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

A DISTRIBUTED ALGORITHM FOR ATMS

Rina Dechter

July 1988
CSD-880057

TECHNICAL REPORT
CSD 8700XX
R-109 (S)
March 1988

A DISTRIBUTED ALGORITHM FOR ATMS *

Rina Dechter

Cognitive System Laboratory, Computer Science Department
University of California, Los Angeles, CA, 90024
Network address: dechter@cs.ucla.edu

Length: 4400 words

topics: Knowledge Representation

ABSTRACT

ATMS has been proposed by De-kleer as an instrument performing non-monotonic reasoning. Reiter and De-Kleer have shown that, formally, the ATMS computes the set of all minimal supports to a query w.r.t to a clausal data-base. We investigate the complexity of the ATMS task in a simplified problem structure and identify the problem's parameters that have a decisive effect on the complexity. It is shown that even in the simplest problem structure of a constraint tree, the complexity of an ATMS is in general exponential, while for bi-valued variables a linear complexity is achieved. We propose a distributed algorithm for ATMS that exploits the structure of a tree and achieves optimal complexity for horn-clause type constraints.

* This work was supported in part by the National Science Foundation, Grant # IRI-8501234

1. Introduction

Truth-maintenance systems are computational schemes that reason non-monotonically, namely, they allow inference systems to incorporate assumptions and defaults, thus, drawing and retracting conclusions in the context of incomplete or uncertain information. In the last decade, two main TMS's approaches emerged. The JTMS approach [Doyle 1979, McAllester 1980] and the ATMS approach [De-Kleer 1986]. In the former, a set of assumptions is provisionally adopted until an inconsistency is derived, at which time, the system produces an alternative consistent set of assumptions. In assumption-based truth-maintenance, the system maintains, with each proposition, not just one, but a whole list of consistent assumption sets, each sufficient to prove the proposition.

An important first step towards formalizing the TMS's tasks was made by Reiter et. al. [Reiter 1987], where the ATMS task is defined within the context of Clause-Management-System. Our aim in this paper is to follow Reiter's footsteps, but instead of using propositional logic as the basic language we use constraint-networks' formulation, a language that has the same expressive power, but more naturally expresses the idea that certain sets of propositionss are mutually exclusive and exhaustive, thus mapped into multi-valued **variables**.

Another reason for using constraint-networks formulation is that, in this framework, tasks and algorithms are clearly distinguished, and a body of knowledge relating the tasks' tractability to the specific structure of the network has been established. By explicitly formulating TMS's goals within the CSP terminology one hopes to exploit constraint networks to come up with either efficient algorithms, or, alternatively, pin-point computational difficulties.

2. The clause management system; a summary

The Clause Management System (CMS) [Reiter 1987] assumes a problem solving architecture which consists of a reasoner, representing the outside world, and the CMS. The CMS forms a knowledge base from a set of propositional clauses received from the reasoner, then, upon receiving a query q the CMS responds with the set of all **minimal supports** of q w.r.t. the current state of the knowledge base. A clause is a disjunction of literals $\{L_1,...,L_n\}$ with no literal repeated. Let Σ be a set of clauses, and C be a clause. A clause S is a **support** to C w.r.t. Σ iff $\Sigma \not\models S$ and $\Sigma \models S \cup C$ (the symbol $\models$ stands for the usual entailment relation). S is a **minimal support** to C w.r.t. Σ iff no proper subset of S is a support to C w.r.t. Σ. Intuitively, S supports C if the negation of S is sufficient to prove C given Σ.

De-Kleer's ATMS [De-Kleer 1986] is a CMS having knowledge-base restricted to horn clauses (a clause with at most one unnegated propositional symbol), called **justifications**, and a distinguished subset of propositional symbols, $\{A_1, \ldots, A_n\}$, called **assumptions**. De-Kleer's task restrict the support to the set of assumptions, i.e., given the current set of justifications, J, and a propositional symbol, α, the task is to compute all sets of assumptions $\{A_1, \ldots, A_r\}$ such that $\{\neg A_1 \models, \ldots, \models \neg A_r\}$ constitute a minimal support to α w.r.t. J.

3. Constraint-formulation for CMS

A **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 tasks most common with these problems 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 following set of propositions: $\{Z = X \mid Y, T \rightarrow Z, L \rightarrow X, R \rightarrow Y\}$ which can be modeled as a CSP with the four bi-valued variables T,Z,R,L and the compound variable XY having the values, 00, 01,10,11 (each pair is one value). The constraints themselves are the logical constraints of implications and equality. The binary constraint network is depicted in figure 1.

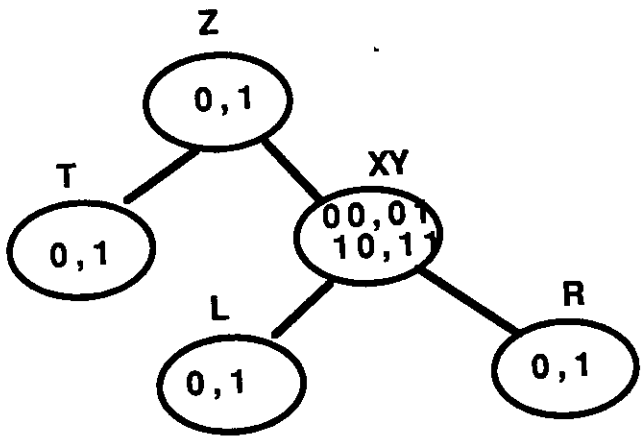


Figure 1: An example CSP

Dynamic Constraint-Networks (DCN) is a sequence of CN's each resulting from a change imposed by the "Reasoner" on the preceding one. A CMS is a dynamic constraint network. Each propositional symbol is a bi-valued variable. A clause is a constraint on the corresponding subset of variables and a set of clauses is a constraint-network. A constraint C is **entailed** (or believed) by a network of constraints R if it holds for **every solution of** R. A **support** for a constraint C w.r.t. network R is a consistent instantiation of subset of the variables, $(X_1 = x_1, \ldots, X_k = x_k)$ such that C holds in the restriction of R to this instantiation. The support is

minimal if no subset of the instantiation is a support to C w.r.t. R.

The tasks of ATMS rephrased into DCN model is to find all minimal instantiations which support a query C, given a network R. When the query is a unary constraint $X = x$?, and X has only two values, we have the restriction of De-Kleers' ATMS to unit clause queries. In this case a subset of the variables are distinguished as **assumption variables** and the support is restricted to instantiations from this set only.

4. A distributed algorithm for finding all minimal supports

The tasks normally considered by JTMS is to maintain a current set of assumptions consistent with the rest of the knowledge-base, to be able to determine if a clause is entailed by the current environment, and, in case of inconsistency to find a new set of assumption that preclude the inconsistency. In a companion paper [Dechter 1988a] we have formulated and performed these tasks within the dynamic constraint network model. The task is accomplished by maintaining **support-numbers** with each value of a variable representing its degree of belief. A support-number of a value v of a variable V, denotes the **frequency** of its appearance in the set of all solutions, i.e., indicating the number of solutions in which V is assigned the value v. Thus a variable's value is **believed** w.r.t. the current assumptions, if it is the only one to obtain a positive support-number w.r.t. other values of the same variable. A variable whose all values have zero supports indicates that no consistent solution exists. We showed that JTMS is linear for binary singly connected networks.

Apparently, the ATMS' task contains the JTMS tasks and more. Since ATMS maintains the set of all minimal supports, when a specific environment is selected (i.e., specific instantiation of the assumption variables), all values that have a minimal support included in this environment are believed, and this can be determined by scanning their labels.

We assume a restricted problem structure of singly connected networks with binary constraints. Although it is a very restricted model it serves two purposes. Being a simplified model, it helps identify the source of the computational difficulties associated with ATMS. Second, efficient algorithms on this model may be adapted to general networks using a tree-clustering transformation [Dechter 1987].

We start by demonstrating the algorithm on the example of figure 1. Suppose that all minimal supports for $Z=1$ are desired. The algorithm generates a directed tree rooted at Z (figure 2a). Then, for each value, computes all its minimal support sets **restricted to its child nodes.** For instance, the set of minimal support sets computed for $XY=11$ is $(L=1,R=1)$, while for $XY=01$ the set is empty since no instantiation of the child variable R and L entails $XY=01$. This subset of minimal supports, coming from the children of a variable, are called **minimal support labels**. The label-set of a parent value v is denoted by $L(v)$ (see figure 2b) while the set of all its minimal supports w.r.t. its rooted subtree is denoted by $\Phi(v)$.

Once all minimal support labels are computed, new supports can be generated by replacing a value in a label by one of its own minimal supports. For instance, since $(XY=11)$ is a minimal support label for $Z=1$ and since $XY=11$ is minimally supported by $(L=1,R=1)$, this later set is a new support for $Z=1$. Notice, however, that the minimality property **is not** maintained by this process; the set $(L=1,R=1)$ is not a minimal support for $Z=1$, since either $L=1$ or $R=1$ independently support $Z=1$. Therefore, each subset generated by the substitution process, should be tested for minimality, a process which may produce smaller support subsets. By adding the value itself to its own supports and by performing the substitution process and the minimality test from leaves to root we get a set of minimal supports for $Z=1$ (see figure 2c).

Not all minimal supports can be generated by the above substitution process. For instance, consider, in figure 1, the network restricted to variables (Z,XY,L). Both $(XY=10)$ and $(XY=11)$ are minimal supports to $Z=1$. Each one of these value, individually, is not supported

by L (i.e., L=1 is consistent with both of them), thus L=1 cannot substitute XY=01 nor XY=10, although , L=1 is a minimal support to Z=1.

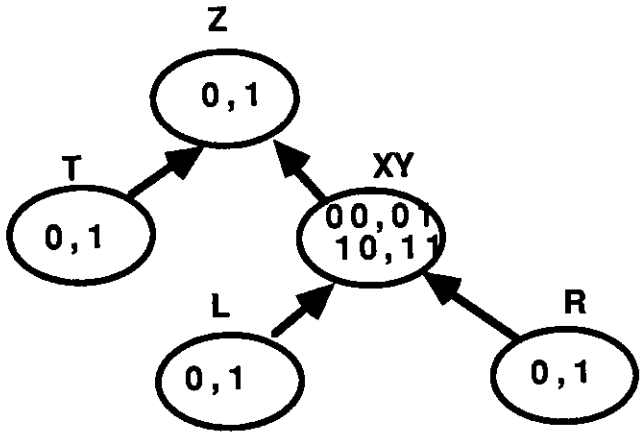
The algorithm we will present does not generate minimal supports of the above "type" and hence is incomplete. It generates minimal supports which are **reachable** by a recursive process, to be described, and is said to be **R-complete**. A complete extension will be given in the full paper [Dechter 1988b]. The reason for using this restriction is that it allows a clearer presentation without compromizing our results.

The algorithm works on a directed tree rooted at the queried variable and it assumes that the network is already maintaining the support numbers with each value. Let $K_V{}^C(c)$ denotes all consistent values of V which are compatible with $C = c$ by the direct constraint between V and C (figure 3a), and let D_C denotes the consistent domain of C (namely, all values of C having a positive support number). Given a variable V with its children $C_1, \ldots, C_l$ (figure 3b) and a value v of V we define a **support label** as an instantiation of subset of the child variables $(C_{i1} = c_{i1}, \ldots, C_{it} = c_{it})$ such that from all (the consistent) V's values **only** v is compatible with each of these child values, namely:

$$\bigcap_{c_{ij} \in D_{ij}} K_V{}^{C_{ij}}(c_{ij}) = \{v\} \qquad (1)$$

A support label is minimal if no subset of it satisfies condition (1).

The algorithm associates each value with **all its minimal support labels.** The computation of this labels, can be performed locally, between every node and its children. Various heuristic algorithms can be used for this task and we will assume a standard search algorithm, that checks condition (1) for all instantiation of one variable first, then go to two variables , using values not selected previously, and continues to larger support labels. The minimal support labels of leaf values are the empty sets.



(a)

For $\alpha \in \{0,1\}$

$$L(L=\alpha)=\{\}$$
$$L(R=\alpha)=\{\}$$
$$L(XY=01)=\{\}$$
$$L(XY=00)=\{\}$$
$$L(XY=10)=\{\}$$
$$L(XY=11)=\{(L=1,R=1)\}$$
$$L(T=\alpha)=\{\}$$
$$L(Z=1)=\{(T=1)(XY=01)(XY=10)(XY=11)\}$$

(b)

$$\Phi(L=0)=\{(L=0)\}$$
$$\Phi(L=1)=\{(L=1)\}$$
$$\Phi(R=0)=\{(R=0)\}$$
$$\Phi(R=1)=\{(R=1)\}$$
$$\Phi(T=0)=\{(T=0)\}$$
$$\Phi(T=1)=\{(T=1)\}$$
$$\Phi(XY=01)=\{(XY=01)\}$$
$$\Phi(XY=10)=\{(XY=10)\}$$
$$\Phi(XY=00)=\{(XY=00)\}$$
$$\Phi(XY=11)=\{(XY=11)(L=1,R=1)\}$$
$$\Phi(Z=1)=\{(Z=1)(T=1)(XY=01)(XY=10)(XY=00)(XY=11)(L=1)(R=1)\}$$

(c)

Figure 2

Once the support labels are computed, $\{v\}$ itself is added to the set of supports (unless v is entailed by the subtree) and then, the rest of the supports are generated using the recursion:

$$S(v) = \{(S(c_{i1}) \cup \cdots \cup S(c_{it})) \mid (c_{i1}, \ldots, c_{it}) \in L(v),\ S(c_{ij}) \in \Phi(c_{ij})\} \tag{2}$$

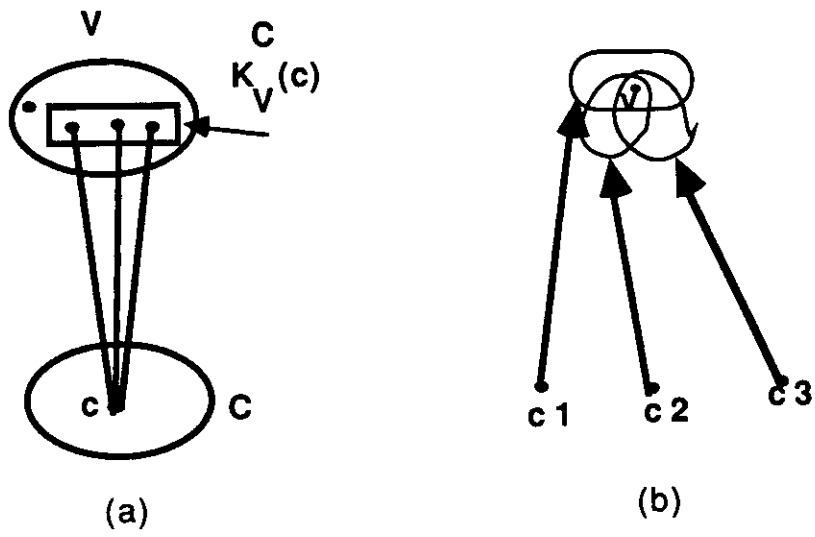


Figure 3:

when $S(v)$ denotes an arbitrary support to v and $\Phi(v)$ is the set of all its minimal supports w.r.t. its subtree, which are reachable by recursion (2). Through the rest of the paper, any reference to **all minimal supports**, or their cardinality is restricted to **reachable** supports unless specified otherwise. As shown by the example, the recursion process may generate non-minimal supports, and thus is accompanied by a minimality procedure.

The set of minimal supports is computed recursively from leaves to root. A variable, whose child variable had already computed their minimal supports, performs two tasks w.r.t. each of its values. First, an **aggregation task** in which it calculates the set of all supports using recursion (2), followed by a **minimality operation** which generates the minimal supports of each previously computed support set. Algorithm **Generate-min-support** describes this procedure for a parent variable, V, its value, v, and all its child-nodes $C_1, \ldots, C_l$.

Generate-min-support(v, $C_1, \ldots, C_l$, L(v))
1. begin
2. $\Phi(v) = \{\}$
3. if v is entailed then $\Phi(v) = \{\{\}\}$, stop.
4. else, Φ (v) <-- {{v}} /* namely,v support itself.
5. for every $(c_{i1}, \ldots, c_{it}) \in L(v)$ do
6. for every $S(c_{i1}) \in \Phi(c_{i1})$, $S(c_{i2}) \in \Phi(c_{i2})$ ... $S(c_{it}) \in \Phi(c_{it})$ do
7. begin
8. $S(v) = \{(S(c_{i1}) \cup \cdots \cup S(c_{it}))$
9. $S'(v))$<-- compute-min($S(v)$)
10. $\Phi(v)$ <-- $\Phi(v) \cup S'(v)$
11. end.
12. end.
13. end.
14. end.

Procedure compute-min($S(v)$) computes all minimal supports to v which are subsets of $S(v)$ by exhaustively testing subsets for "supportness" in decreasing order of their sizes.

5. Complexity analysis

The time complexity of the algorithm is the combined complexity of label-generation and generate-min-support algorithm.

The computation of the minimal support labels, can be performed locally, between every node and its children. For the analysis we assume w.l.o.g. that the tree is uniform with degree d, that each variable has k values, that all minimal labels are of size s and a uniform number, N_l, of minimal support labels for each value (except leaves). Since there are at most $\binom{d}{s}$ variable-subsets of size s and since every combination of values from these subsets may generate a minimal support, the number of support labels, for all values of a variable is bounded by ($\binom{d}{s} \cdot k^s$). Since the support labels of values of the same variable are mutually exclusive, this number is divided among the k values to yield:

$$N_l = \left(\binom{d}{s} \cdot k^{s-1} \right) \tag{3}$$

It is possible to construct a worse-case example in which any subset of $\frac{d}{2}$ variables provides a minimal support for v with each possible instantiation of its variables. On the other hand, since each variable has only k values, the size of a minimal support label (or set) cannot exceed $k-1$. The reason being that each value (i.e., an instantiation) participating in a minimal support, should decrease the intersection set (see (1)) by at least one value. We can therefore assume a worse-case situation in which the number of support labels are of size $s = min(k-1, \frac{d}{2})$ which leads to the following:

Theorem 1:

i. The number of labels, N_l, has a worse case bound of:

$$N_l = \begin{cases} O\left(\frac{d^d}{(d-k+1)^{d-k+1} k)}\right) & \text{, if } k-1 < \frac{d}{2} \\ O\left(k^{\frac{d}{2}-1} \cdot 2^d\right) & \text{, if } k-1 \geq \frac{d}{2} \end{cases} \tag{4}$$

and these bounds are tight.

ii. In the worse-case, the number of labels, N_l, has the following lower bound: (5)

$$N_l \geq (2d)^{\frac{d}{2}}$$

□

Part (i) is derivable by substituting $s = min(\frac{d}{2}, k-1)$ in (3) and performing some simplification while (ii) is derived by substituting $k-1 = \frac{d}{2}$ in (i). Notice that generally the existence of a tree structure **does not avoid exponential computation** of labels.

Given that the minimal support labels are already computed, the complexity of generating all minimal support-sets for a query is composed of the complexity of the "aggregation operation" performing recursion (2) and the operation of minimality. Let $(c_{i1},\ldots,c_{it})$ be a minimal support label of a value v of V, we can see that it can generate as many as $N_s(c_{i1})\cdot N_s(c_{i2})\cdot\cdots\cdot N_s(c_{it})$ support-sets for v via the aggregation operation where $N_s(v)$ is the number of minimal support-sets for v in the subtree rooted at V. Let $N_s(i)$ be the number of support-sets of a value whose rooted tree has a depth i. $N_s(i)$ satisfies the following recursion:

$$N_s(i) = 1 + N_l(N_s(i-1))^s \tag{6}$$

with

$$N_s(0) = 1$$

From this recursion we get that:

$$N_s(i) = \begin{cases} O\left(N_l^{\frac{s^i-1}{s-1}}\right), & s>1 \\ \sum_{j=0}^{i} N_l^j, & s=1 \end{cases} \tag{7}$$

Since the size of a label is bounded by $\min(k-1,d)$ and denoting the depth of the tree by dpt we can conclude:

Theorem 2:

The number of support-sets, N_s, of an arbitrary value is:

$$N_s = \begin{cases} O\left(N_l^{(k-1)^s}\right) & s>1, k-1<d \\ O\left(N_l^{d^s}\right) = O\left(N_l^n\right) & S>1, k-1\geq d \\ O\left(N_l^{dpt}\right) & s=1 \end{cases} \tag{8}$$

and all the above bounds are tight.

□

Since the time complexity of any algorithm that computes all minimal support is at least as the size of the output we get:

Corollary 1:

Equation (8) provides a lower bound for the complexity of ATMS on a singly connected binary network.

□

The minimality process may add an exponential factor to the the computation of each minimal support set. Since the size of a support set cannot exceed $k-1$, and the size of a minimal support label do not exceed $\min(k-1,d)$, the size of intermediate supports generated by the aggregation operation is bounded by $(k-1)\cdot\min(k-1,d)$ yielding the minimality operation to be $O(2^{(k-1)\cdot\min(k-1,d)})$. We conclude therefore:

Theorem 3:

The time complexity, T, of the algorithm is given by:

$$T=\begin{cases} O(N_l^{(k-1)^{dpt}}\cdot 2^{(k-1)^2}) & s>1, k-1<d \\ O(N_l^{n}\cdot 2^{(k-1)d}) & s>1, k-1\geq d \\ O(N_l^{dpt}) & s=1 \end{cases} \tag{9}$$

□

Corollary 2:

For bi-valued variables, $N_l = d$, and $T = N_s = O(d^{dpt}) = O(n)$

□

Restricting the minimal supports to assumption variables only reduces the complexity at most by replacing d or n by N_A, the size of the assumption set. It is possible to show that a synchronized computation of labels can bound label size by:

$$s \leq \min(\frac{d}{2}, k-1, N_A) \tag{10}$$

Therefore, if $N_A \leq d$ (otherwise the complexity is as if all variables are assumptions), and substituting (10) in (3) we get:

Theorem 4:

When $\frac{N_A}{2} \leq k-1$:

i.

$$N_l = O(2^{N_A} \cdot k^{\frac{N_A}{2}-1}) \tag{11}$$

ii.

$$N_s = O(N_l^{N_A}) \tag{12}$$

and these bounds are tight.

□

6. The maintenance task

The minimal support sets can be computed in query time or can be pre-compiled. It may be useful to use a compiled approach, if by utilizing some space, the maintenance task becomes tractable. In view of the above analysis we propose that only the minimal support-labels will be compiled and maintained. Although the minimal support labels are space exponential (see (4)) their dependence on the number of values and the problem's structure may reduce this complexity occasionally.

Since a query may involve any variable, any directed tree may be invoked and thus any partition of the set of neighbours of a variable into a parent node and child nodes is possible. We assume that the tree is maintained with some default directionality and that all computations are made w.r.t. it. This is the approach taken in all TMS's when causality and implication dictates

the direction by which information is kept. When a query involves the root of the tree the minimal support-sets can be generated based on the stored minimal support labels. When a query concerning an arbitrary variable arrives, its directed tree will be created by changing the direction only on the path from it to the root, and as a consequence, only variables on this path will have to recompute their labels (see figure 4a).

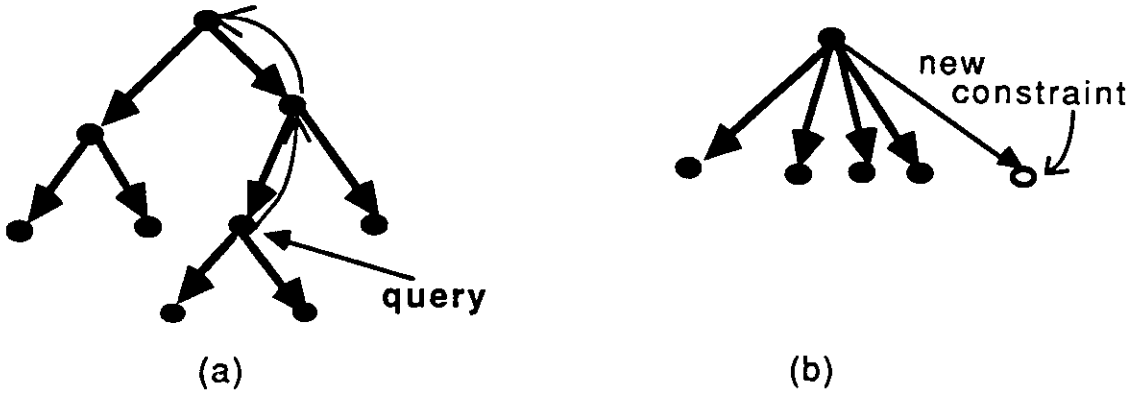


Figure 4: two different parents dictated by different queries

When a new constraint arrives it may invalidate certain values or may present new supports, thus, resulting in two maintenance tasks. The complexity of computing only the **additional** minimal support labels due to a new constraint between a new and an old variable, can be, in the worst case, as complex as recomputing all the minimal labels. Any instantiation of old variables that do not contain a support has a potential to join with a value of the new variable to create a new minimal support label (figure 4b).

Since each change to the network invokes the propagation algorithm which updates the support-**numbers** [Dechter 1988.] and marks some values as inconsistent, the second task involve discarding supports containing such inconsistent values. This operation can be performed by each variable independently, and is linear in the size of the minimal support labels.

7. The case of horn clauses

Restricting the knowledge to horn clauses, as in ATMS [De-Kleer 1986], and the queries to single variable or a conjunction of variables but no disjunctions, corresponds to constraint-networks with bi-valued variable. In this case the minimal support labels, and the minimal support sets consist of one value only, and thus, label computation becomes both time and space linear in the degree d of a variable (corollary 2), while the support-sets computation is linear in n. This shows the tremendous computational saving the horn clause restriction has. In this case the algorithm generates **all minimal supports** (i.e., it is complete) since they are all reachable by the recursive process.

When the horn-clauses are not binary, but still maintain a singly-connected structure, the number of minimal support sets, N_s, is $O(\#c^{(sc-1)^{dpt}})$, when $\#c$ is the number of constraints in which a variable participate, sc, is the number of variables in a constraints, and dpt is the depth of the network. for details see [Dechter 1988b]. Consider, for instance, the following set of horn clauses:

(D --> C), (C,B ---> A), (E,G,F ---> C), (A,C ---> E)

It can be represented by a constraint graph whose nodes are the constraints and arcs connect nodes that share a variable (figure 5a). This problem is shown to be singly connected by removing some of its "redundant" arcs, generating the "join-tree" in figure 5b. (For details see [Dechter 1987]).

Consider the value $A=1$. A participates in two constraints (ABC,AEC) and only within (ABC), $A=1$ gets support. Its minimal support labels consists, therefore, of one set: ($C=1$,$B=1$). In order to compute all the minimal support sets for $A=1$, each value in a minimal label can be replaced by its own support coming from **outside the subtree** which contains A. The generation of the minimal support sets via the support labels is the same as in the binary case. $B=1$ has no support except itself, while $C=1$ is supported by its minimal labels {($D=1$), ($E=1$,$G=1$,$F=1$)}

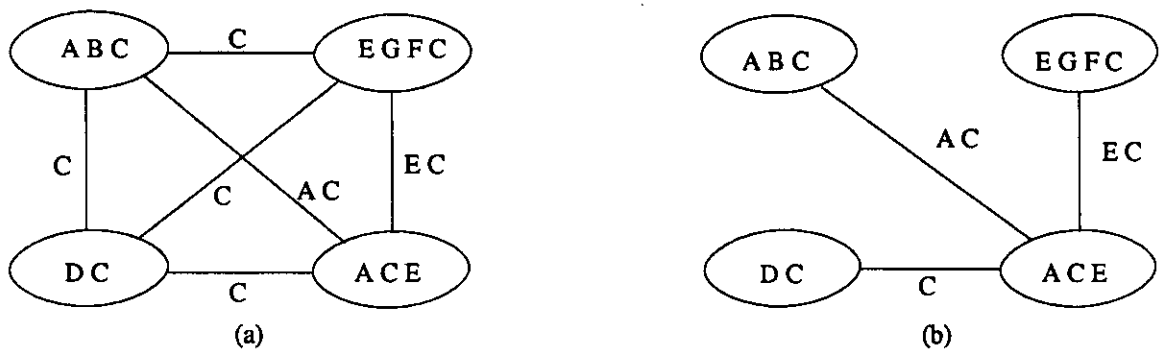


Figure 5

yielding four support sets to $A=1$ which are: $\{(A=1),(B=1,C=1),(B=1,D=1),(B=1,E=1,G=1,F=1)\}$.

8. Summary and conclusions

This paper presents a sound (and partially complete) algorithm for the ATMS task, namely finding the minimal supports for all propositions in a knowledge-base, on a singly connected problem model. With each proposition, the algorithm maintains a set of minimal-support labels to be used for generating the minimal support-sets at query time. We provide a worse case time and space analysis of the algorithm as a function of the number of variables, n, the degree, d, the number of values, k, the number of assumptions, N_A , and the size of support labels, s (obviously all these parameters are not independent). The results are summarized in table 1.

N_l indicates the space complexity, T is the time complexity, while, N_s, gives a lower bound on the time complexity of ATMS as well as the space complexity of algorithms that maintain all the minimal-support sets in a compiled fashion (as in De-Kleer's ATMS). Clearly the lower bounds also limit the performance of any complete algorithm.

	$s = 1$	$s > 1$		
		$k - 1 < \frac{d}{2}$	$k - 1 > \frac{d}{2}$	
			$k - 1 < d$	$k - 1 > d$
N_l	d	$0\left(\frac{d^d}{k(d-k+1)^{d-k+1}}\right)$	$k^{\frac{d}{2}-1} \cdot 2^d$	
N_s	$\sum_{i=0}^{dpt} d^i$	$0(N_l^{(k-1)^{dpt}})$	$0(N_l^{(k-1)^{dpt}})$	$0(N_l{}^{d^{dpt}})$
T	$\sum_{i=0}^{dpt} d^i$	$0(N_s \cdot 2^{(k-1)^2})$	$0(N_s \cdot 2^{(k-1)^2})$	$0(N_s \cdot 2^{(k-1)d})$

Table 1

The table reveals that even on this simplified model, ATMS is $\exp(\frac{d}{2}(k-1)^{\log_d n})$ (substitute (5) for N_l), and thus suggests that, at least on the bassis of a worse-case analysis, the ATMS, **cannot be regarded** as an efficient scheme. Interestingly, the JTMS on this tree model is linear [Dechter 1988.]

The analysis provides a characterization of special cases in which the ATMS is tractable (at least on this model). When the size of support labels is one, the problem becomes linear and the algorithm is complete and optimal. This case quantify the effect of the horn clause restriction on the tractability of this task. The algorithm is also complete for non-binary singly-connected networks, and the number of support sets in that case is exp $((sc-1)^{dpt})$.

The algorithm presented can be generalized to work on arbitrary networks via the tree-clustering transformation [Dechter 1987] in which case its complexity is also exponential in the

size of the largest cluster. The details of this extension are beyond the scope of this paper.

References

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

[Dechter 1988a] Dechter, R. and A. Dechter, "Belief maintenance in dynamic constraint networks," UCLA, L.A., Cal., Tech. Rep. R-108, 1988.

[Dechter 1988b] Dechter, R., "Algorithms for ATMS," UCLA, L.A., Cal., Tech. Rep. R-109 (in preperation), 1988.

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

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

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

[Reiter 1987] Reiter, R. and J. De-Kleer, "Foundations of assumption-based Truth Maintenance Systems: Preliminary report.," in *Proceedings AAAI-87,* Seattle Washington: 1987.