BELIEF MAINTENANCE IN DYNAMIC CONSTRAINT NETWORKS*

Rina Dechter

Cognitive Systems Laboratory
Computer Science Department
University of California, Los Angeles, CA 90024

Avi Dechter

Department of Management Science
California State University, Northridge, CA 91330

Abstract

This paper presents a constraint network formulation of belief maintenance in dynamically changing environments. We focus on the task of computing the degree of support for each proposition, i.e., the number of solutions of the constraint network which are consistent with the proposition. The paper develops an efficient distributed scheme for calculating and revising beliefs in acyclic constraint networks. The suggested process consists of two phases. In the first, called **support propagation**, each variable updates the number of extensions consistent with each of its values. The second, called **contradiction resolution**, is invoked by a variable upon detecting a contradiction, and identifies a minimal set of assumptions that potentially account for the contradiction.

1. Introduction

Reasoning about dynamic environments is a central issue in Artificial Intelligence. When dealing with a complex environment, we normally have only partial description of the world known explicitly at any given time. A complete picture of the environment can only be speculated by making simplifying assumptions which are consistent with the available information. When new facts become known, it is important to maintain the consistency of our view of the world so that queries of interest (e.g., is a certain proposition believed to be true?) can be answered coherently at all times. Various non-monotonic logics as well as truth-maintenance systems have been devised to handle such tasks [Reiter 1987, Doyle 1979, de Kleer 1986].

In this paper we show that **constraint networks** and their associated **constraint satisfaction problems** provide an attractive paradigm for modeling dynamically changing environments. The language of constraint networks was originally developed for expressing **static** problems, i.e., that require a one-time solution of a system of constraints representing all the available information (for example, picture processing [Montanari 1974, Waltz 1975]). A substantial body of knowledge for solving such problems has been developed [Montanari 1974, Mackworth 1977, Freuder 1982, Dechter 1987].

Structuring knowledge by means of constraint networks leads, as we will show, to efficient algorithms for consistency maintenance and query processing. Indeed, truth-maintenance systems often utilize algorithms found in constraint processing in general, e.g., dependency-directed backtracking, constraint propagation, etc. [Stallman 1977, McAllester 1980, Doyle 1979]. The use of constraint networks as the framework for modeling the task of dynamic belief management, allows us to develop an efficient processing algorithm built upon techniques used in the solution of constraint satisfaction problems. Two characteristic features of these techniques are that they are "sensitive" to the structure of the problem so as to take advantage of special structures, and that their performance can be analyzed and predicted. Such theoretical treatment is usually not available in current TMS research.

The paper is organized as follows. Section 2 provides a brief review of the constraint network model and discusses the problem of belief maintenance in its context. The suggested belief revision process consists of two phases, presented first for singly connected binary constraint networks. The first, **support propagation**, is described in Section 3, and the second, **contradiction resolution**, is the subject of Section 4. In Section 5 the algorithm is extended to acyclic networks. Section 6 discusses the extension of the algorithm to general networks, and Section 7 contains a summary and some final remarks.

2. The Model

A constraint network (CN) involves a set of n variables, $X_1, \ldots, X_n$, their respective **domains**, $R_1, \ldots, R_n$, and a set of **constraints**. A constraint $C_i(X_{i_1}, \cdots, X_{i_j})$ is a subset of the Cartesian product $R_{i_1} \times \cdots \times R_{i_j}$ that specifies which values of the variables are compatible with each other. A binary constraint network is one in which all the constraints are binary, i.e., involve at most two variables. A binary CN may be associated with a **constraint-graph** in which nodes represent variables and arcs connect those pairs of variables for which

*This work was supported in part by the National Science Foundation, Grant #DCR 85-01234, and by the Air Force Office of Scientific Research, Grant #AFOSR-88-0177.

constraints are given. Consider, for instance, the CN presented in Figure 1 (modified from [Mackworth 1977]). Each node represents a variable whose values are explicitly indicated, and each link is labeled with the set of value-pairs permitted by the constraint between the variables it connects (observe that the constraint between connected variables is a strict lexicographic order along the arrows).

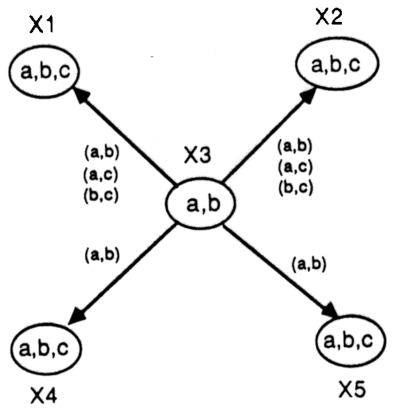

Figure 1: An example of a binary CN

A **solution** (also called an **extension**) of a constraint network is an assignment of values to all the variables of the network such that all the constraints are satisfied. The (static) constraint satisfaction problem associated with a given constraint network is the task of finding one or all of the extensions. In this paper we focus on a related problem, that of finding, for each value in the domain of certain variables, the number (or relative frequency) of extensions in which it participates. We call these figures **supports** and assume that they measure the degree of belief in the propositions represented by those values. (If the set of all solutions was assigned a uniform probability distribution, then the degree of support is precisely the marginal probability of the proposition, namely, its "belief" in the corresponding Bayes network [Pearl 1986] .) In particular, we say that a proposition is believed if it holds in all extensions (i.e., is **entailed** by the current set of formulas). The support figures for the possible values of each variable constitute a **support vector** for the variable.

A **dynamic Constraint-Network (DCN)** is a sequence of static CNs each resulting from a change in the preceding one, representing new facts about the environment being modeled. As a result of such an incremental change, the set of solutions of the CN may potentially decrease (in which case it is considered a **restriction**) or increase (i.e., a **relaxation**).

Restrictions occur when a new constraint is imposed on a subset of existing variables (e.g., forcing a variable to assume a certain value), or when a new variable is added to the system via some links. Restrictions always expand the model, i.e., they **add variables** and **add constraints** so that the associated constraint graph (representing the knowledge) grows monotonically.

Relaxations occur when constraints that were assumed to hold are found to be invalid and, therefore, may be removed from the network. However, it is not necessary to actually remove such constraints in order to cause the effect of relaxation. This can be achieved by modeling each potentially relaxable constraint in a special way which involves the inclusion of a bi-valued variable whose values indicate whether the constraint is "active" or not. Thus, we may assume, as is common in truth maintenance systems, that constraints that are added to the system are never removed.

In the next section we present an efficient scheme for propagating the information necessary for keeping all support vectors consistent with new external information.

3. Support Propagation in Trees

It is well known that a constraint network whose constraint graph is a tree can be solved easily [Freuder 1982, Dechter 1987]. Consequently, the number of solutions in which each value in the domain of each variable participates (namely, the support of this value), can also be computed very efficiently on such tree-structured networks. In this section we present a distributed scheme for calculating the support vectors for all variables, and for their updating to reflect changes in the network.

Consider a fragment of a tree-network as depicted in Figure 2.

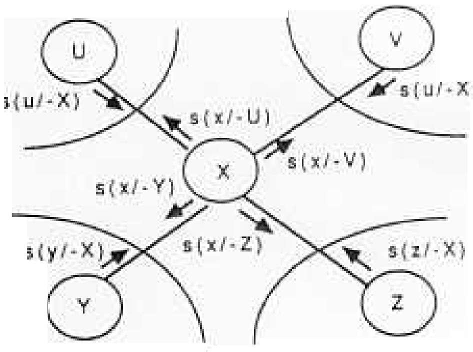

Figure 2: A fragment of a tree-structured CN

The link (X,Y) partitions the tree into two subtrees: the subtree containing X, $T_{XY}(X)$, and the subtree containing Y, $T_{XY}(Y)$. Likewise, the links (X,U), (X,V), and (X,Z), respectively, define the subtrees $T_{XU}(U)$, $T_{XV}(V)$ and $T_{XZ}(Z)$. Denote by $s_X(x)$ the overall support for value x of X, by $s_X(x/Y)$ the support for $X=x$ **contributed by subtree $T_{XY}(Y)$** (i.e., the number of extensions of this subtree which are consistent with $X=x$), and by $s_Y(y/\text{-}X)$ the support for $Y=y$ in $T_{XY}(Y)$. (These notations will be shortened to $s(x)$, $s(x/Y)$ and $s(y/-X)$, respectively, whenever the identity of the variable is clear.) The support for any value x of X is given by:

$$s(x) = \prod_{Y \in X\text{'s neighbors}} s(x/Y)\,, \qquad (1)$$

namely, it is a product of the supports contributed by each neighboring subtree. The support that Y contributes to $X = x$ can be further decomposed as follows:

$$s(x/Y) = \sum_{(x,y)\in C(X,Y)} s(y/-X)\,, \qquad (2)$$

where $C(X,Y)$ denotes the constraint between X and Y. Namely, since x can be associated with several **matching** values of Y, its support is the sum of the supports of these values. Equalities (1) and (2) yield:

$$s(x) = \prod_{Y\in X\text{'s neighbors}} \sum_{(x,y)\in C(X,Y)} s(y/-X)\,. \qquad (3)$$

Equation (3) lends itself to the promised propagation scheme. Suppose that variable X gets from each neighboring node, Y, a vector of restricted supports (referred to as **the support vector from Y to X**),

$$(s(y_1/-X), \ldots, s(y_t/-X))\,,$$

where y_i is in Y's domain. It can then calculate its own support. vector according to equation (3) and, at the same time, generate an appropriate message to each of its own neighbors. The message X sends to Y, $s(x/-Y)$, is the support vector reflecting the subtree $T_{XY}(X)$, and can be computed by:

$$s(x/-Y) = \prod_{Z\in X\text{'s neighbors}\,,\,Z\neq Y} \sum_{(x,z)\in C(X,Z)} s(z/-X)\,. \qquad (4)$$

The message generated by a leaf-variable is a vector consisting of zeros and ones representing, respectively, legal and illegal values of this variable.

Assume that the network is initially in a stable state, namely, all support vectors reflect correctly the constraints, and that the task is to restore stability when a new input causes a momentary instability. The updating scheme is initiated by the variable directly exposed to the new input. Any such variable will recalculate and deliver the support vector for each of its neighbors. When a variable in the network receives an update-message, it recalculates its outgoing messages, sends them to the rest of its neighbors, and at the same time updates its own support vector. The propagation due to a single outside change will propagate through the network only once (no feed-back), since the network has no loops. If the new input is a restriction, then it may cause a contradictory state, in which case all the nodes in the network will converge into all zero support vectors.

To illustrate the mechanics of the propagation scheme described above, consider again the problem of Figure 1. In Figure 3(a) the support vectors and the different messages are presented. The order within a support vector corresponds to the order of values in the originating variable, e.g., message (8,1) from X_3 to X_1 represents $(s_{X_3}(a/-X_1)\ ,\ s_{X_3}(b/-X_1)$. Suppose now that an assertion stating the value of $X_2 = b$ has arrived. In that case X_2 will originate a new message to X_3 of the form (0,1,0). This, in turn, will cause X_3 to update its supports and generate updated messages to X_1, X_4 and X_5 respectively. The new supports and the new updated messages are illustrated in Figure 3(b).

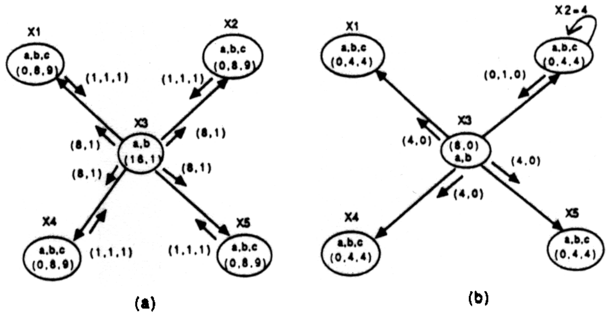

Figure 3: Support vectors before and after a change

If one is not interested in calculating numerical supports, but merely in indicating whether a given value has **some** support (i.e., participates in at least one solution), then **flat** support-vectors, consisting of zeros and ones, can be propagated in exactly the same manner, except that the summation operation in (3) should be replaced by the logic operator **OR**, and the multiplication can be replaced by **AND**.

4. Handling Assumptions and Contradictions

When, as a result of new input, the network enters a contradictory state, it often means that the new input is inconsistent with the **current set of assumptions**, and that some of these assumptions must be modified in order to restore consistency. We assume that certain variables of the network are designated as **assumption variables** which initially are assigned their default values, but may at any time assigned other values as needed. The task of restoring consistency by changing the values assigned to a subset of the assumption variables is called **contradiction resolution**.

The subset of assumption variables that are modified in a contradiction resolution process should be minimal, namely, it must not contain any proper subset of variables whose simultaneous modification is sufficient for that purpose (i.e., like the maximal assumption sets in [Doyle 1979]). A sufficient (but not necessary) condition for this set to be minimal is for it to be as small as possible. Other criteria for conflict resolution sets are suggested in [Petrie 1987]. In this section we show how to identify, in a distributed fashion, the minimum number of assumptions that need to be changed in order to restore consistency. Unlike the support propagation scheme, however, the contradiction resolution process has to be synchronized. Assume that a variable which detects a contradiction propagates this fact to the entire network, creating in the process a directed tree rooted at itself. Given this tree, the contradiction resolution process proceeds as follows.

With each value v of each variable V we associate a weight $w(v)$, indicating the minimum number of assumption variables that must be changed in the directed subtree rooted at V in order to make v consistent in this subtree. These weights obey the following recursion:

$$w(v) = \sum_{Y_i} \min_{(v,y_{ij})\in C(V,Y_i)} w(y_{ij})\,, \qquad (5)$$

where $\{Y_i\}$ are the set of V's children and their domain values are indicated by y_{ij}; i.e. y_{ij} is the j^{th} value of variable Y_i, (see Figure 4).

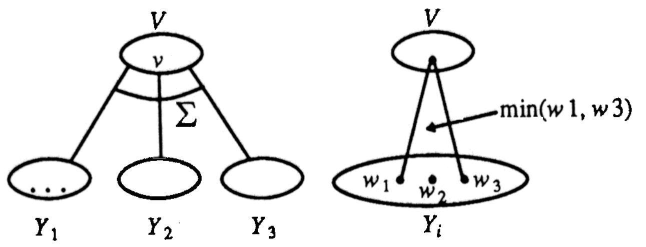

Figure 4: Weight calculation for node v

The weights associated with the values of each assumption variable are "0" for the value currently assigned to this variable, and "1" to all other possible values. For leaf nodes which are not assumption variables, the weights of their legal values are all "0". The computation of the weights is performed distributedly and synchronously from the leaves of the directed tree to the root. A variable waits to get the weights of all its children, computes its own weights according to (5), and sends them to its parent. During this **bottom-up-propagation** a pointer is kept from each value of V to the values in each of its child-variables, where a minimum is achieved. When the root variable X receives all the weights, it computes its own weights and selects one of its values that has a minimal weight. It then initiates (with this value) a **top-down propagation** down the tree, following the pointers marked in the bottom-up-propagation, a process which generates a consistent extension with a minimum number of assumptions changed. At termination this process marks the assumption variables that need to be changed and the appropriate changes required.

There is no need, however, to activate the whole network for contradiction resolution, because the support information available clearly points to those subtrees where no assumption change is necessary. Any subtree rooted at V whose support vector to its parent, P, is strictly positive for all "relevant" values, can be pruned. Relevance can be defined recursively as follows: the relevant values of V are those values which are consistent with some relevant value of its parent, and the relevant values of the root, X, are those which are not known to be excluded by any outside-world-change, independent of any change to the assumptions.

To illustrate the contradiction resolution process, consider the network given in Figure 5(a), which is an extension of the network of Figure 1 (the constraints are strict lexicographic order along the arrows). Variables X_1, X_6 and X_7 are assumption variables, with the current assumptions indicated by the unary constraints associated with them. The support messages sent by each variable to each of its neighbors are explicitly indicated. (The overall support vectors are not given explicitly.) It can be easily shown that the value a for X_3 is entailed and that there are 4 extensions altogether. Suppose now that a new variable X_8 and its constraint with X_3 is added (this is again a lexicographic constraint). The value a of X_8 is consistent only with value b of X_3 (see Figure 5(b)). Since the support for a of X_3 associated with this new link is zero, the new support vector for X_3 is zero and it detects a contradiction. Variable X_3 will now activate a subtree for contradiction resolution, considering only its value b as "relevant" (since value a is associated with a "0" support coming from X_8 which has no underlying assumptions). In the activation process, X_4 and X_5 will be pruned since their support messages to X_3 are strictly positive. X_1 will also be pruned since it has only one relevant value c and the support associated with this value is positive. The resulting activated tree is marked by heavy lines in Figure 5(b). Contradiction resolution of this subtree will be initiated by both assumption variables X_6 and X_7, and it will determine that the two assumptions $X_6 = c$ and $X_7 = c$ need to be replaced with assuming d for both variables (the process itself is not demonstrated).

Once contradiction resolution has terminated, all assumptions can be changed accordingly, and the system can get into a new stable state by handling those changes using support propagation. If this last propagation is not synchronized, the amount of message passing on the network may be proportional to the number of assumptions changed. If, however, these message updating is synchronized, the network can reach a stable state with at most two message passing on each arc. Figure 5(c) gives the new updated messages after the system stabilized.

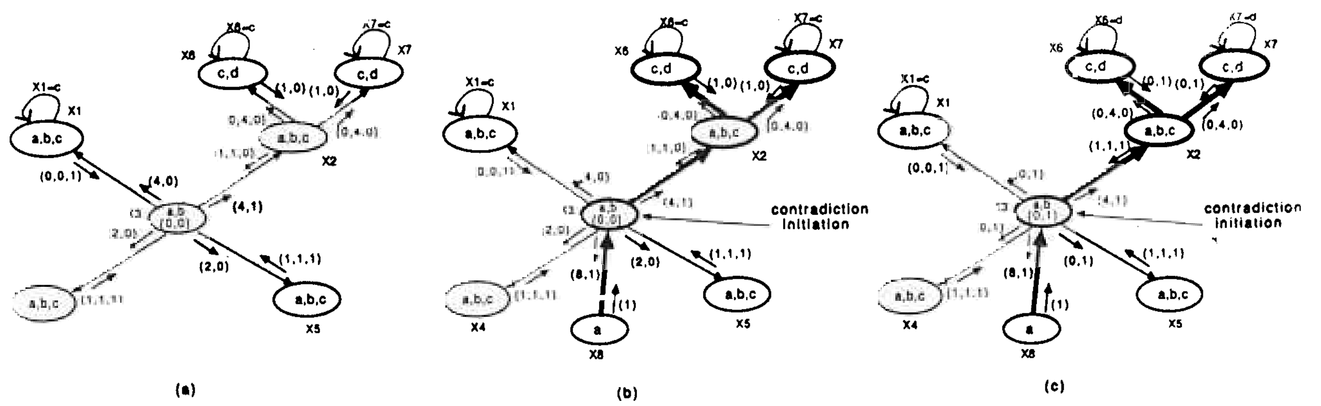

Figure 5: The contradiction resolution process

5. Support propagation in acyclic networks

The support propagation algorithm presented in Section 3 for tree-structured binary networks can be adapted for use with general, non-binary networks, whose **dual constraint graphs** are trees. The dual constraint graph can be viewed as the primal graph of an equivalent binary constraint network, where each of the constraints of the original network is a variable (called a c-variable) and the constraints call for equality of the values assigned to the variables shared by any two c-variables.

For example, Figure 6(a) depicts the dual constraint-graphs of a network consisting of the variables A,B,C,D,E,F, with constraints on the subsets (ABC),(AEF), (CDE), and (ACE) (the constraints themselves are not specified).

The graph of Figure 6(a) contains cycles. Observe, however, that the arc between (AEF) and (ABC) can be eliminated because the variable A is common along the cycle (AFE)-A-(ABC)-AC-(ACE)-AE-(AFE), so the consistency of the variable A is maintained by the remaining arcs. Similar arguments can be used to show that the arcs labeled C and E are redundant and may be removed as well, thus transforming the dual graph into a tree (Figure 6(b)).

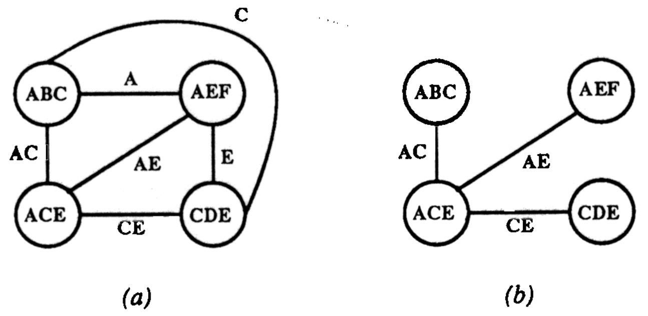

Figure 6: A dual constraint graph of a CSP

A constraint network whose dual constraint graph can be reduced to a tree is said to be **acyclic**. Acyclic constraint networks are an instance of **acyclic databases**, and the tree-structured dual constraint graph is a **join-tree** of the database (see, for example [Beeri 1983],).

Now, consider the fragment of a tree-structured dual constraint graph, whose nodes represent the constraints C, U_1,U_2,U_3, and U_4, given in Figure 7.

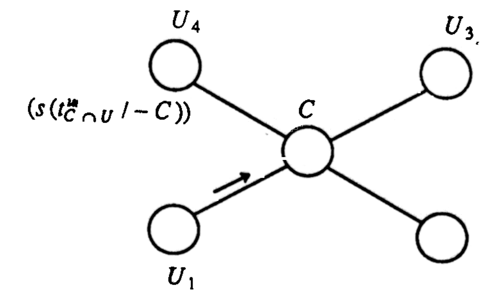

Figure 7: A fragment of a dual constraint graph

We denote by t^c an arbitrary tuple of C. With each tuple, t^c, we associate a support number $s(t^c)$, which is equal to the number of extensions in which all values of t^c participate. Let $s(t^c \mid U)$ denote the support of t^c coming from subtree $T_{CU}(U)$, and let $s(t^u \mid -C)$ denote the support for t^u restricted to subtree $T_{CU}(U)$ (we use the same notational conventions as in the binary case). The support for t^c is given by:

$$s(t^c) = \prod_{U \in C\text{'s neighbors}} s(t^c \mid U) . \qquad (6)$$

The support U contributes to t^c can be derived from the support it contributes to the projection of t^c on $C \cap U$, denoted by $t^u{}_{C \cap U}$, and this, in turn, can be computed by summing all the supports of tuples in U restricted to subtree $T_{CU}(U)$ that have the same assignments as t^c for variables in $C \cap U$. Namely:

$$s(t^c \mid U) = s(t^c{}_{C\cap U} \mid U) = \sum_{t^u{}_{C\cap U} = t^c{}_{C\cap U}} s(t^u \mid -C) . \qquad (7)$$

Equations (6) and (7) yield

$$s(t^c) = \prod_{U \in C\text{'s neighbors}} \sum_{t^u{}_{C\cap U} = t^c{}_{C\cap U}} s(t^u \mid -C) . \qquad (8)$$

The propagation scheme emerging from (8) has the same pattern as the propagation for binary constraints. Each constraint calculates the support vector associated with each of its outgoing arcs using:

$$s(t^u{}_{C\cap U} \mid -C) = \sum_{t^u{}_{C\cap U} = t^{u'}{}_{C\cap U}} s(t^{u'} \mid -C) . \qquad (9)$$

The message which U sends to C is the vector

$$(s(t^{u_i}{}_{C\cap U} \mid -C)) , \qquad (10)$$

where i indexes the projection of constraint U on $C \cap U$. Using this message, C can calculate its own support (using (8)) and will also generate updating messages to be sent to its neighbors.

Having the supports associated with each tuple in a constraint, the supports of **individual values** can easily be derived by summing the corresponding supports of all tuples in the constraint having that value.

Contradiction resolution can also be modified for acyclic networks using the same methodology. Support propagation and contradiction resolution take, on join-trees, the same amount of message passing as their binary network counterparts. Thus, the algorithm is linear in the number of constraints and quadratic in the number of tuples in a constraint (in fact, due to the special nature of the "dual constraints", being all equalities, the dependency of the complexity on the number of tuples t can be reduced from t^2 to $t\log t$, using an indexing technique).

An illustration of this process is provided in the full paper [Dechter 1988a] where an application of this technique to a circuit diagnosis problem is discussed.

6. Support Propagation in General Networks

When the constraint network is not acyclic, the method of tree-clustering [Dechter 1988b] can be used prior to application of the propagation schemes described above. This method uses aggregation of constraints into equivalent constraints involving larger cluster of variables in such a way that the resulting network is acyclic. In this, more general case, the complexity of the procedure depends on the complexity of solving a constraint satisfaction problem for each cluster and is exponential in the size of the larger cluster. For more details see [Dechter 1988b].

7. Summary and conclusions

We presented efficient algorithms for support propagation and for contradiction-resolution in acyclic dynamic constraint networks, and indicated how these algorithms can be extended for a general network using the tree-clustering method. The propagation scheme contains two components: support updating and contradiction resolution. The first handles non-contradictory inputs and requires one pass through the network. The second finds a minimum set of assumption-changes which resolve the contradiction. Contradiction resolution may take five passes in the worst case: activating a diagnosis subtree (one pass), determining a minimum assumption set (two passes) and updating the supports with new assumptions (two passes).

The belief-maintenance mechanism presented here is particularly useful for cases involving minor topological changes, for example, when observations arrive regarding the restriction of an existing constraint rather then the introduction of a new (non-unary) constraint. In such cases the structure of the acyclic network, which may be compiled initially via tree-clustering, does not change.

We do not consider this work to be a proposal for another TMS, since some basic assumptions currently obeyed by TMS developers are not followed here. TMSs try to model the reasoning process of a general knowledge-based system, where the knowledge part is purposely separated from the reasoning part. The TMS, whose input is provided by the reasoner, performs a limited amount of deduction, detects contradictions, and performs dependency-directed backtracking, keeping track of which assertions are assumptions and premises and which ones were deduced. Having this view, the TMS is normally not a complete inference procedure whose existence is justified by maintaining consistency within the explicated reasoning process in an efficient way.

Our view is different in that no separation is made between the knowledge and the reasoning process based on this knowledge. We provide a knowledge-base in its declarative form with a given amount of derivation already performed on it without keeping track of the derivation process. The dependencies in the knowledge-base are also declerative and undirectional. Our goal is to maintain the knowledge consistent and complete under possible changes coming from the outside world (e.g., observations). The dependency structure explicates dependencies in the knowledge structure and not in a particular reasoning path which is based on this knowledge (although these are related). Explanations are not an integral task, although they can be given a declerative definition and can be easily derived from the knowledge.

References

[Beeri 1983]Beeri, C., R. Fagin, D. Maier, and N. Yannakakis, "On the desirability of Acyclic database schemes," *JACM*, Vol. 30, No. 3, July, 1983, pp. 479-513.

[Dechter 1987]Dechter, R. and J. Pearl, "Network-based heuristics for constraint-satisfaction problems," *Artificial Intelligence*, Vol. 34, No. 1, December, 1987, pp. 1-38.

[Dechter 1988a]Dechter, R. and A. Dechter, "Belief maintenance in dynamic constraint networks," UCLA, Los Angeles, CA, Tech. Rep. R-108, February, 1988.

[Dechter 1988b]Dechter, R. and J. Pearl, "A Tree-Clustering Scheme for Constraint Processing," in *Proceedings AAAI-88*, St. Paul, MI: August 1988.

[de Kleer 1986]de Kleer, J., "An assumption-based TMS," *Artificial Intelligence*, Vol. 28, No. 2, 1986.

[Doyle 1979]Doyle, J., "A truth maintenance system," *Artificial Intelligence*, Vol. 12, 1979, pp. 231-272.

[Freuder 1982]Freuder, E.C., "A sufficient condition of backtrack-free search.," *Journal of the ACM*, Vol. 29, No. 1, January 1982, pp. 24-32.

[Mackworth 1977]Mackworth, A.K., "Consistency in networks of relations," *Artificial intelligence*, Vol. 8, No. 1, 1977, pp. 99-118.

[McAllester 1980]McAllester, D.A., "An Outlook on Truth-Maintenance," MIT, Boston, Massachusetts, Tech. Rep. AI Memo No. 551, August, 1980.

[Montanari 1974]Montanari, U., "Networks of constraints: fundamental properties and applications to picture processing," *Information Science*, Vol. 7, 1974, pp. 95-132.

[Pearl 1986]Pearl, J., "Fusion Propagation and structuring in belief networks," *Artificial Intelligence Journal*, Vol. 3, September 1986, pp. 241-288.

[Petrie 1987]Petrie, C.J., "Revised Dependency-Directed Backtracking for Default Reasoning," in *Proceedings AAAI-87*, Seattle, Washington: July, 1987, pp. 167-172.

[Reiter 1987]Reiter, R., "A Logic for Default Reasoning," in *Reading in Nonmonotonic Reasoning*, M.L. Ginsberg, Ed. Los Altos, Cal.: Morgan Kaufman, 1987, pp. 68-93.

[Waltz 1975]Waltz, D., "Understanding line drawings of scenes with shadows," in *The Psychology of Computer Vision*, P.H. Winston, Ed. New York, NY: McGraw-Hill Book Company, 1975.