Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

BELIEF MAINTENANCE IN DYNAMIC CONSTRAINT NETWORKS

Rina Dechter
Avi Dechter

July 1988
CSD-880056

TECHNICAL REPORT
CSD 8700xx
R-108-L
March 1988

BELIEF MAINTENANCE IN DYNAMIC CONSTRAINT NETWORKS

Rina Dechter

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

and

Avi Dechter

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

Net address: dechter@cs.ucla.edu
Phone: (213) 825-3243

Length: 4000 words
Topic area: Automated Reasoning
Keywords: belief revision, constraint-networks, diagnosis.

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 it. A proposition is said to be believed if it receives a non-zero support and its negation receives zero support. 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 that detects a contradiction, and identifies a minimal set of assumptions that potentially account for the contradiction. The support propagation phase is accomplished in a single pass through the network, while the contradiction resolution process requires at most 4 passes. Thus, the impact of any new input to the system can be propagated in at most 5 passes through the network. Extensions of the scheme to general, non-acyclic networks, using a clustering approach, are also discussed.

*This work was supported in part by the National Science Foundation, Grant #DCR 85-01234

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 [Reiter1987, Doyle1979, De-Kleer1986b].

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 has the expressive power of propositional calculus and was mainly 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 [Montanari1974, Waltz1975]). A substantial body of knowledge for solving such problems has been developed [Montanari1974, Mackworth1977, Freuder1982, Dechter1987b].

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. [Stallman1977, McAllester1980, Doyle1979]. The use of constraint networks as the framework for modeling the task of dynamic belief management, allows us to develop 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 remainder of 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. An example taken from the area of electronic circuit diagnosis is worked out in Section 6. Section 7 discusses the extension of the algorithm to general networks, and Section 8 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 constraints are given. Consider, for instance, the CN presented in Figure 1(a) (modified from [Mackworth1977]). 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.)

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

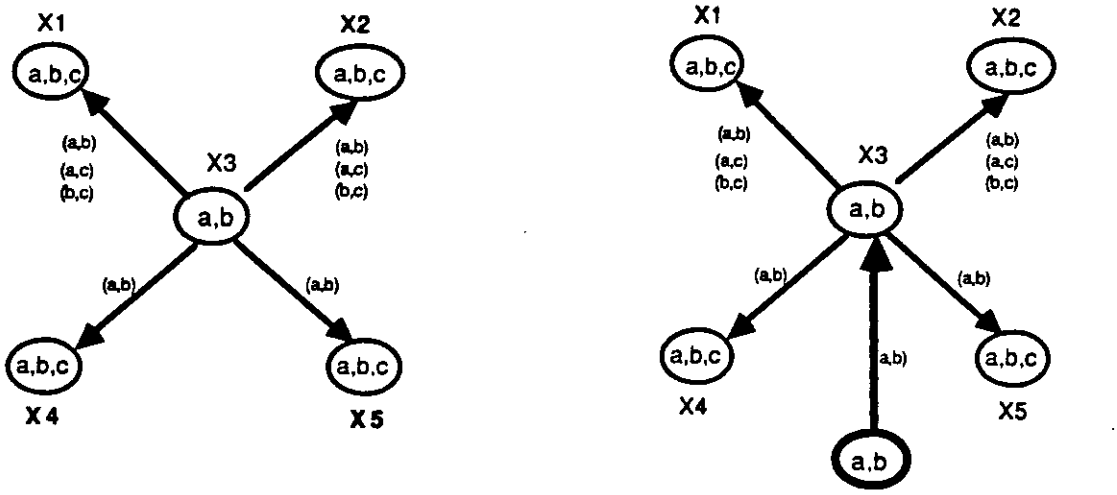

Figure 1: An example of a binary CN

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. 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. Figure 1(b) shows a change occurring in the model of Figure 1a by adding variable X_6 and its associated (lexicographic) constraint. 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 (this modeling technique is used in the example of Section 6). 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 constraint networks whose constraint graph is a tree can be solved easily [Freuder1982, Dechter1985]. 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-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. 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** $\mathbf{T_{XY}(Y)}$ (i.e., the number of extensions of this subtree which are consistent with $X = x$), and by s_Y(y/-X) the support for $Y = y$ in $T_{XY}(Y)$. (These notations will

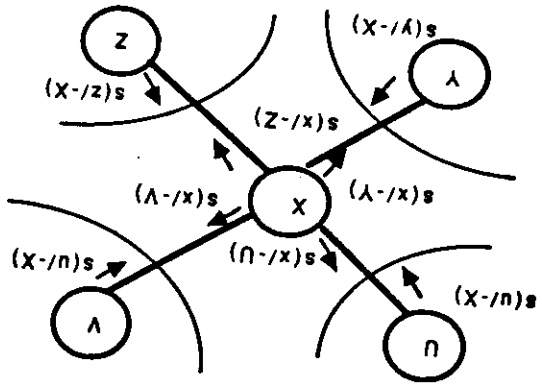

Figure 2: A fragment of a tree-structured CN

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:

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

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:

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

when $C(X,Y)$ 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:

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

Equation (3) lends itself to the promised propagation scheme. If 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 calculates 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:

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

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(a). In Figure 3a 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, namely, 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 the system is forced by an outside change to restrict the value of X_2 to "b". 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).

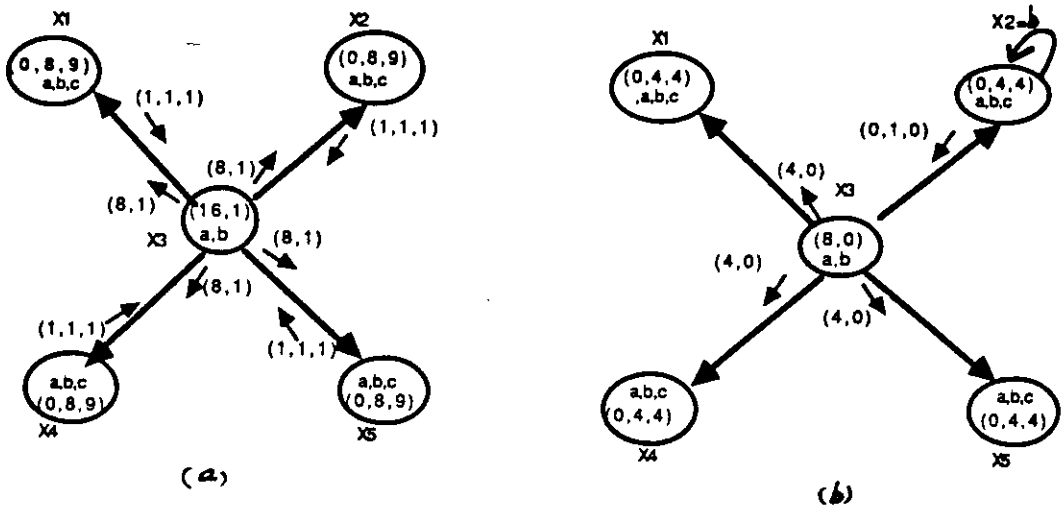

Figure 3: Support vectors before and after a change

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 way, 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. A sufficient (but not necessary) condition for this set to be minimal is for it to be as small as possible. 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:

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

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).

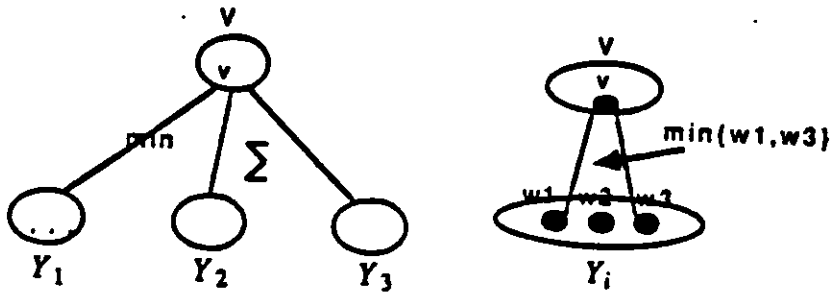

Figure 4: Weight calculation for node v

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. Relevancy 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 to the network in Figure 1(a) (the constraint 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).

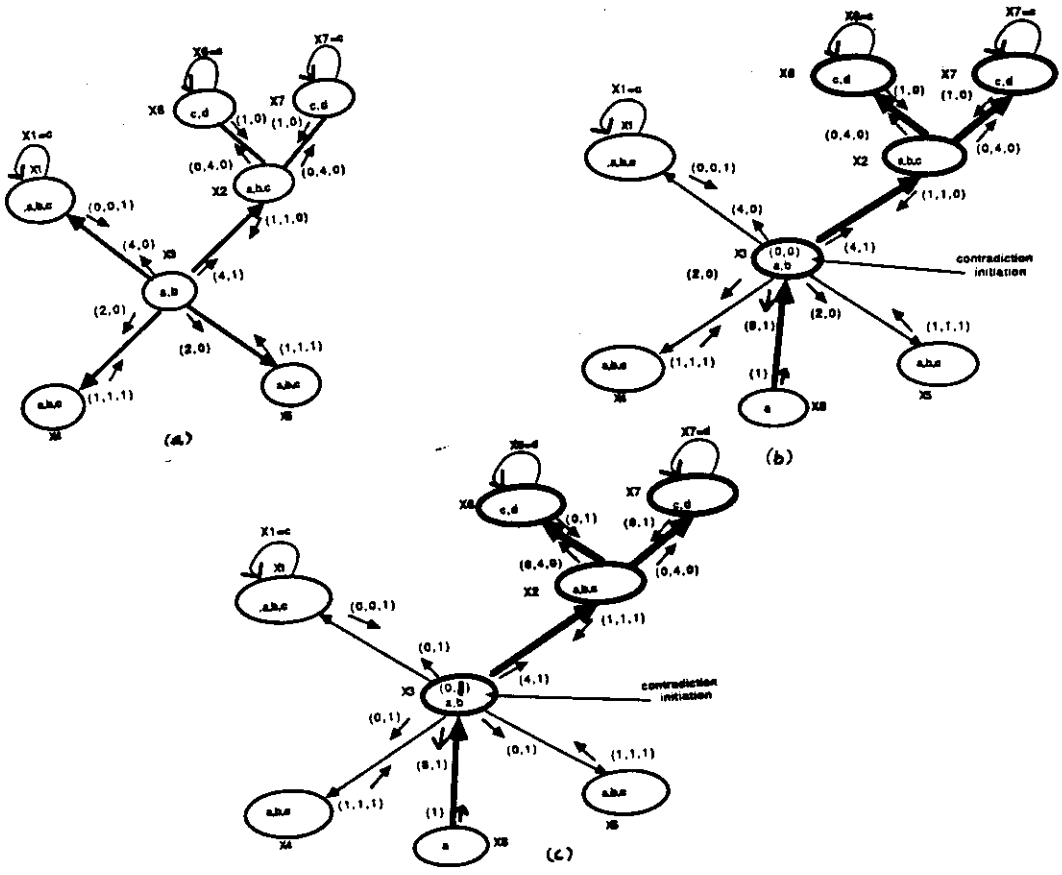

Figure 5: Illustration of the contradiction resolution process

Once contradiction resolution had been 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 had been stabilized.

5. Support propagation in acyclic networks

Acyclic constraint networks extend the notion of a tree-structured constraint network to networks with constraints of higher arity. The equivalent of the constraint graph of binary network in the more general case, called the **primal constraint graph**, consists of a node for each variable and an arc for each two variables related directly by at least one constraint. Alternatively, the network may be represented by a **dual constraint graph**, consisting of a node for each **constraint** and an arc for any two constraints that share at least one variable. The dual constraint graph can be viewed as the primal graph of an equivalent 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, Figures 6(a) and 6(b) depict, respectively, the primal and 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).

Since all the constraints in the dual representation are equalities, any cycle for which all the arcs share a common variable contains redundancy, and thus any arc such that each of the variables in its label is a common variable in some cycle may be removed from the network. The graph remaining after all such arcs have been removed is called a **join-graph**, and its

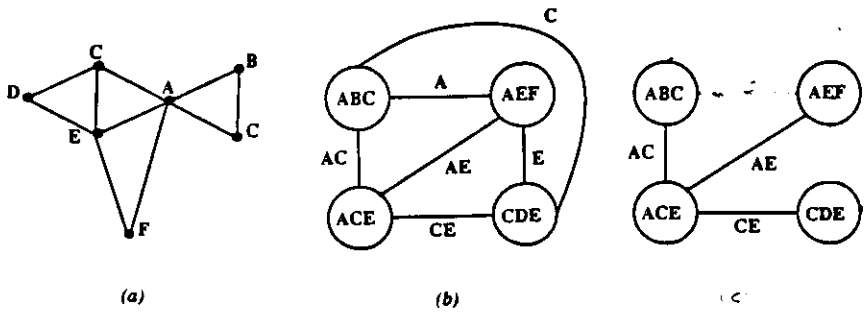

Figure 6: A primal and dual constraint graphs of a CSP

corresponding network is equivalent to the original network.

For example, in Figure 6(b), 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 A variables is maintained by the remaining arcs. Similar arguments can be used to show that the arcs labeled C and E may be removed as well, thus transforming the dual graph into a **join-tree** (see Figure 6(c)).

A Constraint network whose dual constraint graph can be reduced to a join-tree is said to be **acyclic**. Acyclic constraint networks are an instance of **acyclic data bases** discussed at length in [Beeri1983].

With this background in mind, it is easy to show how the support propagation algorithm presented in the previous section for tree-structured binary networks can be adapted for use in acyclic networks. This requires that the network be transformed to its "dual representation" and that a joint-tree is identified. We outline the algorithm next.

Consider the fragment of a join-tree, whose nodes represent the constraints C, U_1, U_2, U_3, U_4, given in Figure 7.

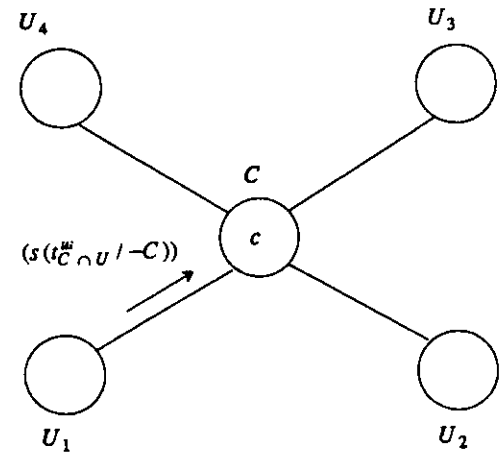

Figure 7: A fragment of a join-tree

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 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:

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

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:

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

Equations (6) and (7) yield

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

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

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

The message which U sends to C is the vector

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

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 join-trees using the same methodology. This process will be illustrated in the next section where these algorithms are demonstrated on a circuit diagnosis example. 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).

6. A Circuit Diagnosis Example

An electronic circuit can be modeled in terms of a constraint network by associating a variable with each input, output, intermediate value, and device. Device variables are bi-valued, having the value "0" if functioning correctly and the value "1" otherwise. There is a constraint associated with each device, relating the device variable with its immediate inputs and outputs. Given input data, the possible values of any intermediate variable or output variable is its "expected value", namely, the value that would have resulted if all devices worked correctly, or some ''unexpected value'' denoted by "e". A variable may have more then one expected value.

Consider the circuit of Figure 8 (also discussed in [De-Kleer1986a, Davis1984, Genesereth1984]), consisting of three multipliers, M_1,M_2,M_3, and two adders, A_1 and A_2. The values of the five input variables, A, B, C, D, and F, and of the two output variables, F and G, are given. The numbers in the brackets are the expected values of the three intermediate points X, Y, and Z, and of the outputs. The relation defining the constraint associated with the multiplier M_1 is given in Figure 9 as an example. Given the inputs and outputs of the circuit, the objective is to identify a minimal set of devices which, if presumed to be malfunctioning, could explain the observed behavior.

The dual graph of the constraint network corresponding to this circuit is given in Figure 10. This network is acyclic, as is evident by the fact that a join-tree can be obtained by eliminating the redundant arc (marked by a dashed line) between constraint (M_2,B,D,Y) and (A_2,Z,Y,G).

Initially, when no observation of output data is available, the network propagates its support numbers assuming all device variables have their default value "0". In this case only one solution exists and therefore the supports for all consistent values are "1" (the support propagation algorithm is not illustrated). The diagnosis process is initiated when the value "10" is

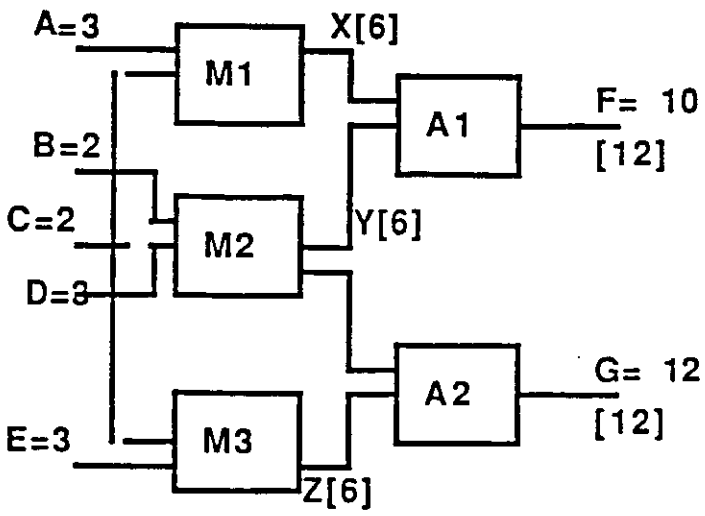

Figure 8: A circuit example

M_1	A	C	X	
0	2	3	6	$w=0$
1	2	3	e	$w=1$

Figure 9: A multiplier constraint

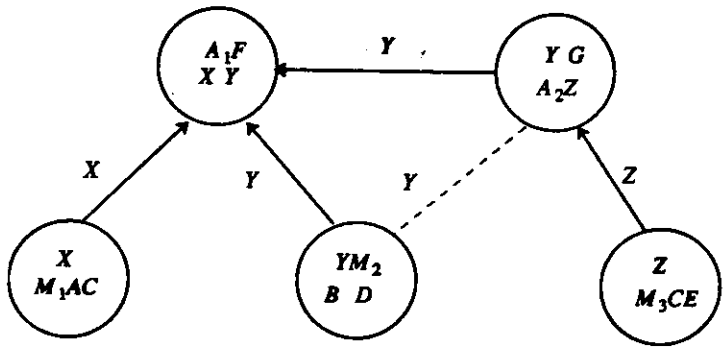

Figure 10: An acyclic constraint network of the circuit example

observed for variable F which is different from the expected value of 12. The value "10" is fixed as the only consistent value of F. At this point, the constraint (X,A_1,F,Y), which is the only one to contain F, induces direction on the join-tree, resulting in the directed tree (rooted at itself) of Figure 11, and contradiction resolution is initiated.

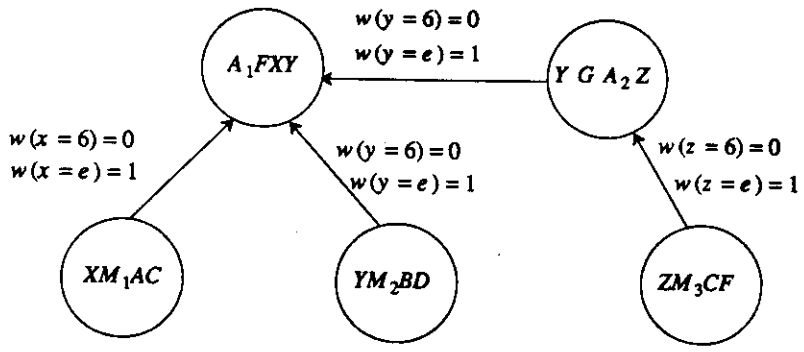

Figure 11: Weight calculation for the circuit example

Each tuple will be associated with the minimum number of assumption changes in the subtree underneath it, and the c-variable will project the corresponding weights on the variables which label its outgoing arc. In Figure 11 the weights associated with the arcs of the three leaf constraints (i.e., the multipliers constraints), are presented. They are derived from the weights associated with their incoming constraints (see the weights in Figure 9). For instance, the weights associated with X is $w(X = 6)=0$ since "6" is the expected value of X when M_1 works correctly (which is the default assumption), and $w(X = e) = 1$ since, any other value can be expected only if the multiplier is faulty. Next, the weights propagate to constraint (Y,G,A_2,Z). This constraint and its weights are given in Figure 12 (note, that G's observed value is 12). The corresponding derived Y's weights are indicated on the outgoing arc of constraint (Y,G,A_2,Z) in Figure 11. Finally, the weights associated with the root constraint (A_1,X,Y,F) are computed by summing the minimum weights associated with each of its child node. The tuples associated with the root

A_2	Z	G	Y	Weights	Faulty Devices
0	6	12	6	$w=0$	none
0	e	12	e	$w=1$	M_3
1	6	12	e	$w=1$	A_2
1	e	12	e	$w=2$	M_3 & A_2

Figure 12: The weights of constraint (Y,G,A_2,Z)

constraint and their weights are presented in Figure 13.

	A_1	F	X	Y	Weights	Faulty Devices
1.	0	10	6	e	2	$(M_3 \vee A_2)$ & M_2
2.	0	10	4	6	1	M_1
3.	0	10	e	e	3	M_1 & M_2 & $(M_3 \vee A_2)$
4.	1	10	6	6	1	A_1
5.	1	10	6	e	3	A_1 & M_2 & $(M_3 \vee A_2)$
6.	1	10	e	e	4	A_1 & M_2 & M_1 & $(M_3 \vee A_2)$

Figure 13: The weights of constraint (A_1,F,X,Y) (the root)

We see that the minimum weight is associated with tuples (2), indicating M_1 as faulty or (4), indicating A_1 as faulty. Therefore, either A_1 or M_1 are faulty.

This example illustrates the efficiency of the contradiction resolution process when the special structure of the problem is exploited. By contrast, handling this problem using ATMS [De-Kleer1986b] exhibits exponential behavior. For more details see [Geffner1987, Dechter1986.].

7. Support Propagation in General Networks

When the constraint network is not acyclic, the method of tree-clustering [Dechter1987a] 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 roughly exponential in the size of the larger cluster. For more details see [Dechter1987a].

8. Summary and conclusions

We presented an efficient propagation 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 computational difficulties associated with the presence of loops are not unique to our constraint-network formulation but are inherent to the basic problem of consistency maintenance. It will appear, under various disguises, in any formalism. The importance of network representation, however, is that it identifies the core of these difficulties, estimates the expected complexity, and provides a unifying theoretical underpinning that encourages the exchange of strategies across domains.

References

[Beeri1983] Beeri, Catriel, Ronald Fagin, David Maier, and Nihalis Yanakakis, "On the desirability of Acyclic database schemes," *JACM*, Vol. 30, No. 3, July, 1983, pp. 479-513.

[Davis1984] Davis, R., "Diagnostic Reasoning based on structure and behaviour," *AI Journal*, Vol. 24, 1984.

[Dechter1985] Dechter, R. and J. Pearl, "The anatomy of easy problems: a constraint-satisfaction formulation," in *Proceedings Ninth International Conference on Artificial Intelligence*, Los Angeles, Cal: August 1985, pp. 1066-1072.

[Dechter1986.] Dechter, R., "Diagnosis with the CSP model," UCLA, Cognitive systens lab, Los Angeles, California, Tech. Rep. R-72, Novemeber, 1986..

[Dechter1987a] Dechter, R. and J. Pearl, "A tree-clustering scheme for Constraint-Processing," UCLA, Cognitive-Systems Lab., Los Angeles, Cal., Tech. Rep. R-92, August 1987.

[Dechter1987b] Dechter, R. and J. Pearl, "Network-based heuristics for constraint-satisfaction problems.," UCLA, L.A. Cal., May, 1987. To be published at the AI-Journal.

[De-Kleer1986a] De-Kleer, J. and B. Williams , "Reasoning about Multiple-Faults," in *Proceedings AAAI-86, ,* Phila, PA.: 1986, pp. 132-139.

[De-Kleer1986b] De-Kleer, J., "An assumption-based TMS," *Artificial Intelligence,* Vol. 28, No. 2, 1986.

[Doyle1979] Doyle, J., "A truth maintenance system," *Artificial Intelligence,* Vol. 12, 1979, pp. 231-272.

[Freuder1982] Freuder, E.C., "A sufficient condition of backtrack-free search.," *Journal of the ACM*, Vol. 29, No. 1, January 1982, pp. 24-32.

[Geffner1987] Geffner, H. and J. Pearl, "An improved constraint-propagation algorithm for diagnosis," in *Proceedings Ijcai*, Milano, Italy: August, 1987.

[Genesereth1984] Genesereth, M.R., "The use of design descriptions in autonated diagnosis," *AI journal*, Vol. 24, 1984, pp. 411-436.

[Mackworth1977] Mackworth, A.K., "Consistency in networks of relations," *Artifficial intelligence*, Vol. 8, No. 1, 1977, pp. 99-118.

[McAllester1980] McAllester, David A., "An Outlook on Truth-Maintenance," MIT, Boston, Massachusetts, Tech. Rep. AI Memo No. 551, August, 1980.

[Montanari1974] Montanari, U., "Networks of constraints :fundamental properties and applications to picture processing," *Information Science,* Vol. 7, 1974, pp. 95-132.

[Reiter1987] 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.

[Stallman1977] Stallman, R.M. and G. J. Sussman, "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis," *Artificial Intelligence,* Vol. 9, No. 2, October 1977, pp. 135-196.

[Waltz1975] 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.