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

CONSTRAINT-BASED BELIEF MAINTENANCE AND ITS APPLICATION TO DIAGNOSIS

Rina Dechter
Avi Dechter

June 1989
CSD-890032

From Proceedings, The Fifth Israeli Symposium on Artificial Intelligence, Vision and Pattern Recognition, December 1988, pp. 2-15.

TECHNICAL REPORT
CSD-890032
R-120
December 1988

CONSTRAINT-BASED BELIEF MAINTENANCE AND ITS APPLICATION TO DIAGNOSIS*

Rina Dechter

Computer Science Department
Technion -- Israel Institute of Technology
Haifa, Israel 32000

and

Avi Dechter

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

Net address: dechter@cs.ucla.edu or Dechter@Techsel.bitnet

ABSTRACT

This paper presents a summary of a constraint-based formulation of belief maintenance system. We define belief in a proposition as the number of solutions of the constraint networks with which it is consistent. The paper outlines a distributed scheme for calculating and revising beliefs in acyclic constraint networks and show its applicability to diagnostic tasks. 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. By utilizing the network's topology, more efficient algorithms for performing these tasks are introduced. 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 and by the airforce office of scientific research, grant #AFSOR-88-0177

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 1986b], and they are referred to as **Belief-Maintenance-systems** or **BMS** for short.

In [Dechter 1988c] we showed that **constraint networks** and their associated **constraint satisfaction problems** provide an attractive paradigm for modeling dynamically changing environments. Constraint networks have the expressive power of propositional calculus and were traditionally used for expressing **static** problems, i.e., that require a one-time solution. (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] and some of its techniques are already utilized in current TMSs, e.g., dependency-directed backtracking, constraint propagation, etc.

Constraint networks is an area where the effects of the problem's topology on its tractability were studied extensively. Exploiting this topological knowledge when using constraint networks for BMS gives rise to efficient maintenance and query processing algorithms. Moreover, the performance and complexity of these algorithms can be analyzed and predicted in advance, a theoretical treatment which is usually not available in current TMS research.

The remainder of the paper is organized as follows. Section 2 reviews the constraint network model and extends it to handle belief maintenance tasks. Sections 3, 4 and 5 summarize the processing algorithms presented at [Dechter 1988c]. These algorithms, which consist of two phases: **support propagation** and **contradiction resolution**, are presented first for singly connected binary constraint networks (sections 3 and 4). Section 6 demonstrates application to circuit diagnosis, 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** (also called **primal 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 (modified from [Mackworth 1977]). Each node represents a variable whose values are explicitly indicated, and a link represents 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.)

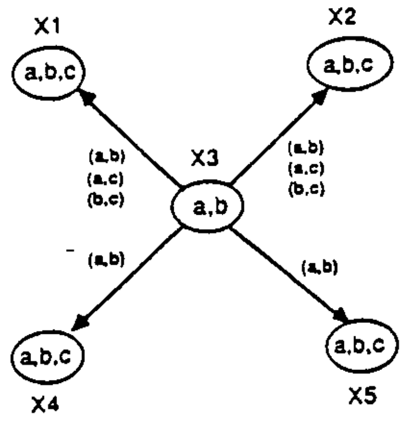

Figure 1: An example of a binary CN

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. For BMS purposes we focus on a related problem, that of finding, for each value in the domain of a certain variables, the number (or relative frequency) of extensions in which it participates. We call these figures **supports** and consider them as measuring the degree of belief in the propositions represented by those values. (If the set of all solutions was assigned a uniform probability distribution, the support will represent the "belief" in an associated 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**).

The primary purpose of a BMS is to reason with uncertain and incomplete information. In order to make inference in such environments, some of the knowledge is "completed" by making additional assumptions which are consistent with the current knowledge. When, due to new information, the knowledge becomes inconsistent, some of these assumptions have to be retracted. We model assumptions in constraint network by annotating some variables as **assumption variables** and giving these variables a special status within the processing algorithms. The possible values of assumption variables will be called **assumptions**.

Assumption variables can be used to model default rules: e.g. birds fly (unless they are dead), an adder functions correctly (unless it is faulty). This is accomplished by adding to the constraint representing the rule an assumption variable with two values: one representing the default assumption (e.g., that the rule "birds fly" is true) and the other representing the exception. For example, we can model "birds fly" with three variables representing the propositions "birds" "fly" and "birds fly", each with possible values {t,f}, where the last variable is an assumption variable whose default assumption is "t". The constraint is given by the following table:

birds	fly	birds fly
t	t	t
f	t	t
f	f	t
t	f	f

Technically, an assumption variable can be used as an enabling/disabling devise for any piece of information. If it is undesirable to eliminate a constraint which is no longer valid then by conjoining the constraint with an assumption variable and activating the disabling assumption, the same effect will be achieved.

3. Support Propagation in Trees

It is well known that constraint networks whose constraint graph is a tree can be solved easily [Freuder 1982, Dechter 1987]. Consequently, the number of solutions in which each value participates (namely, the support of this value), can also be computed very efficiently on trees and these computations can be performed distributedly.

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** $T_{XY}(Y)$ (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 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:

$$s(x) = \prod_{Y \in X\text{'s neighbors}} \; \sum_{(x,y) \in C(X,Y)} s(y/-X) . \qquad (1)$$

when $C(X,Y)$ denotes the constraint between X and Y. Namely, it is a product of supports contributed by each neighboring subtree. These respective supports are represented by the sums in the expression. Equation (1) 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**):

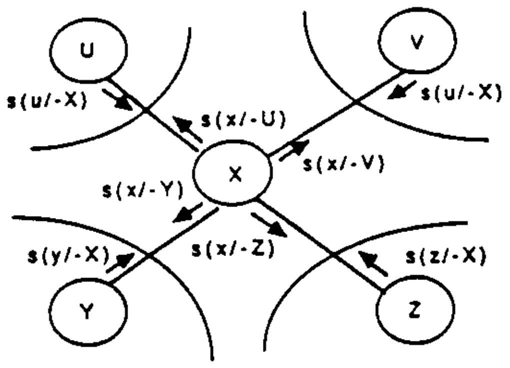

Figure 2: A fragment of a tree-structured CN

$$(s(y_1/{-X}), \ldots, s(y_i/{-X})), \tag{2}$$

where y_i is in Y's domain, it can calculates its own support vector according to equation (1) 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:

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

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 network's information. The maintenance task is to restore this 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 from a neighbor, 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 zero support vectors. In this updating process assumption variables play the same role as regular variables.

If an application does not require numerical supports, but only needs an indication 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 operations should be replaced by the logic operator

OR, and the multiplications 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. The task of restoring consistency by changing some assumptions 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 summarize the contradiction resolution process which identifies, in a distributed fashion a 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 regular 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:

$$w(v) = \sum_{Y_i} \min_{(v, y_{ij}) \in C(V, Y_i)} w(y_{ij}) \,, \tag{4}$$

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 3).

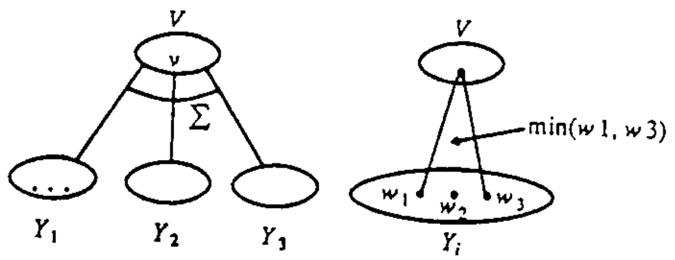

Figure 3: Weight calculation for node v

The weights associated with the values of each leaf 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". Non-leafs assumption variables should modify their weight calculation and add 1 the sum in equation (4) for any of their non-assumed values. 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 (4), 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 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. For details see [Dechter 1988c]. In section 6 we will illustrate the contradiction resolution process for circuit diagnosis.

5. Support propagation in acyclic networks

Acyclic constraint networks extend the notion of a tree-structured binary constraint network to networks with constraints of higher arity. A general 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. Thus, The dual constraint graph give rise to an equivalent binary constraint network, where variables are the constraints of the original network (called a c-variable), their values are their legal tuples and the constraints call for equality of the values assigned to the variables shared by any two c-variables.

For example, Figures 4(a) and 4(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).

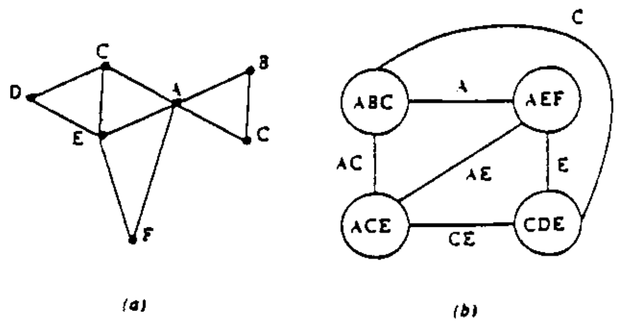

Figure 4: A primal and dual constraint graphs of a CSP

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 corresponding network is equivalent to the original network. For example, in Figure 4(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 4(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 [Beeri 1983].

The support propagation algorithm presented for tree-structured binary networks can be adapted for use in acyclic networks using one of their join-trees. 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 5.

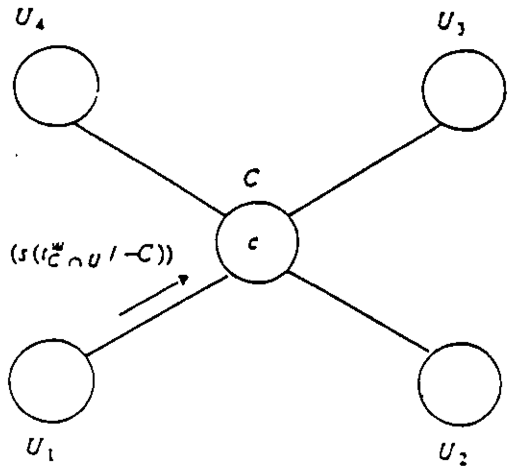

Figure 5: A fragment of a join-tree

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 $s(t^c | U)$ denote the support of t^c coming from subtree $T_{CU}(U)$, and let $s(t^u | -C)$ 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:

$$s(t^c) = \prod_{U \in C\text{'s neighbors}} \sum_{t^u_{C \cap U} = t^c_{C \cap U}} s(t^u | -C) . \tag{5}$$

The sums represents the support each neighbor, U, contributes to t^c. The propagation scheme emerging from (5) has the same pattern as the propagation for binary constraint.

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, on join-trees, are linear in the number of constraints and its dependency on the number of tuples t is $t \log t$ (reduced from t^2 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. Devices are modeled as bi-valued assumption variables, having the default 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. For the purpose of this example we assume that the set of expected values for each variable were determined by some pre-processing and all the other values are marked by the symbol "e".

Consider the circuit of Figure 6 (also discussed in [de_Kleer 1986a, Davis 1984, Genesereth 1984]), 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 7 as an example, as well as the initial weights associated with the tuples of these leaf constraints. 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.

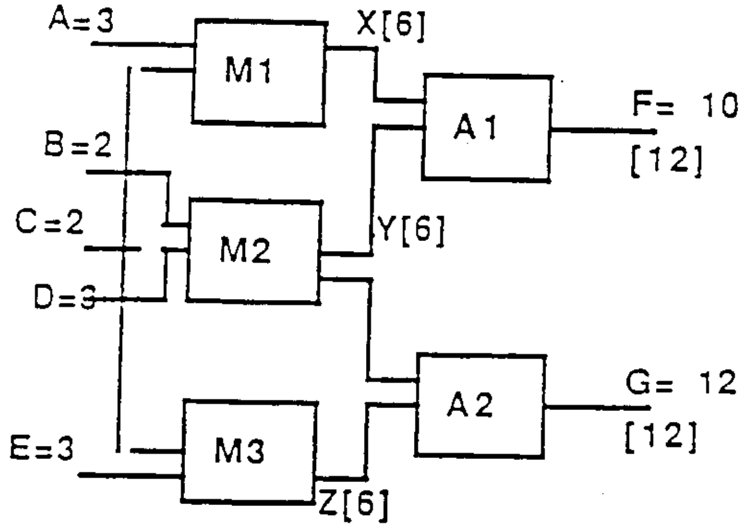

Figure 6: A circuit example

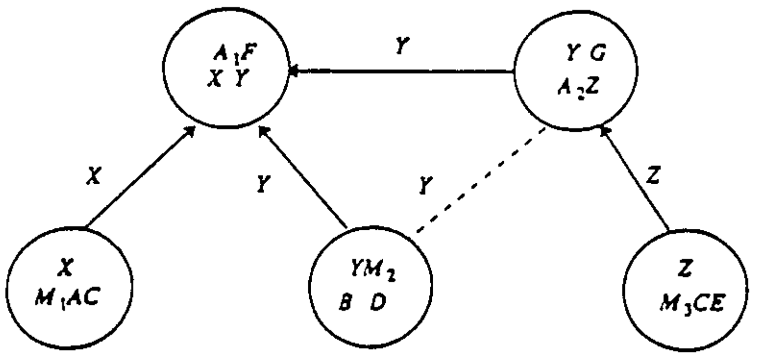

Figure 7: An acyclic constraint network of the circuit example

The dual graph of the constraint network corresponding to this circuit is given in Figure 7. 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). The relation defining the constraint associated with the multiplier M_1 is given in Figure 8 as an example, as well as the initial weights associated with the tuples of these leaf constraints.

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

Figure 8: A multiplier constraint

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 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 9, and contradiction resolution is initiated. 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

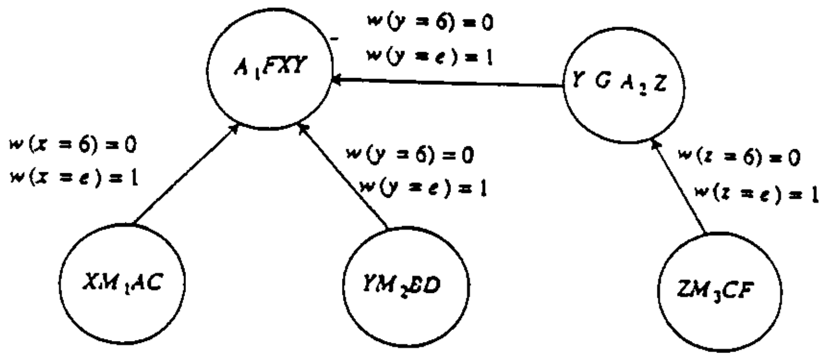

Figure 9: Weight calculation for the circuit example

weights on the variables which label its outgoing arc. In Figure 9 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 8). 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 10 (note, that G's observed value is 12).

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

The corresponding derived Y's weights are indicated on the outgoing arc of constraint (Y,G,A_2,Z) in Figure 9. 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 constraint and their weights are presented in Figure 11. 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. The weights can also be used as a guide for additional measurement taking in order to delineate between the different diagnoses.

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

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_Kleer 1986b] exhibits exponential behavior. See also [Geffner 1987]. In a similar manner a propagation scheme can be devised to extract all minimal diagnoses (not necessarily the minimum-cardinality one [Dechter 1988a]) for further processing by some diagnostic package.

7. Summary and conclusions

This paper presents a summary of a constraint-based Belief Maintenance System that is composed of two maintenance algorithms, one for support propagation and the other for contradiction-resolution. Both algorithms are restricted to acyclic constraint networks for which they are very efficient.

When the constraint network is not acyclic, the method of tree-clustering [Dechter 1988b] can be used in a pre-processing mode. This method uses aggregation of constraints into equivalent constraints involving larger clusters of variables in such a way that the resulting network is acyclic. The complexity of the clustering scheme is exponential in the size of the larger cluster [Dechter 1988b].

The applicability of our BMS is particularly useful for cases involving minor topological changes, namely 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.

References

[Beeri 1983]	Beeri, Catriel, Ronald Fagin, David Maier, and Nihalis Yannakakis, "On the desirability of Acyclic database schemes," *JACM*, Vol. 30, No. 3, 1983, pp. 479-513.
[Davis 1984]	Davis, R., "Diagnostic Reasoning based on structure and behavior," *AI Journal*, Vol. 24, 1984.
[Dechter 1987]	Dechter, R. and J. Pearl, "Network-based heuristics for constraint-satisfaction problems," *Artificial Intelligence journal*, Vol. 34, No. 1, 1987, pp. 1-38.
[Dechter 1988a]	Dechter, R., "Constraint-Directed Approach to Diagnosis," UCLA, Cognitive systems Lab, Los Angeles, California, Tech. Rep. R-72-I, 1988.
[Dechter 1988b]	Dechter, R. and J. Pearl, "Tree-clustering schemes for constraint-processing," in *Proceedings AAAI-88*, St Poul, Minneapolis: 1988.
[Dechter 1988c]	Dechter, R. and A. Dechter, "Belief maintenance in dynamic constraint networks," in *Proceedings AAAI-88*, St. Paul, Minnesota: 1988.
[de_Kleer 1986a]	de_Kleer, J. and B. Williams , "Reasoning about Multiple-Faults," in *Proceedings AAAI-86*, , Phila, PA.: 1986, pp. 132-139.
[de_Kleer 1986b]	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, 1982, pp. 24-32.
[Geffner 1987]	Geffner, H. and J. Pearl, "An improved constraint-propagation algorithm for diagnosis," in *Proceedings Ijcai*, Milano, Italy: 1987.
[Genesereth 1984]	Genesereth, M.R., "The use of design descriptions in automated diagnosis," *AI journal*, Vol. 24, 1984, pp. 411-436.
[Mackworth 1977]	Mackworth, A.K., "Consistency in networks of relations," *Artificial intelligence*, Vol. 8, No. 1, 1977, pp. 99-118.

[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, 1986, pp. 241-288.

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

[Reiter 1987] Reiter, R., ''A Logic for Default Reasoning,'' in *Reading in Non-monotonic 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.