.

Using independent sets of nodes has several advantages. Firstly, there are no intrinsic constraints in such a system. Secondly, any term or constraint has a unique functional (relational) dependence on the nodes. Thirdly, the number of nodes in such a set is less than or equal to the number of quantities.

This simplicity, however, costs us some, though not all, of the flexibility gained by using complex nodes. The problem is that it becomes necessary to establish strict criteria for selecting the particular independent set of nodes to be used. For example, suppose we have the quantities W, X, Y, Z, and we are using the independent set of nodes $D_{XW} = X - W$, $D_{YW} = Y - W$, $D_{ZW} = Z - W$. Now, a constraint of the form $Y - X \in [l, h]$ is input. There are two possible actions. The first is to express $Y - X$ in terms of the existing nodes as $D_{YW} - D_{XW}$, and use the new constraint to refine D_{YW} and D_{XW}. However, this may lead to substantial loss of information. If we have the labels $D_{YW} \in [-3, 5]$, $D_{XW} \in [-5, 3]$, and we add the constraint $Y - X \in [-1, 1]$, then refinement only brings our labels down to $D_{YW} \in [-3, 4]$, $D_{XW} \in [-4, 3]$. If we now ask for the value of $Y - X$ from the revised constraint network, we evaluate this term as $D_{YW} - D_{XW} = [-3, 4] - [-4, 3] = [-6, 8]$. Clearly most of the information in the input constraint has been lost.

The alternative is to add the term $D_{YX} = Y - X$ as a node. In order to maintain the independence of the node set, we must now take out one of the old nodes, either D_{YW} or D_{XW}. Whichever we do, we will end up sacrificing some information, and one has to judge that the information being added is worth more than the information being deleted.

In the case above, the choice of which nodes to include is straightforward. Presumably, more information is expressed by the node D_{YX} with bounds $[-1, 1]$ than by the node D_{YW} with bounds $[-3, 4]$ or by the node D_{XW} with bounds $[-4, 3]$. Thus either D_{YW} or D_{XW} may be sacrificed with a clear conscience. In general, if a node $D_{AB} = A - B$ has the label $[L, U]$, one may associate a measure of uncertainty $L - U$ with the node. In order to reduce the degree of uncertainty in the network as a whole, one chooses to include the nodes with the smallest uncertainty. (This is implemented in SPAM [41].)

This intuitive criterion can be given formal justification as follows. We can think about the space of all tuples of quantity values. The labels on the nodes define one subset of that space, the input constraints define another. The set of values consistent with the label must contain the set of values consistent with the input constraints; otherwise, we have excluded possible solutions with no basis, and our inference process is unsound. However, we would prefer that the label set include as few values outside the constraint set as possible; such values correspond to information lost in going from the constraints to the labels. Depending on the nature of the quantities involved, it may be possible to define a metric on the space of possible sets of values. If this can be done, we can define our objective as minimizing the size of the set of values satisfying the labels, subject to the rule that it must contain the set of values consistent with the constraint. For one particularly simple metric, it can be shown that the size of the set satisfying the lales is minimized by including nodes with the smallest difference between their upper and lower bounds [41]. In complex quantity spaces, however, natural metrics may not exist, and it may be difficult to define precisely what is meant by an optimal set of nodes.

The selection of a set of nodes is further complicated by the fact that it is sometimes advantageous to introduce new "imaginary" quantities to the network. Suppose that we have the set of constraints $\{X_1 - X_0 \in [-2, 2], X_2 - X_1 \in [-2, 2], X_0 - X_2 \in [-2, 2]\}$. If we try to express these using an independent set drawn from the D_{ij}, we will inevitably lose information. If we record $D_{10} \in [-2, 2]$ and $D_{21} \in [-2, 2]$, then evaluating $X_0 - X_2 = -D_{21} - D_{10}$ will give us $[-4, 4]$. However, we can express this constraint set exactly using a spare variable Y, and using the node terms $E_i = Y - X_i$. The label set $\{E_0 \in [-1, 1], E_1 \in [-1, 1], E_2 \in [-1, 1]\}$ allows the retrieval of exactly the correct bounds on $X_i - X_j$. The quantity Y here is essentially a Skolem constant. I have not been able to analyze the problem of finding the best tree, given the ability to add imaginary quantities.

In short, the use of an independent set generally involves some loss of information, and often involves substantial loss of information. If we drop the requirement of an independent set, then we need not lose information, but we must confront the problems of choosing which nodes are worth keeping, and of choosing a representation for constraints and terms. Since the representation is not unique, the choice of the best representation generally involves some

search at run-time. Approaches to these problems are likely to be highly domain-dependent; however, the following general observations may be useful:

The kinds of terms used should be few, mathematically simple, and meaningful in the domain. As discussed, TMM uses differences of dates and SPAM uses the relative position, orientation, and scale of two frames of reference. MERCATOR [17] uses distances and orientations of line segments in the plane.

Networks are frequently organized hierarchically, where the elements of the hierarchy are single quantities or sets of quantities. For example, SPAM organizes each class of parameters (scale, orientation, position) into a more or less arbitrary tree. MERCATOR associates quantities with objects, and organizes objects hierarchically by physical containment. Allen [31] suggests the use of "reference intervals" to organize date lines, and Kahn and Gorry [42] use special reference dates in the same way. The hierarchy can be used to restrict the nodes in the system, by insisting that nodes related only parents and children, or possibly siblings, in the hierarchy. It can also be used to guide search, as discussed below.

The strict hierarchy can be extended by the use of a "kernel" set: a relatively small set of quantities which are in the focus of attention of the inference system. Within the kernel set, all term nodes are represented. Quantities are swapped into and out of the kernel as the system gains and loses interest in them. When a quantity is swapped out, only those terms which properly respect the hierarchy are preserved. Kernel sets are used in TMM [43].

When a constraint is added or a term is evaluated, it must be expressed as a function over a set of independent nodes. The major problem here is the search problem of finding a set of nodes connecting the quantities involved in the term. If a network is organized hierarchically, and terms are kept which relate quantities which are father-and-son or siblings in the hierarchy, then relatively efficient strategies can often be found using a GPS-like strategy. Given two quantities P and Q which must be connected, their common ancestor in the tree, A, is located. At one level down in the hierarchy from A, we connect the ancestor of P to the ancestor of Q; and then, recursively, we connect P to its ancestor and Q to its ancestor (Fig. 6).

11. Propagation and Deletion in Incremental Systems

There remain two problems with incremental systems. The first is the question of how far label inference should be taken when a new constraint is added to the system. In large networks, the complete Waltz algorithm may run very slowly, even for well-behaved types of constraints. Three possibilities suggest themselves:

(1) Propagate to quiescence and hope for the best. Generally, it is safe to assume that the effects of a single new constraint will only propagate through a

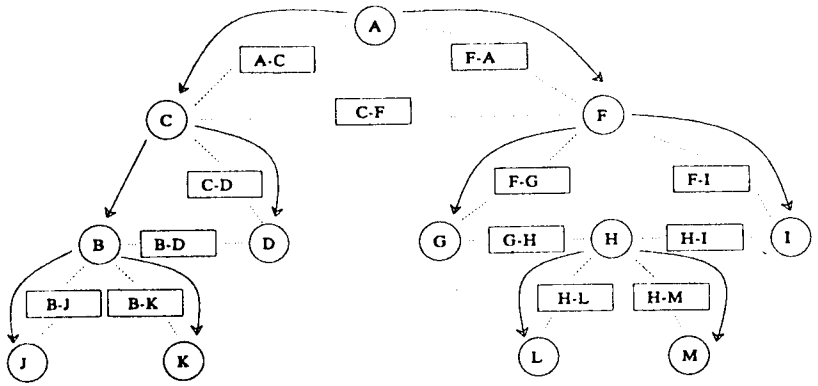

FIG. 6. Hierarchy of quantities. To evaluate $L-J$ as a function of the node terms, we first search from L and J upward in the tree until reaching the common ancestor A. At one level below that, we find C and F as the ancestors of L and J, connected by the node $C-F$. We must now connect C to J. Their common ancestor is C. One level down is B. B can be connected to C by going through D and using the two nodes $B-D$ and $C-D$. B is connected to J by the single node $B-J$. Connecting L and F is similar. Note that, if the net contained the node $L-J$ explicitly, this search technique would miss it.

small portion of the network. New information about the date of Hammurabi is unlikely to affect your belief about the date of the next Superbowl.

(2) Do not propagate at all. Refine the arguments of the new constraint once, and never refer to it again. In this approach, an explicit constraint is not saved at all, since it is only used once. This approach makes the behavior of the system strongly dependent on the order in which constraints are added, but it runs quickly and is often effective enough. It was used in SPAM [16], because the refinement algorithm used a hill-climber, and therefore ran so slowly that one refinement was all that could be afforded.

(3) Use some more or less arbitrary criterion (number of constraints refined on, elapsed time, etc.) to stop propagation after a while. This may not be suitable if the top-level program has a very flexible control structure, since propagation may be resumed accidentally.

The other problem involves deletion. Deleting constraints is often useful in incremental systems. Constraints are deleted when, temporally, they cease to be true, or when, inferentially, the knowledge base ceases to believe them. Generally, deletion is performed in networks using data dependencies, which record the connection between the conclusion of an inference and its premises [44]. Data dependencies work well when there are many premises and relatively shallow inference. The success of this scheme in constraint networks therefore depends critically on the presumption that any constraint will have

only local effects, even after more constraints have been added and propagated. However, it is hard to believe that this will be true in general; it would seem that, when enough time has passed after adding a constraint, almost all the labels will depend on it.[8] If so, then deleting a constraint will involve recomputing all labels, starting from earlier values, and using all the remaining constraints. This is essentially a wholesale computation; the incremental features of the system are lost. If heuristic techniques are used in systems with complex term nodes, then the problem is even more difficult, since choices as to how to express a constraint or which nodes to include may have depended on label values which are now invalidated. McDermott [45] has implemented a system in which each node contains a "signal function" which determines whether its value should be recomputed when its original support is withdrawn.

If interval labels on nodes are used together with data dependencies, then the upper and lower bound on a quantity should be considered separate data with separate dependencies. In general, the upper and lower bounds of a quantity will be derived from different constraints and will be used as a basis for inference in different constraints. If the constraints in the systems are linear inequalities, then this will always be true. For example, if C is the constraint "$5X+3Y>2Z$," then refining C may have the effect of changing the lower bounds of X and Y and the upper bound of Z; it cannot change the upper bounds of X and Y or the lower bound of Z. Moreover, these changes will be dependent only on the current upper bounds of X and Y and the current lower bound of Z. This is not true of nonlinear constraints; the constraint $X^2+Y^2\leq 4$ may change all the bounds of X and Y.

12. Conclusions

In terms of computational complexity, AI systems occupy a strange no-man's-land. The problems (game playing, theorem proving, scheduling, inductive inference) are known to be intractable in general, but AI systems must solve them for large, complex knowledge bases in close to constant time. To close this gap, we must either determine that the problems being addressed have special features that make them tractable, or we must be able to make do with approximate answers in a way which leads both to usable results and to fast algorithms.

Our analysis provides two arguments that label inference may be adequate for inference on quantities in important AI applications. Firstly, there is a significant class of problems which can be expressed in terms of order relationships and bounded differences. Such sets of relationships can be completely solved by wholesale label inference in polynomial time. Secondly, most applications of quantitative inference support a locality assumption that

[8] This is based on my own experience in spatial reasoning systems. Tom Dean informs me that, in temporal reasoning, such interactions are relatively slight (personal communication).

there is a natural grouping of quantities and that most constraints and terms relate quantities within the same grouping. It is plausible that an incremental system with such a locality property will lend itself to local algorithms, such as constraint propagation, and to a hierarchical organization; and these algorithms will run effectively and efficiently on such a structure. However, we have no proof of this.

Beyond these two limited results, our analysis has little that is encouraging. As the language of constraints is made more complex, the inherent computational complexity increases rapidly, and the effectiveness and efficiency of propagating sign or interval labels declines rapidly. We have not found any arguments that the partial results computed by label inference should be adequate for the purposes of AI.

Analysis having failed, we may fall back on empirical evaluation; how well do these systems work in practice? Here the evidence is more encouraging. Kuipers [5], Dean [23], and de Kleer and Brown [3] report, of their various systems, that in practice they run quickly and effectively on problems of substantial size (hundreds of nodes and constraints). Apparently, however, the code must be written with considerable care to achieve this [43, 46].

The major exceptions here have been the spatial reasoning systems SPAM [16] and MERCATOR [17]. (Quantitative inference was by no means the only or even the major problem in these systems; however, it was a substantial problem.) SPAM suffered from two problems. Firstly, the hill-climber used to perform refinement was unreliable and very slow. Secondly, SPAM's requirement that the node set be independent meant that much information was lost in going from explicit constraints to labelled nodes (see Section 10.) In MERCATOR, the problem was mainly one of scale and time; constraint propagation was unbearably slow in dealing with the networks involved (one or two hundred nodes, thousands of refinements), though it was reasonably effective in making important inferences. Possibly, this could have been alleviated by improving the underlying code.

The major outstanding research problems, therefore, seem to be the following:

(1) Design an effective system for quantitative inference in the spatial domain.

(2) Determine the scope of such successful techniques as those cited above. Apply them to other domains.

(3) Determine the power of systems like Kuipers' ENV [5], which propagate a simple binary constraint through more complex constraints.

(4) Demonstrate analytically a relation between a locality assumption and the successful running of an incremental system.

(5) Design an incremental system which allows for the effective deletion of constraints.

Appendix A. Propagation around Constraints of Bounded Differences

In this appendix, we consider constraint propagation around constraints of the form $x - y \in [a, b]$. We first consider networks which contain as nodes all terms of the form $X_{ij} = X_j - X_i$, and then we consider networks with only nodes of the form X_i. In networks of the first kind, there are no explicit constraints; the constraint propagation occurs through implicit constraints of the form $X_{ij} + X_{jk} = X_{ik}$. There are two important results: (1) the Waltz algorithm is complete for assimilation in each of these systems; (2) if the system of constraints is consistent, then, given a proper order for refining constraints, the Waltz algorithm will terminate in time $O(n^3)$ in each of these systems.

The problem of propagating labels around a network of terms X_{ij} is closely related to the problem of calculating shortest paths on weighted graphs. Construct a graph whose nodes are the quantities x_i, and which has an arc of cost c_{ij} from x_i to x_j just if the constraint $x_j - x_i \leq c_{ij}$ is in the system. Then given any path $x_1, x_2, \ldots, x_k$, with costs, $c_{12}, c_{23}, \ldots, c_{n-1,n}$, these correspond to the inequalities $x_2 - x_1 \leq c_{12}$, $x_3 - x_2 \leq c_{23}, \ldots$. Summing these inequalities gives $x_n - x_1 \leq c_{12} + c_{23} + \cdots + c_{n-1,n}$. Thus the relation $x_j - x_i \leq c$ is derivable from the equations iff there is a path from x_i to x_j of cost $\leq c$. Indeed, we can read the statement "$x_j - x_i \leq c$" as meaning just that there is a path from x_i to x_j of length at most c; in this case, the deduction is a deduction about paths in the graph. The system of equations is consistent just if there is no negative cost cycle in the graph. (The connection between inequalities of this form and the shortest paths problem was first observed in [47].)

The basic refinement rule for bounded differences is as follows: If $X_{ij} \in [a_{ij}, b_{ij}]$, $X_{jk} \in [a_{jk}, b_{jk}]$, $X_{ik} \in [a_{ik}, b_{ik}]$, then X_{ik} may be refined to $X_{ik} \in [\max(a_{ik}, a_{ij} + a_{jk}), \min(b_{ik}, b_{ij} + b_{jk})]$. This corresponds exactly to the central operation of most shortest path algorithms: $d(u, v) = \min(d(u, v), d(u, w) + d(w, v))$. Therefore, we can carry out these algorithms within a label inference system just by choosing the constraints to refine in the proper order. (The order, in other words, is all that the algorithm buys us.) In particular, we know that Floyd's algorithm finds all shortest paths in time $O(n^3)$ [48, p. 198]. Therefore, we likewise can cause label inference to terminate in time $O(n^3)$ for consistent sets of constraints and labels by choosing the constraints to refine in the same order as Floyd's algorithm: First perform all refinement of constraints $a_{i1} + a_{1j} = a_{ij}$, then on constraints $a_{i2} + a_{2j} = a_{ij}$, and so on. Moreover, the correctness of Floyd's algorithm guarantees the completeness of the Waltz algorithm.

Recently, Fredman and Tarjan [49] have devised an improvement to Floyd's algorithm which runs in time $O(n^2 \log n + en)$ where e is the number of edges

(constraints), by using a carefully tailored data structure to choose the order in which to perform the above operation. Therefore, this running time can be achieved for Waltz's algorithm, at the cost of using a very complex control structure. On the other hand there are known to be pathological examples where, by choosing to carry out the operations in the wrong order, it may take $O(2^n)$ operations to find the correct solution (see, for example, [48, p. 221 Problem 5.24]. Therefore, the Waltz algorithm must be careful in choosing the order to do refinement, to avoid an unnecessary exponential running time.

If the constraints are inconsistent, then it is possible for the Waltz algorithm to enter into an arbitrarily long loop. Consider, for example, performing label inference around the constraints $x = y$ and $x = y + 1$, starting with the labels $x \in [0, 1000]$, $y \in [0, 1000]$. We begin by using the second equation to deduce $x \in [1, 1000]$, $y \in [0, 999]$; then use the first equation to deduce $x \in [1, 999]$, $y \in [1, 999]$; then use the second equation to deduce $x \in [2, 999]$, $y \in [1, 998]$; and so on. It will take 500 iterations to discover the contradiction.

Systems which maintain just the simple quantities x_i as nodes can also be modelled in the same way. We can treat a bound on x_i as a bound on $x_i - x_0$, where x_0 is an additional quantity associated with the constant value 0. The problem is thus isomorphic to constructing single-source shortest paths algorithms, using only the assignments "$d(x_0, x) = \min(d(x_0, x),\ d(x_0, y) + d(y, x))$." If all arc lengths are positive—that is, all upper bounds are positive and all lower bounds are negative—then Dijkstra's algorithm gives an ordering of propagation which terminates in time $O(n^2)$. If there are negative arc lengths, then a cyclic order of refinement, as an Algorithm A.1, gives an $O(n^3)$ solution.

Algorithm A.1. Single-source shortest path algorithm.
/* This procedure computes the cost of the least cost path from 1 to i. */

```
procedure Single-source (C: cost matrix)
  var D: array of reals;   /* D[i] is the cost of the cheapest path from
                              1 to i */
  begin for i := 1 to n do D[i] := C [0, i];
    for k := 1 to n - 1 do
      for j := 1 to n do
        for i := 1 to n do D[j] := min(D[j], D[i] + C[i, j])
  end
```

/* At the beginning of the first iteration of the outer loop ($k = 1$), each vertex whose shortest path has one arc has D correctly set. At the end of the first iteration, each vertex whose shortest path has two arcs has D correctly set. At the end of the kth iteration, each vertex whose shortest path has k arcs has D correctly set. */

Appendix B. Interval Label Inference on Unit Coefficient Linear Inequalities

There are two major results in this appendix. The first deals with label inference around constraints of a general kind. It states that, for any set of interval refinements and any starting set of interval labels, one of two conditions holds. The refinements may contain a "self-dependency," in which the value of some bound depends on itself, in a strong sense. This is a potential infinite loop. If there is no such self-dependency, then there is a "bee-line" set of refinements, which reaches quiescence after changing each bound on each variable at most once.

The second result applies to refinement on unit coefficient constraints in particular. Here we can show that no consistent set of constraints can lead to a self-dependency, and that, therefore, there is always a bee-line set of refinements. We end with some discussion of how to apply these results algorithmically.

B.1. Dependencies and redundancies in general refinement

Definition B.1. Given an n-tuple of variables $\langle X_1, X_2, \ldots, X_n \rangle$, a *valuation* is a function from the variables to reals. We write $V(X_i) = v_i$ or $V = \langle v_1, \ldots, v_n \rangle$.

Example: $\langle 3, 5, 2 \rangle$ is a valuation on $\langle X_1, X_2, X_3 \rangle$.

Definition B.2. Given an n-tuple of variables $\langle X_1, X_2, \ldots, X_n \rangle$, a *labelling* is a function from the variables to closed intervals. We write $L(X_i) = [a_i, b_i]$ or $L = \langle [a_1, b_1], \ldots, [a_n, b_n] \rangle$. Note that a labelling L can be considered as a set of valuations on $\langle X_1, \ldots, X_n \rangle$, where a valuation V is in L if $V(X_i) \in L(X_i)$ for each i. Alternately, we can look at L as a valuation on the $2n$ variables $\langle X_1^l, X_1^u, X_2^l, X_2^u, \ldots \rangle$, where X_i^l, X_i^u are the upper and lower bounds of X_i. We will play off these two viewpoints in the course of the proof.

Example: $L = \langle [2.0, 4.0],\ [-1.0, 3.0],\ [5.0, 7.0] \rangle$ is a labelling on $\langle X_1, X_2, X_3 \rangle$. If $V = \langle 3.0, 0.0, 5.0 \rangle$, then $V \in L$. L is associated with the valuation $\langle 2.0, 4.0, -1.0, 3.0, 5.0, 7.0 \rangle$ on the variables $\langle X_1^l, X_1^u, X_2^l, X_2^u, X_3^l, X_3^u \rangle$.

Definition B.3. If B is a bound on a variable X (i.e. B is either X^l or X^u), then $\text{SIGN}(B) = 1$ if B is an upper bound and -1 if B is a lower bound. If L and L' are labellings, L' is said to be *tighter* than L on B if $L'(B) = L(B) - \text{SIGN}(B) \cdot \Delta$ for some $\Delta > 0$. Thus, an upper bound is tightened by lowering; a lower bound is tightened by raising.

Example: If $L = \langle [1.0, 3.0], [5.0, 7.0] \rangle$ and $L' = \langle [0.0, 3.0], [5.0, 6.0] \rangle$ are labellings on $\langle X_1, X_2 \rangle$, then L is tighter on X_1^l and L' is tighter on X_2^u.

Lemma B.4. *If L and L' are two labellings the following conditions are equivalent*:
(a) *L' is a subset of L.*
(b) *For each variable bound B, L' is tighter than or equal to L on B.*

Proof. Immediate from Definition B.3. In this case, we say that L' is at least as tight as L.

Example: $\langle [2.0, 4.0], [5.0, 6.0] \rangle$ is at least as tight as $\langle [1.0, 4.0], [5.0, 7.0] \rangle$.

Definition B.5. A *constraint* is a set of valuations. Given a constraint C, a labelling L, and a variable X we define $\text{REFINE}(C, L, X_i) = \{V(X_i) \mid V \in L \cap C\}$. This is the set of values of X_i consistent with the labelling and the constraint.

Example: If $C = \{V \mid V(X_2) \geq V(X_1)\}$ and $L = \langle [1.0, 3.0], [0.0, 2.0] \rangle$, then $\text{REFINE}(C, L, X_2) = [1.0, 2.0]$.

Definition B.6. Given a constraint C and a variable X_i we define two *refinement operators* $R^u(X_i, C)$ and $R^l(X_i, C)$, which are functions from labellings to labellings. If $L = \langle a_1, b_1, \ldots, a_n, b_n \rangle$, then $R^u(X_i, C)(L)$ is formed by replacing b_i with $\text{upper\_bound}(\text{REFINE}(C, L, X_i))$, and $R^l(X_i, C)(L)$ is formed by replacing a_i with $\text{lower\_bound}(\text{REFINE}(C, L, X_i))$. Thus, these refinement operators allow us to replace one bound at a time.

Example: If C and L are as in the previous example, then $R^l(X_2, C)(L) = \langle [1.0, 3.0], [1.0, 2.0] \rangle$ and $R^u(X_2, C)(L) = \langle [1.0, 3.0], [0.0, 2.0] \rangle$.

Lemma B.7. *For any constraint C, variable X and labelling L,*
(a) *$R^u(X, C)(L)$ and $R^l(X,)(L)$ are subsets of L and supersets of $C \cap L$.*
(b) *If L' is a subset of L, then $R^u(X, C)(L')$ is a subset of $R^u(X, C)(L)$, and likewise for R^l.*

Proof. Immediate from the definitions.

Definition B.8. For any refinement operator R, the *output* bound of R, $\text{OUT}(R)$, is the bound which is affected by R. The *arguments* of R, $\text{ARGS}(R)$, are the bounds, other than $\text{OUT}(R)$ itself, which enter into the calculation of $\text{OUT}(R)$.

Example: Let C be the constraint $X_1 \geq X_3 + X_2 - 4.0$. $R = R^u(X_3, C)$ is the replacement of X_3^u by $\min(X_3^u, X_1^u - X_2^l + 4.0)$. Therefore $\text{OUT}(R) = X_3^u$ and $\text{ARGS}(R) = \{X_1^u, X_2^l\}$.

Henceforward, we will be looking at sets of constraints and sets of refinement operators derived from those constraints. Given a series $R = \langle R_1, R_2, \ldots, R_m \rangle$ of refinements, and a labelling L, we will denote the composition of all the refinements as $R(L) = R_m(R_{m-1}(\cdots(R_2(R_1(L)))\cdots))$. A series S is a subseries of R if S contains a subset of the refinements in R in the same order. S need not be a consecutive subset; $\langle R_1, R_3, R_6 \rangle$ is a subseries of $\langle R_1, R_2, R_3, R_4, R_5, R_6 \rangle$. Given a series $\langle R_1, \ldots, R_m \rangle$, the notation $R_{i,\ldots,j}$ denotes the consecutive subseries $\langle R_i, R_{i-1}, \ldots, R_j \rangle$. Series are assumed to be finite unless otherwise specified.

Lemma B.9. *Given a series R of refinement operators, if S is a subseries of R, then, for all labellings L, $R(L)$ is at least as tight as $S(L)$.*

Proof. By induction on the length k of S. If S has zero length, then $S(L) = L$ and $R(L)$ is tighter than L by Lemma B.7(a). If the statement is true for $S_{1,\ldots,k-1}$, then let $S_k = R_j$. Since $S_{1,\ldots,k-1}$ is a subseries of length $k-1$ of $R_{1,\ldots,j-1}$, we know inductively that $R_{1,\ldots,j-1}(L)$ is at least as tight as $S_{1,\ldots,k-1}(L)$. Applying $S_k = R_j$ to this equation and using Lemma B.7(b) gives $R_{1,\ldots,j}(L)$ is at least as tight as $S_{1,\ldots,k}(L)$.

Definition B.10. Given a series of refinement operators $R_1, \ldots, R_m$ and a labelling L, R_i is said to be *active* if it changes the value of L; that is, $R_{1,\ldots,i}(L) \neq R_{1,\ldots,i-1}(L)$. If all the refinement in a series are active on L, the series is active on L.

Example: In the example of Definitions B.5 and B.6, $R^u(X_2, C)$ is not active on L; $R^l(X_2, C)$ is active.

Lemma B.11. *Given a series R of refinement operators, and a labelling L, if Q is the subseries consisting of the active refinements, then $Q(L) = R(L)$.*

Proof. Clearly, since inactive refinements do not affect the labelling, they may be omitted without changing the result.

Definition B.12. Given a series of refinements $R_1, \ldots, R_m$, we say that R_i is an immediate predecessor of R_j if $i < j$, $\text{OUT}(R_i) \in \text{ARGS}(R_j)$, and for all k such that $i < k < j$, $\text{OUT}(R_k) \neq \text{OUT}(R_i)$. Thus, some particular argument of R^j has been set most recently in the series by R_i. We say that R_j depends on R_i if (a) $j = i$ or (b) (recursively) R_j depends on R_k, and R_i is an immediate predecessor of R_k. We say that R_j depends on some bound B if, for some i, R_j depends on R_i and $B \in \text{ARGS}(R_i)$.

Example: Suppose $R = \langle R_1, R_2, R_3, R_4 \rangle$ and $\text{OUT}(R_1) = B_1$, $\text{ARGS}(R_1) = \{B_2, B_3\}$; $\text{OUT}(R_2) = B_2$, $\text{ARGS}(R_2) = \{B_4\}$; $\text{OUT}(R_3) = B_3$, $\text{ARGS}(R_3) = \{B_1, B_2\}$; $\text{OUT}(R_4) = B_1$, $\text{ARGS}(R_4) = \{B_2, B_3\}$. Then R_1 and R_2 are im-

mediate predecessors of R_3; R_2 and R_3 are immediate predecessors of R_4; and R_4 depends (indirectly) on R_1 (see Fig. B.1).

Lemma B.13. *Let $R = \langle R_1, \ldots, R_m \rangle$ be a series of refinements, not all of which are dependent on R_1. Let $Q = \langle Q_1 = R_1, Q_2, \ldots \rangle$ be the refinements which depend on R_1 in order, and let $P = \langle P_1, P_2, \ldots, P_k \rangle$ be the refinements which do not depend on R_1. Then the series $R' = \langle P_1, P_2, \ldots, P_k, Q_1, Q_2, \ldots \rangle$ is at least as tight as R on any labelling L.*

Proof. The P do not depend on any of the Q. Hence the value of the arguments of the P are unchanged and they return the same values for their output variables. When each Q_i is entered, the input labelling is at least as tight as in the original series. Hence, the output variables are made at least as tight as they were in the original series. Thus, when we are all done, each variable has been set to a value at least as tight as its setting in the original series of refinements.

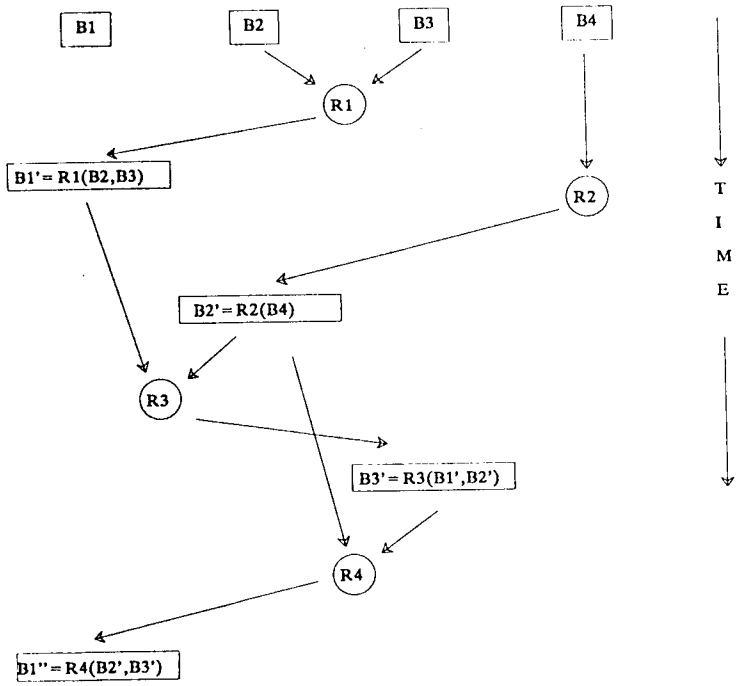

FIG. B.1. Dependency of refinements.

Example: In the example of Definition B.12, the refinement series $R' = \langle R_2, R_1, R_3, R_4 \rangle$ is at least as tight as the series $R = \langle R_1, R_2, R_3, R_4 \rangle$, since R_2 does not depend on R_1. The reverse is not necessarily true, since, in R', R_1 does depend on R_2.

Definition B.14. A series of refinements $R = \langle R_1, \ldots, R_m \rangle$ is *self-dependent* for labelling L if it is active on L and R_m depends on $\text{OUT}(R_m)$, its own output variable. R *contains a self-dependency* for L if, for some i and j, the subseries $R_i, \ldots, R_j$ is self-dependent for $R_{1,\ldots,i-1}(L)$.

Example: In the example of Definition B.12, if the series is active on some labelling L, then it is self-dependent on L since R_4 depends on $B_1 = \text{OUT}(R_4)$.

Lemma B.15. *Any infinite sequence of active refinements contains a self-dependency for L.*

Proof. For any N it is possible to choose refinements R_i, R_j, where $j > i > N$ and where R_i is an immediate predecessor of R_j. For if it were not possible, this would mean that for all $j > N$, all the arguments of R_j are set by refinement before N. This would mean that no refinement could be active twice after N, since it would get the same value as before. But in an infinite sequence of finitely many refinements, some refinements must appear infinitely often. By iterating this argument, we find that it is possible to find a subseries $Q = \langle Q_1, Q_2, \ldots, Q_k \rangle$ such that Q_i is an immediate predecessor of Q_{i+1} in R. Since this list can be made arbitrarily long, at some point we will have two Q with the same output bound. At this point, we have a self-dependency.

Definition B.16. A series of refinements, is *redundant* if two refinements have the same output variable bound.

Example: The series of refinements in the example of Definition B.12 is redundant, since R_1 and R_4 have the same output variable B_1.

Lemma B.17. *Given a redundant series of active refinements $R = \langle R_1, \ldots, R_m \rangle$ and a labelling L, either the series contains a self-dependency for L or there exists a shorter series which is at least as tight on L.*

Proof. Assume, without loss of generality, that all the refinements are active on L. Since R is redundant, it contains multiple refinements with the same output variable. Let R_i and R_j, $i < j$, be two refinements with the same output variable such that no refinements in between i and j has that same output variable. There are two cases to consider:

(1) R_j depends on R_i. In this case, the series contains a self-dependency.

(2) R_j does not depend on R_i. Let $B = \text{OUT}(R_i)$. Consider the subseries $S = \langle R_i, \ldots, R_j \rangle$. As in Lemma B.13, let $Q = \langle Q_1 = R_i, Q_2, \ldots \rangle$ be the

refinements between R_i and R_j which do depend on R_i and let $P = \langle P_1, P_2, \ldots, P_k = R_j \rangle$ be those which do not depend on R_i. Then, by Lemma B.13 the series $S' = \langle P_1, P_2, \ldots, P_k, Q_1, Q_2, \ldots \rangle$ is at least as tight as S. Moreover, even if we remove R_i from S', leaving $S'' = \langle P_1, P_2, \ldots, P_k, Q_2, Q_3, \ldots \rangle$, S'' is still as tight as S. For, in the original series S, R_j was active. Therefore, it set $\text{OUT}(R_j)$ to a tighter value than did R_i. Since it does not depend on any of the Q_i, it sets $\text{OUT}(R_j)$ to the same value in S'' as it did in S. The only effect of the inclusion of R_i in S', as opposed to its omission in S'', is that B may be set to a higher value still, but even without this it is still as tight as in S. But S'' is shorter than S. If we replace S'' for S in the original series R, we obtain a series shorter than R but at least as tight.

Example: Let $R = \langle R_1, R_2, R_3, R_4 \rangle$ where $\text{OUT}(R_1) = B_1$, $\text{ARGS}(R_1) = \{B_2, B_3\}$; $\text{OUT}(R_2) = B_2$, $\text{ARGS}(R_2) = \{B_3\}$; $\text{OUT}(R_3) = B_1$, $\text{ARGS}(R_3) = \{B_2, B_4\}$; $\text{OUT}(R_4) = B_5$, $\text{ARGS}(R_4) = \{B_1, B_2\}$. Suppose R is active on some labelling L. By Lemma B.13, the modified series $R' = \langle R_2, R_3, R_1, R_4 \rangle$ is at least as tight on L, since neither R_2 nor R_3 depends on R_1. Moreover, the value calculated by R_3 for B_1 is unaffected by the transformation, since, in either series, R_3 depends only on the values of B_2, and B_4; and the value of B_2 calculated by R_2 depends only on the starting value of B_3. Thus, nothing is lost to R_3 by not having R_1 before it. If we now delete R_1 from R', leaving $R'' = \langle R_2, R_3, R_4 \rangle$, the state of the labelling on entering R_4 is exactly the same as it was in R. Thus R'' is as tight as R, and shorter (see Fig. B.2).

Definition B.18. Given a set of refinements $S = \{R_1, \ldots, R_k\}$, a labelling L is quiescent on the set if $R_1(L) = R_2(L) = \cdots = L$. Given a set of refinements $S = \{R_1, \ldots, R_k\}$, a labelling L and a series of refinements from S, $Q = \langle Q_1, Q_2, \ldots, Q_m \rangle$, we say that Q brings L to quiescence if $Q(L)$ is quiescent on S.

Lemma B.19. *Given a set of refinements S and a labelling L, let P and Q be two series of refinements drawn from S. If $Q(L)$ is quiescent on S, then it is at least as tight as $P(L)$.*

Proof. Since $Q(L)$ is quiescent, $P(Q(L)) = Q(L)$. (None of the refinements in P can have any effect on $Q(L)$.) However, since $Q(L)$ is at least as tight as L, $P(Q(L))$ is at least as tight as $P(L)$. Hence $Q(L)$ is at least as tight as $P(L)$.

Corollary B.20. *Given a set of refinements S and a labelling L, if Q and P are two different series of refinements both of which bring L to quiescence, then $Q(L) = P(L)$.*

Proof. By the above lemma, $P(L)$ is as tight as $Q(L)$ and vice versa, so the two are equal.

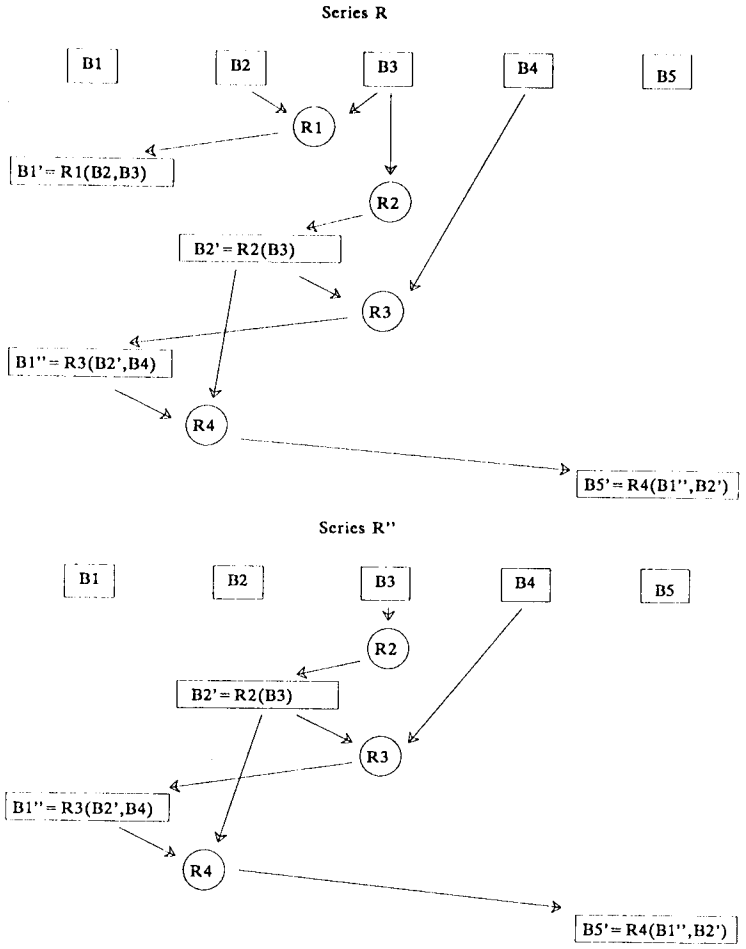

FIG. B.2. Compressing a series of refinements.

Theorem B.21. *Given a set of refinements, $S = \{R_1, \ldots, R_m\}$, on n variables, and a labelling L, then either there exists a series of refinements in S containing a self-dependency on L, or there exists a series of refinements which is not redundant which brings L to quiescence.*

Proof. Let Q be any infinite series of refinements from S in which each refinement appears infinitely often. Let R be the subseries of Q containing all

the active refinements. If R has no self-dependency, then by Lemma B.15 it must be finite. If R is finite, then $R(L)$ must be quiescent on S, since, after the end of R, Q tries all the different refinements in S and none of them are active. Moreover, since R has no self-dependency, it can be gradually pruned down, step-by-step, using Lemma B.17 until it has no redundancies left. Each step leaves it at least as tight as it was before. Since no series of refinements can get tighter than the quiescent state (Lemma B.19) the resultant series of refinements must reach quiescence.

B.2. Unit coefficient constraints

Definition B.22. A *unit coefficient constraint* is a relation C of the form

$$p \leq \sum_{k \in \mathrm{PC}(C)} X_k - \sum_{k \in \mathrm{NC}(C)} X_k \leq q ,$$

where p and q are constants and $\mathrm{PC}(C)$ and $\mathrm{NC}(C)$ are disjoint sets of variable indices. (PC and NC stand for "positive coefficients" and "negative coefficients.")

Henceforward in this section, we will assume that all constraints are unit coefficient constraints, and that all refinements are drawn from unit coefficient constraints.

Lemma B.23. *For any variable X_i, labelling L, and constraint C of the form in Definition B.22,* $\mathrm{REFINE}(C, L, X_i)$ *can be calculated as follows: Let* $L(X_k) = [a_k, b_k]$ *for all k and let* $\mathrm{REFINE}(C, L, X_i) = [a_i', b_i']$.

If $i \in \mathrm{PC}(C)$, *then*

$$a_i' = \max\left(a_i,\ p + \sum_{k \in \mathrm{NC}(C)} a_k - \sum_{k \in \mathrm{PC}(C), k \neq i} b_k\right),$$

$$b_i' = \min\left(b_i,\ q + \sum_{k \in \mathrm{NC}(C)} b_k - \sum_{k \in \mathrm{PC}, k \neq i} a_k\right).$$

If $i \in \mathrm{NC}(C)$, *then*

$$a_i' = \max\left(a_i,\ \sum_{k \in \mathrm{PC}(C)} a_k - q - \sum_{k \in \mathrm{NC}(C), k \neq i} b_k\right),$$

$$b_i' = \min\left(b_i,\ \sum_{k \in \mathrm{PC}(C)} b_k - p - \sum_{k \in \mathrm{NC}(C), k \neq i} a_k\right).$$

Proof. By solving the inequality.

Lemma B.24. *If refinement $R(X, C)$ is active on L, then R has one of the following forms:*

If $i \in \mathrm{PC}(C)$, *then*

$$a_i' = p + \sum_{k \in \mathrm{NC}(C)} a_k - \sum_{k \in \mathrm{PC}(C), k \neq i} b_k ,$$

$$b_i' = q + \sum_{k \in \mathrm{NC}(C)} b_k - \sum_{k \in \mathrm{PC}(C), k \neq i} a_k .$$

If $i \in \mathrm{NC}(C)$, *then*

$$a_i' = \sum_{k \in \mathrm{PC}(C)} a_k - q - \sum_{k \in \mathrm{NC}(C), k \neq i} b_k ,$$

$$b_i' = \sum_{k \in \mathrm{PC}(C)} b_k - p - \sum_{k \in \mathrm{NC}(C), k \neq i} a_k .$$

Proof. Immediate from Lemma B.23.

Lemma B.25. *Let the refinement R be active on L. Let B be a bound in* $\mathrm{ARGS}(R)$. *Let L' be tighter than L, and let $L'(B)$ be tighter than $L(B)$ by an amount $\Delta > 0$. Then $R(L')$ is tighter than $R(L)$ on* $\mathrm{OUT}(R)$ *by at least Δ.*

That is, under the specified conditions, a change in the input is mirrored by at least as great a change in the output. Here the limitation to unit coefficient equations is critical; this is not true of other kinds of refinement.

Proof. Since R is active on L, $R(L)$ evaluates $\mathrm{OUT}(R)$ as the sum and difference of the bounds in $\mathrm{ARGS}(R)$, including B. R is not necessarily active on L', since $\mathrm{OUT}(R)$ may have been very much tightened in L'. However, in computing $\mathrm{OUT}(R)$ in $R(L')$ we make sure that it is at least as tight as the same sum and difference over the values in L'. Since tightening in the arguments shows up as tightening in the sum with coefficient 1, this sum is tighter in L' than in L by the sum of the tightenings of all the bounds in $\mathrm{ARGS}(R)$. Hence, $\mathrm{OUT}(R)$ is tighter in $R(L')$ than in $R(L)$ by at least the sum of all the tightenings. However, this sum contains no negative terms, and one term Δ. Hence, $R(L')$ is tighter than $R(L)$ on $\mathrm{OUT}(R)$ by at least Δ.

Example: Let C be the constraint $4 \geq X_1 - X_2 + X_3$. Let $R = R^{\mathrm{u}}(X_3, C)$. Thus R sets X_3^{u} to $\min(X_3^{\mathrm{u}}, -X_1^{\mathrm{l}} + X_2^{\mathrm{u}} + 4)$. Let $L = \langle [5.0, 8.0], [2.0, 4.0], [0.0, 5.0] \rangle$, and $L' = \langle [7.0, 8.0], [2.0, 4.0], [0.0, 2.0] \rangle$. The L' is tighter by 2.0 on X_1^{l}, which is an $\mathrm{ARGS}(\mathrm{OUT}(R))$. $R(L)$ evaluates X_3^{u} as 3.0, and $R(L')$ evaluates it as 1.0. Thus $R(L)$ is tighter than $R(L')$ by 2.0 on X_3^{u}.

Lemma B.26. *Let the sequence $R = \langle R_1, \ldots, R_m \rangle$ be active on a labelling L and dependent on the variable bound B. Let $B_m = \mathrm{OUT}(R_m)$ be the last variable*

bound set. Let L' be a labelling which is tighter than L, such that $L'(B) = L(B) - \text{SIGN}(B) \cdot \Delta$, for some $\Delta > 0$. Then $R(L')$ is tighter than $R(L)$ on B_m by at least Δ.

Proof. By Lemma B.7(a), since L' is tighter than L, $R_{1,\ldots,i}(L')$ is at least as tight as $R_{1,\ldots,i}(L)$ for each i. Let $R_{i_1}, R_{i_2}, \ldots$ be a subseries of R such that $B \in \text{ARGS}(R_{i_1})$, $B_1 = \text{OUT}(R_{i_1}) \in \text{ARGS}(R_{i_2})$, etc. Since no refinement has reset B_0 before applying R_{i_1}, it is still true that $R_{1,\ldots,i_1-1}(L')$ is tighter than $R_{1,\ldots,i_1-1}(L)$ on B by Δ. (These are the labellings just before applying R_{i_1}.) By Lemma B.25, it follows that $R_{1,\ldots,i_1}(L')$ is tighter than $R_{1,\ldots,i_1}(L')$ on B_1 by at least Δ. (These are the labellings after applying R_{i_1}.) Similarly, since B_1 is not reset before applying L_{i_2}, it follows that $R_{1,\ldots,i_2-1}(L')$ is tighter than $R_{1,\ldots,i_2-1}(L)$ on B_1 by Δ. Again, it follows that $R_{1,\ldots,i_2}(L')$ is tighter than $R_{1,\ldots,i_2}(L')$ on B_2 by at least Δ. And so it goes inductively, up through R_m.

Lemma B.27. *Let $R = \langle R_1, \ldots, R_m \rangle$ be self-dependent for L. Let $B = \text{OUT}(R_m)$. Construct the infinite series of refinements*

$$R^\infty = RRR \cdots = \langle R_1, R_2, \ldots, R_m, R_1, R_2, \ldots, R_m, R_1, \ldots \rangle$$

by iterating the series R infinitely often. Let $L_0 = L$, $L_1 = R(L)$, $L_2 = R^2(L) = R(R(L))$, etc. Then the sequence $L_0(B)$, $L_1(B)$, $L_2(B)$, ... increases or decreases at least arithmetically.

Proof. Since R_m is active, $R(L) = L_1$ tighter than L on B. Let $\Delta = \text{abs}(L_1(B) - L(B))$. Then L_1 satisfies the conditions for L' in Lemma B.26. Hence $R^2(L) = L_2$ is tighter than L_1 on B by at least Δ, and hence tighter than L by at least 2Δ. Using L_2 as L' in Lemma B.26 gives the conclusion that L_3 is tighter than L_2 on B by at least Δ. Inductively, L_k is tighter than L on B by at least $k\Delta$.

Lemma B.28. *If $\langle R_1, \ldots, R_m \rangle$ is self-dependent for L, then the original set of constraints and labels was inconsistent.*

Proof. We are able to construct a series of refinements which tightens B toward plus or minus infinity. This means either that some upper bound on some variable is arbitrarily small or that some lower bound is arbitrarily large, which is an impossible conclusion. However, since each refinement is a necessary consequence of the original state of the system, the original system must have been inconsistent.

Theorem B.29. *If $\langle R_1, \ldots, R_n \rangle$ contains a self-dependency for L, then the original constraints together with L are inconsistent.*

Proof. Let $\langle R_i, \ldots, R_j \rangle$ be self-dependent. The original set of constraints together with L imply L_{i-1}. Using Lemma B.28 on L_{i-1} and $\langle R_i, \ldots, R_j \rangle$ shows that the constraints and original labellings are inconsistent.

Theorem B.30. *Starting with a consistent set of constraints and a labelling L, there is a nonredundant series of refinements which reaches a quiescent state. This may be called a "bee-line" series for L.*

Proof. Follows immediately from Theorems B.21 and B.29.

Corollary B.31. *Given a set of constraints on n variables and a labelling L which are consistent, let R be a series containing each refinement drawn from the set exactly once. Then $R^{2n} = \langle R, R, R, \ldots (2n \text{ times}) \rangle$ reaches quiescence.*

Proof. R^{2n} contains the bee-line series as a subseries, and therefore is as tight as the bee-line series on L. Since the bee-line series reaches quiescence, so does R^{2n}.

The only remaining question is how quickly this can be implemented. It turns out that all the refinements for a single constraint on a labelling can be done together in time O(length of the constraint), by incrementally changing the sum and differences in Lemma B.24, rather than recalculating each time. Thus, the average time to apply a refinement is O(1). Thus, we are repeating $2E$ refinements $2n$ times, giving us an $O(nE)$ algorithm.

The techniques of Mackworth and Freuder [18], used in Algorithm 3.2, can be applied to make this practically more efficient, if not to give theoretically better bounds. The idea is that if none of the arguments of a given constraint have changed since it was last refined, then the constraint need not be refined again. Thus, we can use a queueing structure, where a constraint is added to the queue when one of its arguments is changed. It is easily shown that, if constraints are taken off the queue, either in fixed, cyclic order or in FIFO order, then the series of refinements will be a superseries of the bee-line series when it is not more than E times as long, where E is the number of refinements. Thus, either ordering scheme gives us a worst case of $O(nE)$ running time before reaching quiescence.

ACKNOWLEDGMENT

Drew McDermott introduced me to the problems described here, and supervised much of my research on them. I would also like to thank Sanjaya Addanki, Philip Davis, Tom Dean, and the reviewers for their comments on an earlier draft. The work in this paper was supported in part by NSF grant DCR-8402309.

REFERENCES

1. Waltz, D., Understanding line drawings of scenes with shadows, in: P. Winston (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).
2. Stefik, M., Planning with constraints (MOLGEN: Part 1), *Artificial Intelligence* **16** (1981) 111–139.
3. de Kleer, J. and Brown, J. S., A qualitative physics based on confluences, *Artificial Intelligence* **24** (1984) 7–83.
4. Davis, R., Diagnosis via causal reasoning: Paths of interaction and the locality principle, in: *Proceedings AAAI-83*, Washington, DC (1983) 88–92.
5. Kuipers, B., Commonsense reasoning about causality: Deriving behavior from structure, *Artificial Intelligence* **24** (1984) 169–203.
6. Simmons, R., 'Commonsense' arithmetic reasoning, *Proceedings AAAI-86*, Philadelphia, PA, 1986.
7. Brooks, R.A., Symbolic reasoning among 3-D models and 2-D images, *Artificial Intelligence* **17** (1981) 285–348.
8. Ambler, A. P. and Popplestone, R. J., Inferring the positions of bodies from specified spatial relationships, *Artificial Intelligence* **6** (1975) 157–174.
9. Lozano-Perez, T., The design of a mechanical assembly system, Tech. Rept. AI-TR-397, MIT, Cambridge, MA, 1976.
10. Taylor, R.H., A synthesis of manipulator control programs from task-level specifications, Memo AIM 282, Stanford University, Stanford, CA, 1976.
11. Sutherland, I.E., SKETCHPAD: A man-machine graphical communication system, MIT Lincoln Labs, Tech. Rept. 296 Cambridge, MA, 1963.
12. Borning, A., ThingLab—An object oriented system for building simulations using constraints, in: *Proceedings IJCAI-77*, Cambridge, MA (1977) 497–499.
13. Sussman, G.J. and Steele, G. L., CONSTRAINTS—A language for expressing almost hierarchical descriptions, *Artificial Intelligence* **14** (1980) 1–39.
14. Southwell, R., *Relaxation Methods in Engineering Science* (Clarendon, Oxford, 1940).
15. Hummel, R.A. and Zucker, S.W., On the foundations of relaxation labeling processes, *IEEE Trans. Patt. Anal. Mach. Intell.* **5** (1983) 267–287.
16. McDermott, D.V. and Davis, E., Planning routes through uncertain territory, *Artificial Intelligence* **22** (1984) 107–156.
17. Davis, E., Representing and acquiring geographic knowledge, Research Rept. 292, Yale University, New Haven, CT, 1984.
18. Mackworth, A.K. and Freuder, E.C., The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artificial Intelligence* **25** (1985) 65–74.
19. Mackworth, A.K., Consistency in networks of relations, *Artificial Intelligence* **8** (1977) 99–118.
20. Freuder, E., Synthesizing constraint expressions, *Commun. ACM* **21** (11) (1978) 958–966.
21. McDermott, D. V., Spatial inferences with ground, metric formulas on simple objects, Research Rept. 173, Yale University, New Haven, CT, 1980.
22. Boggess, L., Computational interpretation of English spatial prepositions, Rept. T-75, CSL, University of Illinois, Urbana-Champaign, IL, 1979.
23. Dean, T., Temporal imagery: An approach to reasoning about time for planning and problem solving, Research Rept. 433, Yale University, New Haven, CT, 1985.
24. Alefeld, G. and Herzberger, J., *Introduction to Interval Computation* (Academic Press, New York, 1983).
25. Winograd, T., *Understanding Natural Language* (Academic Press, New York, 1972).
26. Forbus, K., Qualitative process theory, *Artificial Intelligence* **24** (1984) 85–168.
27. Sacerdoti, E., *A Structure for Plans and Behavior* (Elsevier, New York, 1977).
28. Vere, S., Planning in time: Windows and durations for activities and goals, *IEEE Trans. Patt. Anal. Mach. Intell.* 5 (3) (1983) 246–267.

29. Fahlman, S.E., A planning system for robot contruction tasks, *Artificial Intelligence* **5** (1974) 1–49.
30. Malik, J. and Binford, T.O., Reasoning in time and space, in: *Proceedings IJCAI-83*, Karlsruhe, F.R.G. (1983) 343–345.
31. Allen, J.F., Maintaining knowledge about temporal intervals, *Commun. ACM* **26** (1983) 832–843.
32. Vilain, M. and Kautz, H., Constraint propagation algorithms for temporal reasoning, in: *Proceedings AAAI-86*, Philadelphia, PA, 1986.
33. Williams, B.C., Qualitative analysis of MOS circuits, *Artificial Intelligence* **24** (1984) 281–346.
34. Yemini, Y., Some theoretical aspects of position-location problems, in: *Proceedings 20th Symposium on the Foundations of Computer Science* (1979) 1–7.
35. Kozen, D. and Yap, C.K., Algebraic cell decomposition in NC, in: *Proceedings 26th Symposium on the Foundations of Computer Science* (1985) 515–521.
36. Richardson, D., Some undecidable problems involving elementary functions of a real variable, *J. Symbolic Logic* **33** (1968) 514–520.
37. Karmakar, N., A new polynomial time algorithm for linear programming, in: *Proceedings 16th ACM Symposium on the Theory of Computing*, Washington, DC (1984) 302–311.
38. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, CA, 1979).
39. McAllester, D., An outlook on truth maintenance, Tech. Memo 521, MIT AI Lab, Cambridge, MA, 1980.
40. Luenberger, D.G., *Linear and Non-Linear Programming* (Addison-Wesley, Reading, MA, 1973).
41. Davis, E., Organizing spatial knowledge, Research Rept. 193, Yale University, New Haven, CT, 1981.
42. Kahn, K. and Gorry, G.A., Mechanizing temporal knowledge, *Artificial Intelligence* **9** (1977) 87–108.
43. Dean, T., Planning and temporal reasoning under uncertainty, in: *Proceedings IEEE Conference on Knowledge Representation*, Denver, CO, 1984.
44. Doyle, J., A truth-maintenance system, *Artificial Intelligence* **12** (1979) 231–272.
45. McDermott, D.V., Data dependencies on inequalities, in: *Proceedings AAAI-83*, Karlsruhe, F.R.G. (1983) 266–269.
46. Kuipers, B., Personal communication.
47. Pratt, V.R., Two easy theories whose combination is hard, Tech. Rept., MIT, Cambridge, MA, 1977. Cited in: Shostak, R., Deciding linear inequalities by computing loop residues, *J. ACM* **28** (4) (1981) 769–779.
48. Aho, A.V., Hopcroft, J. and Ullman, J., *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).
49. Fredman, M.L. and Tarjan, R., Fibonacci heaps and their uses in improved network optimization algorithms, in: *Proceedings Symposium on the Foundations of Computer Science* (1984) 338–346.
50. Allen, J. and Kautz, H., A model of naive temporal reasoning, in: J. Hobbs and R. Moore (Eds.), *Formal Theories of the Commonsense World* (Ablex, Norwood, NJ, 1985) 251–268.

Received November 1985; revised version received July 1986