A CONSTRAINT-NETWORK APPROACH TO TRUTH-MAINTENANCE

Rina Dechter

February 1987
CSD-870009

TECHNICAL REPORT
R-80
February, 1987

A CONSTRAINT-NETWORK APPROACH TO TRUTH-MAINTENANCE

Rina Dechter

Artificial Intelligence Center
Hughes Aircraft Company
and
Cognitive Systems Laboratory, Computer Science Department
University of California, Los Angeles
Net address: dechter@locus.ucla.edu
Tel: (213) 825-3243

Track: Science track
Topic area: Automated Reasoning.
Keywords: belief revision, constraint-networks, diagnosis, truth-maintenance .

ABSTRACT

This paper presents a constraint-network formulation for maintaing consistency of beliefs in dynamically changing knowledge bases. It exploits techniques developed for Constraint-Satisfaction problems and provides an efficient distributed scheme for updating beliefs in singly connected constraint-networks.

We present a belief-revision process consisting of two phases. In the first phase, called **support-propagation**, each variable updates the number of extensions consistent with each of its values. The second, called **diagnosis**, is invoked by a variable that detects a contradiction, and identifies a minimal set of assumptions that accounts for the contradiction. The support-propagation phase is accomplished in a single pass through the network while the diagnosis process takes at most 4 passes. Overall, the impact of any new input to the system can be propagated in at most 5 passes through the network. Extensions of this scheme to multiply-connected networks using the cycle-cutset and clustering approaches, are also discussed.

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

1. Introduction

Reasoning in dynamic environment is a central issue in Artificial Intelligence. If one assumes a partially described state of the world (representing the current focus of reasoning), and allows the "unexplicated" portion to impinge upon it by either adding or deleting facts, the difficulty is to keep the explicated knowledge "consistent" so that queries of interest (e,g., is P true?) can be reliably answered at all times. Various non-monotonic logics as well as truth-maintenance systems have been devised to handle such tasks [Doyle 1979, De-Kleer 1983]. This paper presents a **constraint-network** formulation of this problem and, using theoretical results developed for managing static networks, offers both an effective scheme of belief revision and a unifying abstraction within which other formulations can be tested, understood and compared.

We shall assume, as is in propositional and predicate calculus, that knowledge is represented by facts and relations among facts, i.e. by constraints among various entities. The **constraint-Satisfaction Model** (to be expounded later) is a simple language developed for representing systems of constraints that has the expressive power of propositional calculus and is extensively used for expressing **static** problems. A substantial body of knowledge for solving problems stated in this language was developed [Mackworth 1977, Montanari 1974, Freuder 1982, Dechter 1985] and some of its techniques e.g. backtrackings, dependency-directed backtracking etc, were also used in truth-maintenance systems [Doyle 1979, De-Kleer 1983]. The purpose of this paper is to establish a firmer link between non-monotonic reasoning systems and Constraint-networks, by showing how the latter handles dynamically changing environments.

2. The model

A **static** constraint satisfaction problem (CSP), also referred to as a **a constraint-network** (CN), involves a set of n variables $X_1,...,X_n$, each represented by its domain values, $R_1,\ldots,R_n$, 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 solution (also called an extension) is an assignment of values to all the variables which satisfy all the constraints, and the task is to find one or all solutions. A constraint is usually represented by the set of all tuples permitted by it. A **Binary CSP** is one in which all the constraints are binary, i.e., they involve only pairs of variables. A binary CSP can be associated with a **constraint-graph** in which nodes represent variables and arcs connect pairs of variables which are constrained explicitly. Consider, for instance, the CSP presented in figure 1a (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 constrained variables (the constraint between connected variables is a strict lexicographic order along the arrows.)

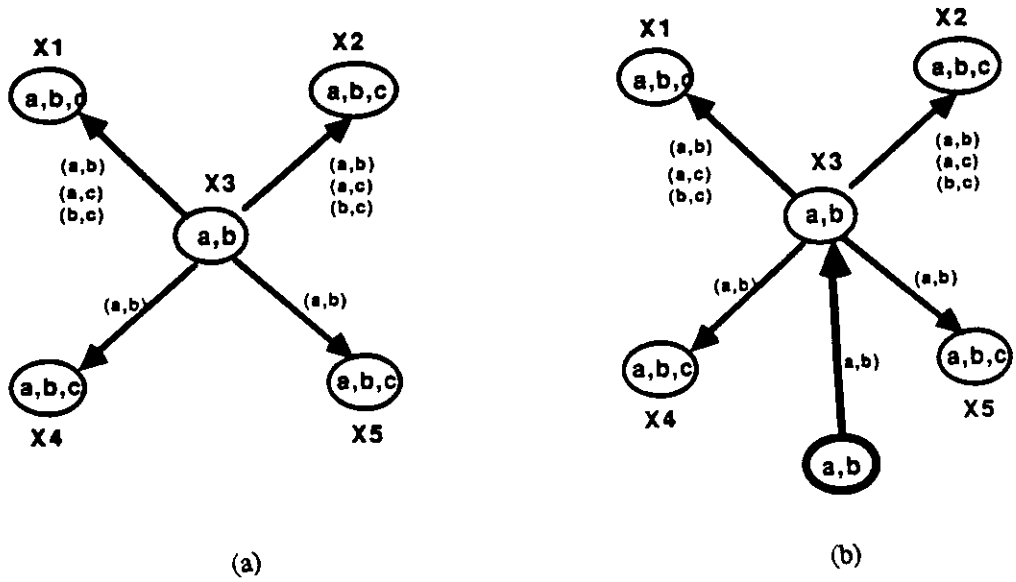

Figure 1: An example CSP

Dynamic Constraint-Networks (DCN) is a sequence of static CN's each resulting from a change imposed by the "outside world" on the preceding one. Changes can be of two types: **restrictions or relaxations.** Restrictions occur when a new constraint is imposed on a subset of existing variables (e.g. forcing a certain value for a variable), or when a new variable is added to

the system via some links. The added links may represent unary constraints, (i.e. restricting the domain values), or of a higher dimension (restricting a subset of the newly introduced variables and some variables already in the network). Figure 1b shows a change occurring in the model of figure 1a, by adding variable X_6 and its associated (lexicographic) constraint. Restrictions can only expand the model, i.e. they **add variables** and **add constraints** so that the associated constraint graph (representing the knowledge) monotonically grows.

We will impose this monotonicity property on **relaxing changes** as well, although such changes more naturally correspond to deletion of facts. If, for instance, a certain fact or constraint, is no longer known to be true, (i.e. it is in a "don't know" state), it seems appropriate to remove it from the network altogether. We insist, however, that the network will only grow (with the exception that unary constraints can be deleted) with the effect that, once a variable or a constraint enters the network, it will always stay there. This restriction requires to model potentially relaxable constraints in a special way. Consider, for example, two variables X_1 and X_2 with a binary constraint C_{12}, (Figure 2a). In order to express the change "C_{12} is not known to be true any more" an additional variable, X_3, is introduced, together with a a ternary constraint between X_1, X_2 and X_3 representing the relations: " $X_3 = 1$ if C_{12} holds and $X_3 = 0$ otherwise" (in figure 2b the edges are connected to indicate a ternary constraint). Initially, there is a unary constraint associated with variable X_3 restricting its value to "1", thus expressing the fact that constraint C_{12} holds. A change requiring the removal of C_{12} can be expressed by eliminating this unary constraint from X_3. In general, when an i-ary constraint is not expected to be always true it should be modeled by an i+1-ary constraint as indicated by the example.

The traditional tasks posed to the network can be to find a consistent extension or to find all consistent extensions. In Truth-maintenance systems the interest is in finding **theorems**, i.e. in detecting those variables having only one consistent value in all extensions. Such variable-value pairs, will be called **implied**. The task on which this paper focus, however, is to determine

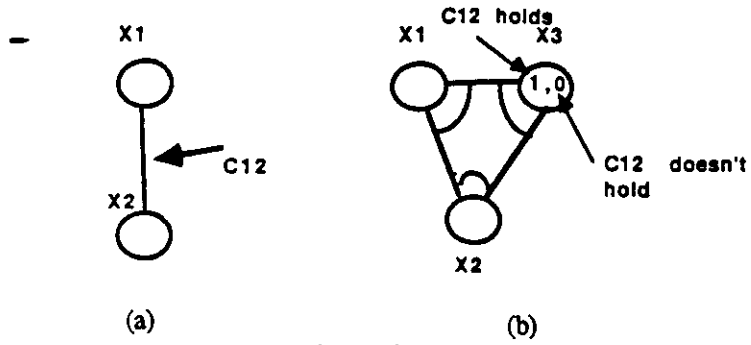

Figure 2

what we call, the **support vector**, of each variable, namely, to label each value with the frequency of its appearance in the set of all extensions. The reason for selecting this task is that all the other tasks can be easily derived from it. For instance, a variable's value is implied if its associated support is positive and is zero for the rest of the values. Similarly, having a variable with a **zero-support-vector** indicates a contradiction, (i.e. no consistent extension exists). Additionally, the pattern of support vectors can facilitate the efficient extraction of the set of all extensions (by making consistent assignment of values with non-zero supports).

The main contribution of this paper lies in providing an efficient scheme of propagating the revisions necessary for keeping all support vectors consistent with external changes. The scheme is presented for singly connected networks, and can be extended to the multiply-connected networks, where its complexity becomes a function of the sparseness of the network.

The following section describes how to derive and maintain the support information efficiently, in a binary tree-structured DCN (Dynamic-Constraint-Network). Section 4 extends the model to handle assumptions and contradictions, section 5 extends the scheme to include k-ary constraints, and section 6 discusses extensions to multiply-connected networks and contains concluding remarks.

3. Support propagation in trees

It is known that when constraint networks have tree-topology, they can be solved easily [Dechter 1985, Freuder 1982]. Moreover, **the number of solutions** on such tree-networks can be computed very efficiently. It turns out, that many of the sequential algorithms on trees can be adapted naturally to a distributed implementation and can be used for propagating beliefs and maintaining consistency.

Let each variable be associated with a **support vector**. The following paragraphs present a scheme for updating the support vectors following a change to the network. Consider a fragment of a tree-network as depicted in figure 3.

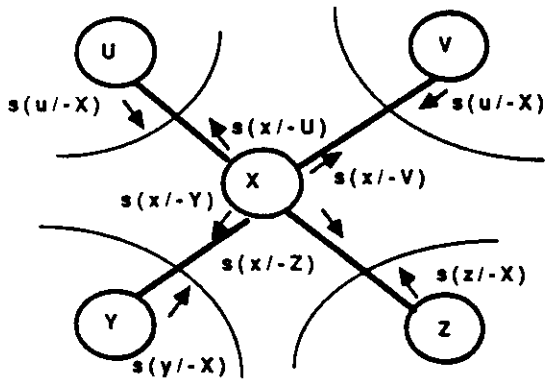

Figure 3

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 contributed by subtree $T_{XY}(Y)$, i.e. the number of extensions restricted to this subtree that are consistent with $X = x$, and by s_Y(y/-X) the support for $Y = y$ in $T_{XY}(Y)$. (These notations will be shorten to $s(x)$,$s(x/Y)$ and $s(y/-X)$ respectively whenever the

identity of the contributing subtree is clear). The support for any value x of X is given by:

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

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:

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

Namely, since x can be associated with several **matching** values of Y, its support is the sum of the supports of the associated y-values. Equalities (1) and (2) yield:

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

Equation (3) lends itself to a nice propagation scheme, i.e., if variable X gets from each neighboring node, Y, a vector of restricted supports, (also called **the support vector from Y to X**)

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

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

$$s(x/-Y) = \prod_{Z \in neighbours,\ Z \neq Y} \sum_{(x,z) \in C(X,Z)} s(z/-X) \tag{4}$$

The message generated by a leaf-variable is a vector of "zero"s and "one"s expressing the unary constraint on its domain, i.e. "0" is associated with each illegal value and "1" with every legal value.

Assume, that the network is initially in a stable state and the task is to maintain 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, 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" , it may cause a contradictory state, in which case, all the nodes in the network will converge into a "0" support vector.

To illustrate the mechanics of the propagation scheme described above, consider the problem of figure 1a. In figure 4a 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 4b.

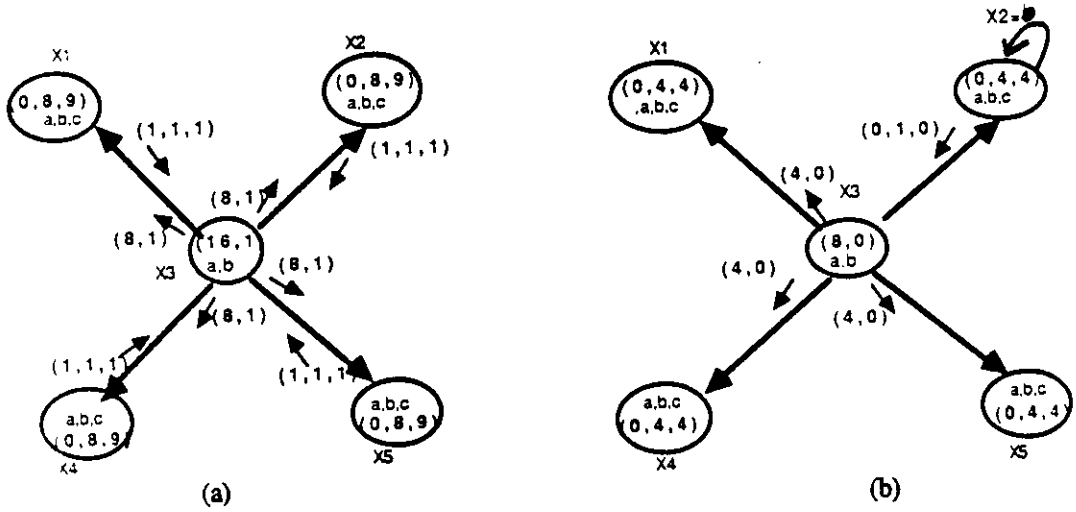

Figure 4

If one is not interested in calculating numerical supports but merely in indicating whether a given value participates in some extension, (i.e. having "1" if it does and "0" otherwise), then **flat** support-vectors can be propagated in exactly the same way; the summation operation in (3) should be replaced by the logic operator "OR", while the multiplication can be replaced by

"AND".

4. Handling assumptions and contradictions

While most of the changes imposed on the system comes from the outside world, it is common in non-monotonic reasoning to allow the system to make **assumptions** i.e., temporary restrictions imposed by the system for inferential purposes. When contradictions occur, they no longer indicate a state of dead-end but only state that under the current set of assumptions a consistent extension no longer exists. The system is now at liberty (or obligation) to reverse and change the assumptions to achieve a non-contradictory state. This process is modeled by allowing the system internally to impose **unary constraint** on some variables, marked as **assumptions**, and later to delete or change them in a contradiction resolution process.

The main task in a conflict resolution process, is to identify a smallest set of assumptions that may account for the conflict. We will call this process **the diagnosis task**, (Uncoincidently, this task is closely related to the way we formulated the problem of circuit diagnosis, where devices were modeled as assumption variables and the task was to identify a minimal number of faulty devices that will explain the system fault. For details see [Dechter 1986., Geffner 1986].). The diagnosis process, too, can be performed distributedly but, unlike the support propagation scheme, it has to be synchronized. Assume, at first, that when variable X detects a contradiction it propagates this information to the whole network, and during this propagation process it creates a directed tree rooted at itself. (Later we show that only a relevant subtree actually needs be activated). Given this tree, the diagnosis process proceeds as follows:

With each value v of V we associate a weight $w(v)$, indicating the **minimum number of assumption-changes** in the directed subtree rooted at V, which needed to make v consistent. These weights obey the following recursion:

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

when $\{Y_i\}$ 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 5).

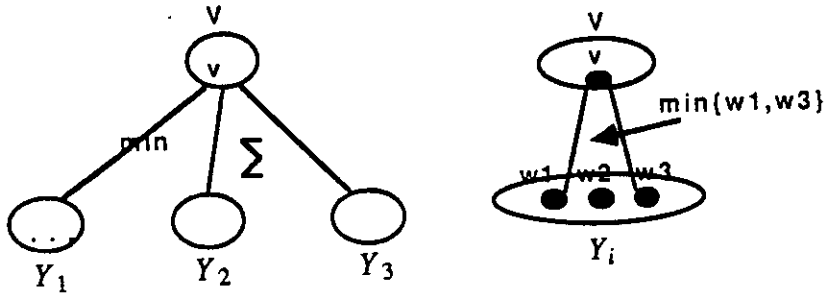

Figure 5

The weights associated with assumption-variables will be "0" if no change is assumed and "1" otherwise. The computation of the weights is performed distributedly and synchronously from the leaves 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 minimum 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 the diagnosis process, 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 fol-

lows: 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 diagnosis process, consider the network given in figure 6a, which is an extension to the network in figure 1a (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 neighbours are explicitly indicated. (The overall support vectors are not given explicitly.) It can be easily shown that the value a for X_3 is implied 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 6b). 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 diagnosis, 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 6b. The diagnosis process 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 diagnosis computation itself is not demonstrated).

Once the diagnosis process had been terminated, all assumptions can be changed accordingly, and the system can get into a new stable state by handling those changes as if they were an outside world changes, i.e by propagating the newly updated support vectors. If this propagation is not synchronized, the amount of message passing on the network may be proportional to the

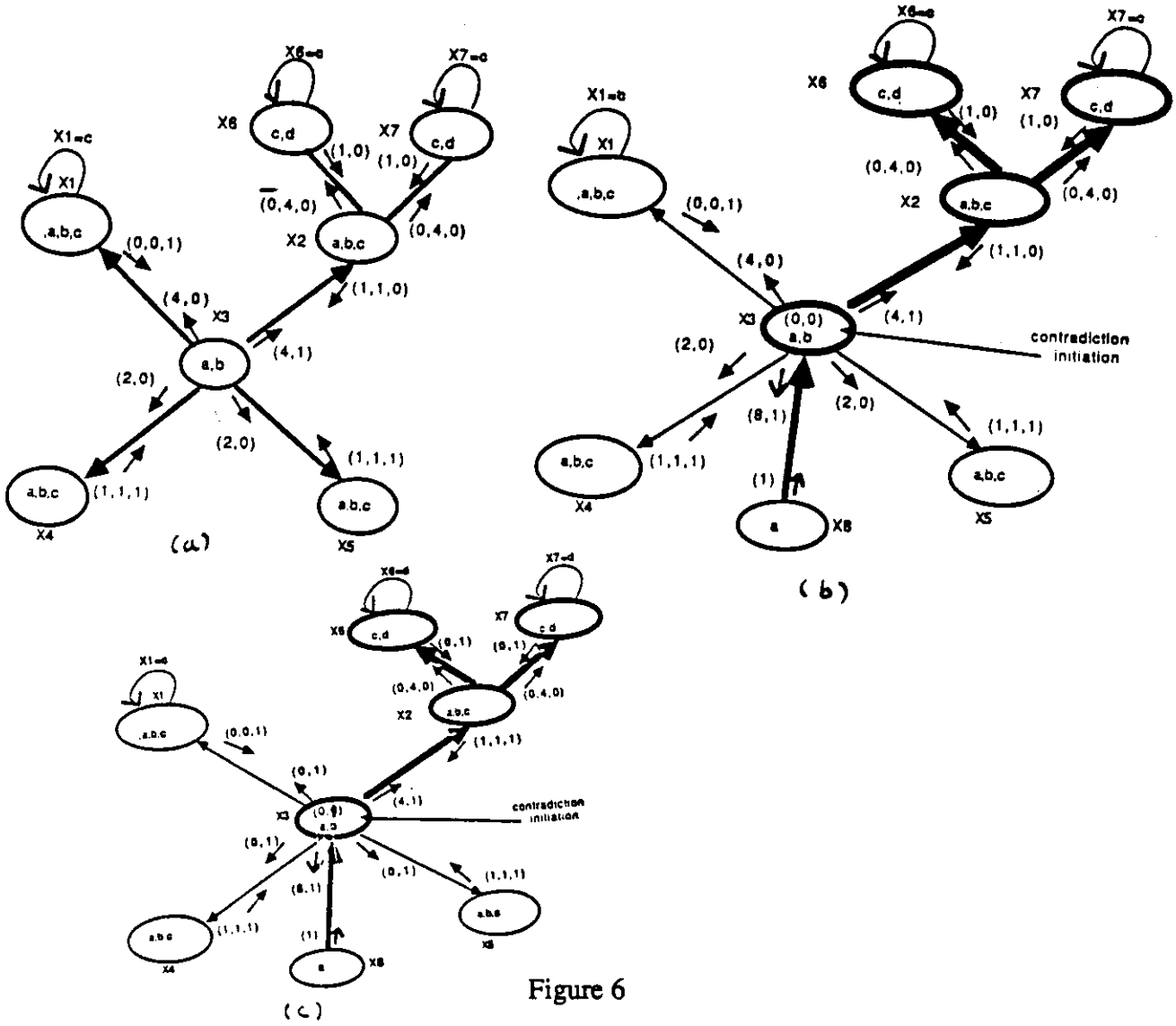

Figure 6

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 6c, give the new updated messages after the system had been stabilized.

5. Handling arbitrary constraints

Non-binary constraints can have various graphical representations each capturing some aspects of the constraint. In Bayes networks, for instance [Pearl 1986], they are represented by directed graph, namely, a node and all its parent nodes is connected by a hyper constraint, quantified by its conditional probability given all its parents. For our needs we choose to stay in an undirected graph representation, and represent a k-ary constraint by a complete graph among

the constrained variables, adding connecting marks to distinguish it from a set of binary constraints. (see figure 7).–The parallel of trees in such **generalized constraint graphs** are **singly-connected** networks, in which every two constraints have at most one common variable and a sequence of "connected" constraint does not contain a cycle. Figure 7a and 7b presents both singly-connected and multiply-connected networks, respectively.

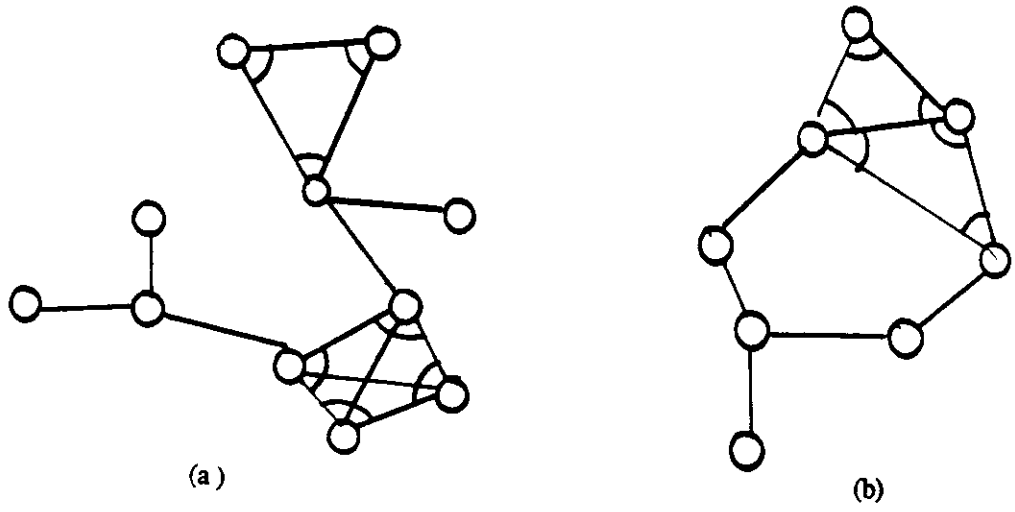

Figure 7

An extension of the support propagation scheme to singly-connected networks follows. Consider a variable X and its neighboring constraints, as depicted in figure 8a. Variable X participates in constraints C_1 (a 4-ary constraint) , C_2 , and C_3 (binary constraints). An i-ary constraint separates the network into i singly connected sub-networks each containing one of the constraint variables. Denote by $T_C(X)$, the tree containing X that result from separation by constraint C (see figure 8a), by $s(x/-C)$ the support for $X = x$ which is contributed by $T_C(X)$ (see figure 8a), and by $C_i(X)$ the i^{th} constraint associated with X. The overall support for x of X satisfies the following recursion:

$$s(x) = \prod_{C_i(X)} \sum_{(x, v_1, v_2, \ldots, v_t) \in C_i(X)} \prod_{j=1}^{t} s(v_j/-C_i(X)) \qquad (6)$$

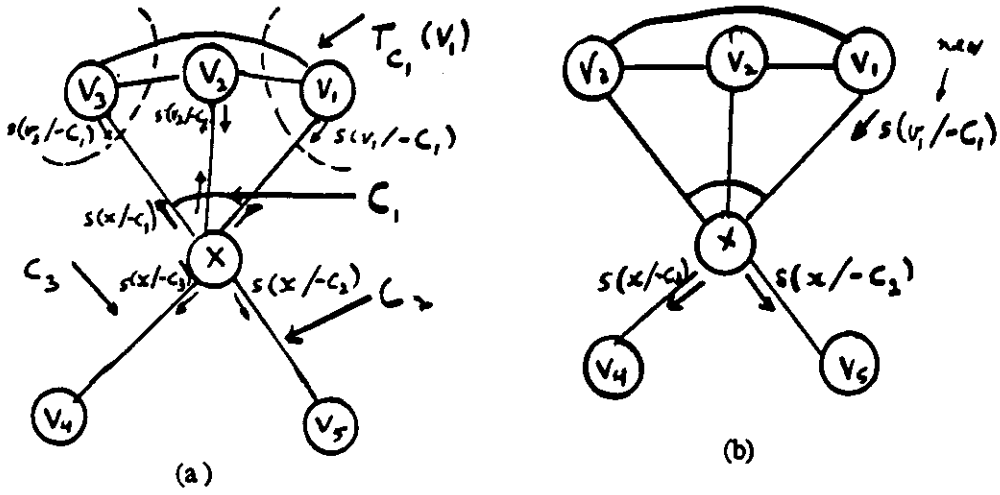

Figure 8

Equation (6) suggests the following modification to the support propagation scheme: A variable, X, sends to **each of its neighboring variables** via constraint C, its support vector $(s(x_1/-C),...,s(x_t/-C))$, i.e. the support which it receives "outside" of constraint C (i.e. contributed by $T_C(X)$). At the same time it will receive from each of this neighbours their supports outside this constraint C, enabling it to compute both its overall support (according to (6)), and its respective contributing supports to be sent to the rest of its neighbors. When a variable gets a message from a neighbor in constraint C, it updates the relevant messages to all its neighbors **excluding those that share constraint C with it** (see figure 8b). For this scheme to work, each variable has to know all its constraints explicitly, namely, an i-ary constraint will have to be duplicated at each one of its i variables.

The diagnosis process can be similarly modified for singly connected general constraint graphs. For details see [Dechter 1986.] Both the support propagation phase and the diagnosis process on singly-connected networks take the same amount of message passing as their simplified (binary-constraints) versions.

6. Summary and conclusions

We presented an efficient scheme for belief revision in dynamic networks of constraints which are singly connected. The scheme contains two main phases: support updating and conflict resolution. The first handle non-contradictory inputs and requires one pass through the network. The latter finds a minimum set of assumption-changes that resolves the conflict. The diagnosis process 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).

When the network contains loops the scheme can be extended using two approaches: Cycle-cutset and clustering. The **cycle-cutset** method is based on the idea that when a variable is instantiated, it actually cuts all paths through it. Therefore, if we instantiate a set of variables that constitutes a cycle-cutset, the resulting network is singly connected and our propagation scheme holds. This method has been used and tested in static CSPs. (see [Dechter 1987]. **Clustering** involves grouping variables into clusters so that the resulting structure is singly connected. This require computing and storing the constraint associated with each cluster. The quality of this approach is dependent on the sizes of the resulting clusters while the quality of the cycle-cutset method is dependent on the size of the cutset. A combination of the two approaches is also feasible.

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 and will appear, under various disguises, under any formalism. The importance of network representation, though, 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

[Dechter 1985] 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: 1985, pp. 1066-1072.

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

[Dechter 1987] Dechter, R. and J. Pearl, "The cycle-cutset method for improving search performance in AI applications," in *Proceeding of the 3rd IEEE on AI Applications*, Orlando, Florida: 1987.

[De-Kleer 1983] De-Kleer, Johan, "Choices without backtracking," in *Proceedings AAAI*, Washington D.C.: 1983, pp. 79-85.

[Doyle 1979] Doyle, John, "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 1986] Geffner, H. and J. Pearl, "Am improved constraint propagation algorithm for diagnosis," UCLA, CS department, Cognitive systems lab., Los Angeles, California, Tech. Rep. R-73, 1986.

[Mackworth 1977] Mackworth, A.K., "Consistency in networks of relations," *Artifficial 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.