On computing minimal models*

Rachel Ben-Eliyahu
Computer Science Department
Technion — Israel Institute of Technology
Haifa 32000
Israel
rachelb@cs.technion.ac.il

Rina Dechter
Information & Computer Science
University of California
Irvine, California 92717
USA
dechter@ics.uci.edu

September 18, 1994

*This work was partially supported by an IBM graduate fellowship to the first author, by NSF grants IRI-9157636 and IRI-9200918, by Air Force Office of Scientific Research grant AFOSR 900136, by a grant from Xerox Palo Alto research center, and by Toshiba of America. Part of this work was done while the first author was a graduate student at the Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, California, USA.

Abstract

This paper addresses the problem of computing the minimal models of a given CNF propositional theory. We present two groups of algorithms. Algorithms in the first group are efficient when the theory is almost Horn, that is, when there are few non-Horn clauses and/or when the set of all literals that appear positive in any non-Horn clause is small. Algorithms in the other group are efficient when the theory can be represented as an acyclic network of low-arity relations. Our algorithms suggest several characterizations of tractable subsets for the problem of finding minimal models.

1 Introduction

One approach to attacking NP-hard problems is to identify *islands of tractability* in the problem domain and to use their associated algorithms as building blocks for solving hard instances, often approximately. A celebrated example of this approach is the treatment of the propositional satisfiability problem.

In this paper, we would like to initiate a similar effort for the problem of finding one, all, or some of the *minimal models* of a propositional theory. Computing minimal models is an essential task in many reasoning systems in Artificial Intelligence, including propositional circumscription [McC80, McC86, Lif85] and minimal diagnosis [Rei87, dKMR92], and in answering queries posed on logic programs (under stable model semantics [GL91, BNNS91]) and deductive databases (under the generalized closed-world assumption [Min82]). While the ultimate goal in these systems is not to compute minimal models but rather to produce plausible inferences, efficient algorithms for computing minimal models can substantially speed up inference in these systems.

Special cases of this problem have been studied in the diagnosis literature and, more recently, the logic programming literature. Algorithms used in many diagnosis systems [dKW87, dKMR92] are highly complex in the worst case: To find a minimal diagnosis, they first compute all prime implicates of a theory and then find a minimal cover of the prime implicates. The first task is output exponential, while the second is NP-hard. Therefore, in the diagnosis literature, researchers have often compromised completeness by using a heuristic approach. The work in the logic programming literature (e.g. [BNNS91]) focused on using efficient optimization techniques, such as linear programming, for computing minimal models. A limitation of this approach is that so far it did not address the issue of worst-case and average-case complexities.

We want to complement these approaches by studying the task of finding all or some of the minimal models in general, independent of any specific domain. We will use the "tractable islands" methodology to provide more refined worst-case guarantees. The two primary "islands" that we use are *Horn theories* and *acyclic theories*. It is known that Horn theories have a unique minimal model that can be found in linear time [DG84, IM87]. Our near-Horn algorithms try to associate an input theory with a "close" Horn theory, yielding algorithms whose complexity is a function of this "distance".

For acyclic theories, we will show that while finding one or a subset of the minimal models can be done in output-polynomial time, the task of finding all minimal models is more complex. We will set up necessary and sufficient conditions under which an acyclic theory has a unique minimal model. Testing the conditions for unique minimal model and generating this model (if it exists) can be done in time polynomial in the size of the acyclic theory. We will also present a tree-algorithm that generates all the minimal models of an acyclic theory.

Once we have an efficient algorithm for generating minimal models of tree-like theories, we can apply it to any arbitrary theory by first compiling the theory into a tree. The resulting complexity will often be dominated by the complexity of this compilation process and will be less demanding for "near-tree" theories.

2 Preliminary definitions

A clause is *positive* if it contains only positive literals and is *negative* if it contains only negative literals. We will sometimes refer to a clause as a set of literals. In this paper, a *theory* is a set of clauses. Given a theory Φ, *size*(Φ) will denote its length, that is, the number of symbols in the theory. Given a set A, $|A|$ denotes its cardinality. A set of literals *covers* a theory iff it contains at least one literal from each clause in the theory. A set of covers of a theory is *complete* iff it is a superset of all minimal covers of the theory.

A theory is called *positive* if it is composed of positive clauses only. Given a theory Φ and a set of literals S, the operation $\Phi \oslash S$ performs *unit propagation* on $\Phi \bigcup S$. *Unit propagation* is the process where given a theory Φ, you do the following until Φ has no unit clauses: you pick a unit clause C from Φ, delete the negation of C from each clause and delete each clause that contains C. Itai and Makowsky [IM87] have shown that unit propagation can be done in time which is linear in the size of the theory[1]. For each theory Φ, $nf(\Phi)$ denotes $\Phi \oslash \emptyset$. For each model M, $pos(M)$ denotes the set of atoms to which M assigns **true**. We will sometimes refer to a model as a set of literals, where a negative literal $\neg P$ in the model means that the model assigns **false**

[1]Itai and Makowsky worked with Horn theories but their method applies also to general theories.

to P and a positive literal P in the model means that the model assigns **true** to P.

Definition 2.1 **(X-minimal model)** *Let Φ be a theory over a set of atoms $\mathcal{L}$, $X \subseteq \mathcal{L}$, and M a model for Φ. M is an X-*minimal *model for Φ iff there is no other model M' for Φ such that $pos(M') \bigcap X \subset pos(M) \bigcap X$. If M is an X-minimal model for $X = \mathcal{L}$, it will be called simply a* minimal model.

3 A general algorithm for computing a minimal model

Cadoli [Cad92] has shown that the problem of finding an X-minimal model for a theory is $P^{NP[O(\log n)]}$-hard. Roughly, this means that it is at least as hard as problems that can be solved by *some* deterministic polynomial algorithm that uses $O(\log n)$ calls to an NP oracle. In Figure 1 we show an algorithm for computing X-minimal models that takes $O(n^2)$ steps and uses $O(n)$ calls to an NP oracle (where n is the number of variables in the theory). In Figure 2 we show a variation of this algorithm that uses a procedure for satisfiability called *model-sat* that also returns a model in case the theory is satisfiable.

Lemma 3.1 *Algorithm Find-X-minimal is correct.*

Proof: It is clear that the algorithm returns *false* iff Φ is inconsistent, and *true* otherwise. It is left to show that when Φ is satisfiable, at the end of the execution of the algorithm M is an X-minimal model. Assume by contradiction that M is not an X-minimal model. So there must be a model M' such that $M' \bigcap X \subset M \bigcap X$. Let k be minimal such that $M'(P_k) = \textbf{false}$ and $M(P_k) = \textbf{true}$. Since k is minimal, and by the way the algorithm works (statement 4), it must be that $\{\neg P_k\} \bigcup \Phi \bigcup \{P_i \mid M(P_i) = \textbf{true}, 1 \leq i < k\} \bigcup \{\neg P_i \mid M(P_i) = \textbf{false}, 1 \leq i < k\}$ is consistent and therefore it should be the case that $M(P_k) = \textbf{false}$, a contradiction. $\Box$

Lemma 3.2 *Algorithm Find-X-minimal2 is correct.*

Proof: Similar to the proof of Lemma 3.1. $\Box$

Algorithm Find-X-minimal2 suggests the following theorem which will be used in proofs in the following sections:

Find-X-minimal(Φ, X, M)

Input: A theory Φ and a subset X of the variables in Φ.

Output: **true** if Φ is satisfiable, **false** otherwise. In case Φ is satisfiable, the output variable M is an X-minimal model for Φ.

1. If $\neg$sat(Φ) return **false**;

2. For $i := 1$ to n do $M[i] :=$ **false**;

3. Let $P_1, ..., P_n$ be an ordering on the variables in Φ such that the first $|X|$ variables are all the variables from X.

4. For $i := 1$ to n do
If sat$(\Phi \bigcup \{\neg P_i\})$ then $\Phi := \Phi \oslash \{\neg P_i\}$
else $\Phi := \Phi \oslash \{P_i\}$, $M[i] :=$ **true**;

5. return **true**;

Figure 1: Algorithm Find-X-minimal

Theorem 3.3 *Let C be a class of theories over a language $\mathcal{L}$ having the following properties:*

1. *There is an algorithm α such that for any theory $\Phi \in C$, α decides whether Φ is satisfiable and produces a model for Φ (if there is one) in time $O(t_C)$.*
2. *C is closed under instantiation, that is, for every $\Phi \in C$ and for every literal L in $\mathcal{L}$, $\Phi \oslash \{L\} \in C$.*

Then for any theory $\Phi \in C$, an X-minimal model for Φ can be found in time $O(|X| t_C)$.

Proof: Follows from the correctness of Algorithm Find-X-minimal2. $\Box$

Since satisfiability of a 2-CNF theory can be checked in time linear in its size [SES76], we have the following corollary:

Corollary 3.4 *An X-minimal model for a 2-CNF theory Φ can be found in time $O(|X| * l)$, where l is the legth of the theory.*

Find-X-minimal2(Φ, X, M)

Input: A theory Φ and a subset of the variables in Φ, X.

Output: **true** if Φ is satisfiable, **false** otherwise. In case Φ is satisfiable, the output variable M is an X-minimal model for Φ.

1. If $\neg$model-sat(Φ, M) return **false**;

2. $negX := \{P|P \in X, \neg P \in M\}$;
$X := X - negX$;
$\Phi := \Phi \bigcup \{\neg P | P \in negX\}$;

3. While $X \neq \emptyset$ do

a. Let $P \in X$;

b. If $\neg$model-sat$(\Phi \bigcup \{\neg P\}, M')$ then $\Phi := \Phi \oslash \{P\}$
else $\Phi := \Phi \oslash \{\neg P\}$, $M := M'$;

c. $X := X - \{P\}$;

4. return **true**;

Figure 2: Algorithm Find-X-minimal2

However, using a straightforward reduction from VERTEX COVER [Kar72], we can show that if we are interested in finding a minimum cardinality model for a 2-CNF theory (namely, a model that assigns **true** to a minimum number of atoms), the situation is not so bright:

Theorem 3.5 *The following decision problem is NP-complete: Given a* positive *2-CNF theory* Φ *and an integer* K*, does* Φ *have a model of cardinality* $\leq K$*?*

4 Algorithms for almost-Horn theories

In this section, we present algorithms for computing minimal models of a propositional theory which are efficient for almost Horn theories. The basic idea is to instantiate as few variables as possible so that the remaining theory will be a Horn theory and then find a minimal model for the remaining theory in linear time.

4.1 Algorithm for theories with only a few non-Horn clauses

Algorithm MinSAT (Figure 3) is efficient when most of the theory is Horn and there are only few non-Horn clauses. Given a theory, MinSAT works as follows: It first tries to solve satisfiability by unit propagation. If the empty clause was not generated and no positive clause is left, the theory is satisfiable, and the unique minimal model assigns **false** to the variables in the remaining theory. If a nonempty set of positive clauses is left, we compute a cover for the remaining set of positive clauses, replace them with the cover, and then call MinSAT recursively on the new theory. If the theory is not satisfiable, or if we are interested in all minimal models, we have to call MinSAT again with a different cover.

Algorithm MinSAT is shown in Figure 3. The procedure $UnitInst(\Phi, I, Sat)$ takes as input a theory Φ and returns $nf(\Phi)$. I is an output variable which contains the set of unit clauses used for the instantiations. Sat is false iff the empty clause belongs to the normal form; otherwise Sat is true. The procedure $combine(I, M)$ takes as input a set of literals I and a set of sets of literals M and returns the set $\{S | S = W \bigcup I, W \in M\}$.

We group all the propositional theories in classes $\Psi_0, \Psi_1, \ldots$ as follows:

- $\Phi \in \Psi_0$ iff $nf(\Phi)$ has no positive clauses or contains the empty clause.
- $\Phi \in \Psi_{k+|C|}$ iff for each A that is a complete set of covers for C, where C is the set of positive clauses in $nf(\Phi)$, and for each S in A, $\Phi \oslash S$ belongs to Ψ_j for some $j \leq k$.

Observation 4.1 *If a theory has k non-Horn clauses it belongs to the class Ψ_j for some $j \leq k$. Hence, Horn theories belong to Ψ_0.*

We can show the following:

Lemma 4.2 *Algorithm MinSAT is correct.*

Proof: The proof goes by induction on the minimal k such that Φ belongs to Ψ_k.

MinSAT(Φ, M)

Input: A theory Φ.

Output: **true** if Φ is satisfiable, **false** otherwise. In case Φ is satisfiable, the output variable M will contain a set of models for Φ that is a superset of *all* the *minimal* models of Φ.

1. Φ :=UnitInst(Φ, I, Sat); If not Sat return **false**;

2. If Φ contains no positive clauses then
begin
$M := \{I \bigcup \{\neg P | P$ is a variable in $\Phi\}\}$; return **true**;
($*$ Note that M is a set of sets $*$)
end.

3. $M := \emptyset$; Let A be a complete set of covers for the set of all the positive clauses in Φ.
For each $S \in A$ do:
If MinSAT$(\Phi \bigcup S, M')$ then
$M := M \bigcup$ combine(I, M');

4. If $M = \emptyset$ then return **false** else return **true**;

Figure 3: Algorithm MinSAT

case $k = 0$ If $nf(\Phi)$ contains the empty clause then $UnitInst$ will return **false**, and so MinSAT will return **false**. Else if Φ has no positive clause then the model that assigns **false** to all the atoms is the minimal model of Φ and this is the model which the algorithm returns in Step 2.

Induction step Let Φ belong to some Ψ_k such that $k > 0$. It is easy to see that each minimal model of Φ can be represented as a union of two sets B and C, where B is a minimal cover of the positive clause of Φ and C is a minimal model of the theory $\Phi \bigcup \{P \mid P \in B\}$. Hence the result follows.

Proposition 4.3 *If $\Phi \in \Psi_k$ then MinSAT runs in time $O(nm^k)$, where n is the length of the input and m the maximum number of positive literals that* *appear in any clause.*

Proof: By induction on k. Case $k = 0$ — easy. Suppose $\Phi \in \Psi_k$ for some $k > 0$, and let C be the set of positive clauses in Φ. Hence for every S which is a cover of C $\Phi \bigcup S$ is in $\Psi_{k-|C|}$. So by the induction hypothesis each call to MinSAT($\Phi \bigcup S, M'$) takes $O(nm^{k-|C|})$ steps. A complete set of covers for C can be found in $O(m^{|C|})$ steps and there are at most $m^{|C|}$ covers in this set, therefore the algorithm runs in $O(nm^k)$ steps. $\square$

This is also the worst case complexity for deciding satisfiability using MinSAT. Since for every k the class Ψ_k is closed under instantiation, we can use Theorem 3.3 to prove that:

Proposition 4.4 *If a theory Φ belongs to the class Ψ_k for some k, then an X-minimal model for Φ can be found in time $O(|X|nm^k)$.*

Algorithm MinSAT returns a superset of all the minimal models. To identify the set of all minimal models, we need to compare all the models generated. Since there are at most m^k models generated by MinSAT, each of size at most n, the complexity of finding all minimal models for a theory in the class Ψ_k is $O(nm^{2k})$.

4.2 Algorithms that exploit the interaction between the positive literals of the theory

In this section we will identify tractable subsets for *satisfiability* and for finding all minimal models by using topological analysis of the interactions between the positive literals of the theory. The *positive graph* of a theory, defined next, reflects on these interactions.

Definition 4.5 (positive graph of a theory) *Let Φ be a theory. The* positive graph *of Φ is an undirected graph (V, E) defined as follows:*

***V*=** $\{P|P$ *is a positive literal in some clause in* $\Phi\}$,

***E*=** $\{(P,Q)|$ P *and* Q *appear positive in the same clause*$\}$.

Note that Φ is a Horn theory iff its positive graph has no edges.

Definition 4.6 (vertex cover) *Let $G = (V, E)$ be a graph. A* vertex cover *of G is a set $V' \subseteq V$ such that for each $e \in E$ there is some $v \in V'$ such that $v \in e$.*

We take "vertex cover of the theory" to mean "vertex cover of the positive graph of the theory".

An algorithm that computes a superset of all minimal models based on a vertex cover of a theory can consider all possible instantiations of the variables in the cover. Each such instantiation yields a Horn theory for which we can find a minimal model (if there is one) in linear time. When we combine the model for the Horn theory with the cover instantiation, a model of the original theory results. We can show that a superset of all minimal models of a theory can be generated in this way. If we are interested only in deciding satisfiability, we can stop once the first model is found. Hence,

Theorem 4.7 *If the positive graph of a theory Φ has a vertex cover of cardinality c, then the satisfiability of Φ can be decided in time $O(n2^c)$, where n is the size of the theory, and an X-minimal model for Φ can be found in time $O(|X|n2^c)$. The set of all minimal models of Φ can be found in time $O(n2^{2c})$.*

Proof: Let Φ be a theory and V a vertex cover of the positive graph of Φ such that $|V| = c$. Φ is consistent iff there is an instantiation of the variables in V such that the remaining Horn theory is consistent. Since there are 2^c instantiations of V, and since deciding consistency of a Horn theory is linear, we get that satisfiability of Φ can be decided in $O(n2^c)$ steps. Since the set of all theories having a positive graph with a vertex cover of size c for some c is closed under instantiation, By Theorem 3.3 we can find a minimal model for Φ in time $O(|X|n2^c)$. For the rest we will first show that every minimal model M of Φ can be represented as the union of I and H where $I = \{P \mid P \in V, P \text{ was instantiated by some instantiation } f \text{ to } \mathbf{true}\}$ and H is the minimal model of the Horn theory we get after the instantiation f. Let $I = \{P \mid P \in V, M(P) = \mathbf{true}\}$. Since M is minimal, $H = M - I$ must be a minimal model of $\Phi \bigcup I$.

So a super set S of all the minimal models of Φ can be found in time $O(n2^c)$, and we have also $|S| \leq 2^c$. We then have to compare each 2 models found to identify the minimal ones, and so the whole process takes $O(n2^{2c})$ steps. $\square$

VC-minSAT(Φ, M, G)

Input: A theory Φ and a positive graph of Φ, G.

Output: **true** if Φ is satisfiable, otherwise **false**. If Φ is satisfiable, M contains a superset of all minimal models for Φ.

1. $\Phi := UnitInst(\Phi, I, Sat)$;

2. If $\neg Sat$ return **false**; $G := Update(\Phi, G)$;

3. If G has no arcs then
begin $M := I \bigcup \{\neg P | P$ appears is a variable in $\Phi\}$; return **true**; *end.*

4. $P := MaxDegree(G)$; $Sat :=$ **false**; $M = \emptyset$;

5. If VC-minSAT($\Phi \bigcup \{P\}, M^+, G$) then
$M := combine(I, M^+)$;

6. If VC-minSAT($\Phi \bigcup \{\neg P\}, M^-, G$) then
$M := M \bigcup combine(I, M^-)$;

7. If $M == \emptyset$ return **false** else return **true**.

Figure 4: Algorithm VC-minSAT

In general, the problem of finding a minimum-cardinality vertex cover of a graph is NP-hard. A greedy heuristic procedure for finding a vertex cover could simply remove the node with maximum degree from the graph and continue with the reduced graph until all nodes are disconnected. The set of all nodes removed is a vertex cover.

Algorithm VC-minSAT (Figure 4) integrates the above heuristic into a backtrack algorithm for finding the minimal models. $MaxDegree$ takes the positive graph as an input and returns an atom (node) that has a maximum degree. If there is more than one such atom, it chooses the one that appears in a maximum number of non-Horn clauses in the theory. $Update(\Phi, G)$ returns the positive graph of Φ. We can show that algorithm VC-minSAT produces a superset of all the minimal models.

Another approximation algorithm for finding a vertex cover is based on the idea of "maximal matching" (see [Gav]). That approximation algorithm is guaranteed to find a vertex cover that is at most twice as large as a minimum one, and it can be also combined with a backtrack algorithm for finding the minimal models, similar to the way we combine two heuristics in Algorithm VC-minSAT.

We should mention here that the idea of initializing variables in a theory until the remaining theory is Horn has been suggested, in the context of solving the satisfiability problem, by Gallo and Scutella [GS88] and was recently extended by Dalal and Etherington [DE92]. The advantages of our approach are that we provide an intuitive criteria for how the variables to be instantiated are selected and we classify the performance of the algorithm using a well-understood and largely explored graphical property, vertex cover.

Also note that we could define the *negative graph* of a theory just as we defined the positive graph. We could then write an algorithm that is analogous to VC-minSAT and is efficient for deciding satisfiability of theories for which the negative graph has a small vertex cover. Clearly, algorithm minSAT has also an analogous algorithm that considers negative instead of positive clauses.

Comment 4.8 *To clarify the difference between algorithm VC-minSAT and algorithm MinSAT, consider the following classes of theories Γ_1 and Γ_2. Class Γ_1 is the class of all theories of the form*

$$\{A_1 \vee D, \neg A_1 \vee A_2 \vee D, \neg A_2 \vee A_3 \vee D, ..., \neg A_n \vee A_{n+1} \vee D\}$$

and class Γ_2 is the class of all theories of the form

$$\{A_1 \vee B_1, \neg D \vee A_1 \vee B_1, ..., \neg D \vee A_m \vee B_m\}$$

where n and m are arbitrary positive integers, and D and all A_i's and B_i's are atoms. An arbitrary theory in Γ_1 has a positive graph with a vertex cover of size 1 (take $\{D\}$ as a vertex cover), but might belong to Ψ_k for some $k \geq n$. On the other hand, an arbitrary theory from Γ_2 belongs to Ψ_2 but might have a minimum vertex cover of size m. Therefore, in general algorithm MinSAT is better than algorithm VC-minSAT for the class Γ_2, but algorithm VC-minSAT is better than MinSAT for the class Γ_1.

5 Computing minimal models on acyclic networks of relations

In this section we provide efficient algorithms for theories that can be represented as acyclic relations of low arity. We next define the notions of *constraint networks* and *relations* and show how they can represent propositional theories and their satisfying models. We chose to switch to the language of relations and constraint networks since the notions of acyclicity and topological-based tractability was developed primarily within this framework [Mai83, Dec92].

Definition 5.1 (relations, networks, schemes) *Given a set of variables $X = \{X_1, ..., X_n\}$, each associated with a domain of discrete values $D_1, ..., D_n$, respectively, a* relation *(or, alternatively, a* constraint*) $\rho = \rho(Y_1, ..., Y_k)$ is any subset*

$$\rho \subseteq D_1 \times D_2 \times ... \times D_k.$$

where $k \leq n$ and for each $1 \leq i \leq k$ $Y_i \in X$ and D_i is the domain of Y_i. The projection *of ρ onto a subset of variables R, denoted $\Pi_R(\rho)$ or ρ_R, is the set of tuples defined on the variables in R that can be extended to a tuple in ρ. A* constraint network *N over X is a set $\rho_1, \ldots, \rho_t$ of such relations. Each relation ρ_i is defined on a subset of variables $S_i \subseteq X$. We also denote by $\rho(S_i)$ the relation specified over S_i. The set of subsets $S = \{S_1, .., S_t\}$ is called the* scheme *of N. The network N represents a unique relation $rel(N)$ defined over X, which stands for all consistent assignments (or all solutions), namely,*

$$rel(N) = \{x = (x_1, ..., x_n) | \ \forall S_i \in S, \Pi_{S_i}(x) \in \rho_i\}.$$

A partial assignment $T = t$ is a value assignment to a subset of variables $T \subseteq X$. The operator $\bowtie$ is the join operator in relational databases defined as follows. Let ρ_1 and ρ_2 be two relations defined over the variable sets S_1 and S_2, then $\rho_1 \bowtie \rho_2$ is a relation over $S_1 \bigcup S_2$, and $t \in \rho_1 \bowtie \rho_2$ iff $t_{S_1} \in \rho_1$ and $t_{S_2} \in \rho_2$.

Any propositional theory can be viewed as a special kind of constraint network, where the domain of each variable is $\{0, 1\}$ (corresponding to $\{$**false**,**true**$\}$

and where each clause specifies a constraint (in other words, a relation) on its propositional atoms. The *scheme of a propositional theory* is accordingly defined as the scheme of its corresponding constraint network, and the set of all models of the theory is the set of all solutions of its corresponding constraint network.

Example 5.2 *Consider the theory* $\Phi = \{\neg A \vee \neg B, \neg B \vee \neg C, C \vee D\}$. *This theory can be viewed as a constraint network over the variables* $\{A, B, C, D\}$, *where the corresponding relations are the truth tables of each clause, that is,* $\rho(AB) = \{00, 01, 10\}$, $\rho(BC) = \{00, 01, 10\}$, *and* $\rho(CD) = \{01, 10, 11\}$. *The scheme of the theory* Φ *is* $\{AB, BC, CD\}$. *The set of all solutions to this network (and hence the set of models of* Φ*) is*

$$\rho(ABCD) = \{0001, 0010, 0011, 0101, 1001, 1010, 1011\}.$$

Note that Φ *has two minimal models:* $\{0001, 0010\}$.

The scheme of a theory can be associated with a *constraint graph* (also called a *dual* constraint graph [DP89] or an *intersection* graph [Mai83]) where each relation in the scheme is a node in the graph and two nodes are connected iff the corresponding relations have variables in common. The arcs are labeled by the common variables. For example, the constraint graph of the theory Φ of Example 5.2 is as follows:

```
      BC
      /\
  B /    \ C
   /      \
  AB      CD
```

In general, theories that correspond to a constraint graph that is a tree are called *acyclic theories*, and their corresponding tree-like constraint graph is called a *join tree*. In the following we make these notions more precise.

Sometimes a constraint network looks *cyclic* while, by removing some of its redundant arcs it will become acyclic. Removing redundant arcs does not change the problem, and identifying such arcs can be done in linear time in the size of the network [Mai83].

If by eliminating some redundant arcs from its intersection graph the constraint graph becomes a tree, then we say that the network is *acyclic*, and call the resulting tree a *join-tree*. An acyclic network may have more than one join-tree. Here is a formal definition.

Definition 5.3 *[Mai83]*
Given a set of relations $\rho_1, ..., \rho_p$ having the scheme $\{S_1, ..., S_p\}$, a join-graph is an arc-subgraph[2] of the constraint graph over $S_1, ..., S_p$ (called intersection graph in [Mai83]), satisfying that if $X \in S_i \cap S_j$ then there exist a path between S_i and S_j whose all labeled arcs contain X. A join tree is a join graph that is a tree. A network of relations is acyclic if it has a join-tree.

We next define the concept of *pair-wise consistency*.

Definition 5.4 (pair-wise consistency [Mai83]) *A pair of relations ρ_1, ρ_2 are pair-wise consistent iff every tuple in ρ_1 can be consistently extended by a tuple in ρ_2 and vice-versa. Formally, iff*

$$\rho_1 = \Pi_{S_1}(\rho_1 \bowtie \rho_2), \ \textit{and} \ \rho_2 = \Pi_{S_2}(\rho_1 \bowtie \rho_2).$$

It was shown that testing or enforcing pair-wise consistency can be done in polynomial time [Mai83]. Pair-wise consistency parallels the notion of *arc-consistency* developed in the context of binary constraint networks [Mac85].

We next present two algorithms for computing minimal models for *acyclic* theories. These algorithms will be extended to arbitrary theories via a procedure known as *tree-clustering* [DP89], which compiles any theory into a join tree of relations. The tree-clustering algorithm is reviewed in Appendix A.

Consequently, given a general theory, the algorithms presented next work in two steps: First, a join-tree is computed by tree-clustering, and then a specialized tree-algorithm for computing the minimal models is applied. The complexity of tree-clustering is exponential in the size of the maximal arity of the generated relations, and hence our algorithms are efficient for theories that can be compiled into networks of low-arity relations. We should note, however, that even in the cases where tree-clustering is time expensive, it might still be useful since it offers a systematic way of representing the models of the theory in a hierarchical structure capable of supporting information retrieval without backtracking. We say that an ordering of a variables is

[2]An arc-subgraph of a graph is a graph that contains a subset of the arcs but all the nodes.

backtrack free relative to a given set of relations iff the variables can be instantiated to a solution of the network with no dead ends, while at each step consulting those constraints defined over the relevant variables. The notion of *backtrack-free search* is identical to *monotonicity of join plans* [Mai83].

5.1 Finding a subset of all minimal models

For the rest of Section 5, we will assume that we are dealing with constraint networks that correspond to propositional theories, and hence the domain of each variable is $\{0, 1\}$ and we have the ordering $1 \succ 0$. We will also assume that we are looking for models that are minimal over all the atoms in the language of the theory, namely, X-minimal models where X is the set of all atoms in the theory.

Definition 5.5 *Given a relation ρ defined on a set of variables X, and given two tuples r and t in ρ, we say that $t \succ r$, iff for some X_0 in X, $t_{X_0} \succ r_{X_0}$ and, for all $X_i \in X$, $t_{X_i} \succ r_{X_i}$ or $t_{X_i} = r_{X_i}$. We say that t and r agree on a subset of variables $S \subseteq X$ iff $r_S = t_S$.*

Definition 5.6 (conditional minimal models) *Given a relation ρ over X and a subset of variables $S \subseteq X$, a tuple $t \in \rho$ is* conditionally minimal *w.r.t. S iff there does not exist $r \in \rho$ such that r agrees with t on S and $t_{X-S} \succ r_{X-S}$. The set of all conditional minimal models (tuples) of ρ w.r.t. $S = s$ is denoted by $min(\rho|S = s)$. The set of all conditional minimal models (tuples) of ρ w.r.t. S is denoted $min(\rho|S)$ and is defined as the union over all possible assignments s to S of $min(\rho|S = s)$. $min(\rho|\emptyset)$ is abbreviated to $min(\rho)$.*

Example 5.7 *Consider the relation*

$$\rho(ABCD) = \{0111, 1011, 1010, 0101, 0001\}.$$

In this case, we have $min(\rho) = \{1010, 0001\}$, $min(\rho|\{C, D\}) = \{0111, 1011, 1010, 0001\}$, and $min(\rho|\{A\}) = \{0001, 1010\}$.

One can verify that: (1) any minimal tuple of a projection $\Pi_S(\rho)$ can be extended to a minimal tuple of ρ, but not vice versa, namely a minimal models of ρ may not be minimal model of $\Pi_S(\rho)$; (2) a conditionally minimal tuple is not necessarily a minimal tuple; and (3) a minimal tuple is a conditional minimal tuple w.r.t. to all subsets. The following lemma justify our forthcoming algorithms.

Lemma 5.8 *Let ρ_Y and ρ_Z be two relations over Y and Z, respectively, let $X = Y \cup Z$ and let $T = Y \cap Z$. Let $\rho = \rho_Y \bowtie \rho_Z$. then,*

1. *For every $t \in \rho_T$, $min(\rho|T = t) = min(\rho_Y|T = t) \bowtie min(\rho_Z|T = t)$*
2. *$min(\rho) \subseteq min(\rho_Y|T) \bowtie min(\rho_Z|T)$*
3. *If $T = t$ is minimal over ρ_T then $min(\rho) \supseteq min(\rho|T = t)$. Consequently,*
$$min(\rho) \supseteq min(\rho_Y|T = t) \bowtie min(\rho_Z|T = t).$$

Proof: 1. It is clear that $min(\rho|T = t) \supseteq min(\rho_Y|T = t) \bowtie min(\rho_Z|T = t)$. We will show the other direction. Let $t\prime \in min(\rho|T = t)$. By definition, $t\prime_T = t$. Clearly, $t\prime_Y \in min(\rho_Y|T = t)$ or else, there is an extension of $t\prime_T$ to variables in Y that is smaller than $t\prime$, contradicting its conditional minimality. The argument for Z is identical. 2. Follows immediately from the fact that a minimal model is always a conditional minimal model relative to any subset. Thus, $min(\rho) \subseteq min(\rho|T = t)$ and from part 1 the claim follows. 3. Assume t is a minimal model of ρ_T and let $t\prime$ be a conditional minimal model relative to t, namely $t\prime \in min(\rho|T = t)$. We claim that $t\prime \in min(\rho)$. Else, there is another tuple t^0 that is smaller than $t\prime$ in ρ. It cannot be the case that ${t^0}_T$ is smaller than t since t is minimal relative to T. Consequently, ${t^0}_T = t\prime_T = t$. The extension of ${t^0}_T$ to Y cannot be smaller than $t\prime$'s since $t\prime$ is conditionally minimal relative to $T = t$, yielding a contradiction. $\Box$.

We are now ready to show that given a join-tree, a subset of all minimal models can be computed in output linear time. The idea is as follows: Once we have a rooted join-tree which is pair-wise consistent we can take all the minimal tuples in the relation of the root node and extend them (via the join operation) with the matching conditional minimal tuples of the child node, conditioned on variables common to both. This can be continued until we reach the leaves. It can be shown that all the models computed in this way are minimal and that they are generated in a backtrack-free manner; however, not *all* minimal models will be generated. In order to enlarge the set of minimal models captured, we can re-apply the procedure where each

node serves as a root. The following notations will be used in the rest of the sections:
N is an input network of constraints
ρ is the relation associated with N.
T_N is a compiled join-tree of N.
X is the set of variables $X_1, ..., X_n$, of N.
d is the maximal size of a variable domain.
$S = \{S_1, ..., S_r\}$ is the scheme of the join-tree.
ρ_i is the relation associated with S_i.
n is the number of variables in N.
r is the number of relations in the join-tree, T_N ($r \leq n$)
p is the maximal arity of each relation in the join-tree.
k is the maximal number of tuples in each relation of T_N. ($k \leq d^p$).
t is the overall number of tuples in the join-tree ($t \leq r \cdot k$).

Definition 5.9 (parents of S) *Given a scheme $S = \{S_0, ..., S_r\}$ of a rooted join-tree, T_N, $S_{p(i)}$ will denote the parent subset of S_i in the tree. We call an ordering $d = S_0, .., S_r$ a* tree-ordering *iff a parent node always precedes its child nodes.*

Definition 5.10 *Let T be a rooted join-tree with S_0 at the root. Let ρ_i be the relation associated with S_i and let $S_0, S_1, ..., S_r$ be a tree-ordering. We define*

$$\rho^0(T) = \bowtie_{i=0...r} (min(\rho_i | S_{p(i)})).$$

Theorem 5.11 *Let T be a rooted join-tree with a tree-ordering $S_0, S_1, ..., S_r$, then 1. $\rho^0(T)$ is a subset of all the minimal models of T, and 2. $\rho^0(T)$ can be computed in $O(l \cdot k \cdot r + r \cdot p \cdot k^2)$ steps where l is the number of minimal models in the output.*

Proof: 1. By induction on r. For $r = 0$ the claim clearly holds. Suppose $r > 0$. Let T^* be join-tree rooted at S_0 that includes all the subsets $S_0, ..., S_{r-1}$. By the induction hypothesis,

$$\rho^0(T^*) = \bowtie_{i=0..r-1} (min(\rho_i | S_{p(i)}))$$

is a subset of all minimal models of the rooted join tree T^*. Let $S^* = \bigcup_{i=0}^{r-1} S_i$. Assume t is in $\rho^0(T)$ but it is not minimal. So, there must be a tuple t'

in the relation described by T such that $t \succ t'$. It cannot be the case that $t_{S^*} \succ t'_{S^*}$ because this will be a contradiction to the induction hypothesis, hence, it must be the case that

$$t_{S^*} = t'_{S^*}, \tag{1}$$

therefore, in order for t' to be smaller it must obey $t_{S_r} \succ t'_{S_r}$. However, from (1) it follows that $t_{S_{P(r)}} = t'_{S_{P(r)}}$. Consequently, since $t_{S_r} \succ t'_{S_r}$, it must be that $t_{S_r} \notin min(\rho_r|S_{P(r)})$, a contradiction to the way $\rho^0(T)$ is defined.

To show that $\rho^i(T)$ is polynomially computable observe that a join tree can be made backtrack-free by enforcing pair-wise consistency between adjacent relations. This operation can be accomplished in $O(k \cdot logk)$ comparisons steps between the tuples of the two relations. Each comparison is $O(p)$, yielding $O(p \cdot k \cdot logk)$ steps, overall.

The conditional minimal subsets can be computed in $O(p \cdot k^2)$ steps. After that, the resulting relation can be computed in a backtrack-free manner from root to leaves, yielding an overall bound of $O(r \cdot pk^2 + l \cdot k \cdot r)$, where l is the number of the minimal models in the output. $\Box$

Example 5.12 *Consider the join-tree of the theory Φ in Example 5.2. Assuming BC is the root, we can use the tree-ordering $d = BC, AB, CD$. Since tuple $(BC = 00)$ is the only minimal model of $\rho(BC)$, it is selected. This tuple can be extended by $A = 0$ and by $D = 1$, resulting in one minimal model of ρ, namely the tuple $(ABCD = 0001)$. If AB plays the role of a root, we will still be computing the same minimal model. However, when CD plays the role of a root, we will compute the tuple $(ABCD = 0010)$, which is also a minimal model of ρ.*

From Theorem 5.11, it follows that, given an acyclic network or any general backtrack-free network relative to an ordering d, one minimal model can be computed in time that is linear in the size of the network, and the total subset of minimal models $\rho^0(T)$ can be computed in time proportional to the size of the set. We summarize this in algorithm **min1**, given in Figure 5. By adding the complexity of generating a join-tree (See Appendix A) to that of generating the minimal models we get:

Theorem 5.13 (complexity of min1)
*The complexity of **min1** is $O(r \cdot d^p + r^2 \cdot p \cdot k^2 + r^2 \cdot l \cdot k)$, where l is the*

min1(Φ)
Input: A theory Φ.
Output: A subset of all the minimal models of Φ.

1. Apply tree-clustering to Φ. If the theory is not satisfiable, stop and exit. Else, generate join-tree T. Apply pair-wise consistency to T.
2. For each node R in T and for the join tree T' rooted at R compute $\rho^0(T')$.
3. Output the union of all models computed.

Figure 5: Algorithm **min1**

number of minimal models in the output. For near-tree networks that can be embedded in trees whose relation's arity p *satisfies* $p = O(log_d n)$ ***min1****'s complexity is* $O(n^4 \cdot logn + n^3 \cdot l)$.

Proof: 1. We obtain this bound by adding together the cost of tree clustering ($O(r \cdot d^P)$) and the cost of computing $\rho^0(T)$ multiplied by r, since the process is restarted from any node in the tree as a root. 2. Clearly, since $p = O(log_d n)$, $k = O(n)$ and since $r \leq n$ we can substitute n for r, n for k, and $log_d n$ for p, yielding an overall complexity of $O(l \cdot n^3 + n^4 \cdot logn)$ $\Box$

Algorithm **min1** does not necessarily produce all minimal models as the following example shows.

Example 5.14 *Consider the join-tree where the variables are* $\{A, B, C, D, E, F, G\}$, *the scheme is a tree* $\{ABC, BCDEF, EFG\}$, *and the corresponding relations are* $\rho(ABC) = \{011, 110, 000\}$, $\rho(BCDEF) = \{11011, 10100, 00010\}$, *and* $\rho(EFG) = \{110, 000, 101\}$. *The reader can verify that the tuple* $\{0110110\}$ *is a minimal model for this network, but its projection relative to any of the relations is not minimal.*

We can characterize the models generated by **min1** as follows:

Proposition 5.15 *A minimal model* $t \in T$ *will be generated by algorithm* ***min1*** *iff there is a tree ordering* $\{S_1, ..., S_r\}$ *of* T *such that for each* $0 \leq i \leq r$ $t_{S_i} \in min(\rho(S_i)|S_{P(i)})$.

Proof: The proof follows immediately from the definition of $\rho^0(T)$. $\Box$

Clearly, if all minimal models of a network have the property specified in Proposition 5.15, all will be generated by **min1**. This condition has a limited use in general since it is not identifiable from the network's input. Since **min1** returns at least one model (if the network is consistent), it returns all minimal models of networks having just one unique minimal model. The following theorem shows that networks having unique minimal models can be identified in linear time.

Theorem 5.16 *A tree-network has a unique minimal model iff every relation in the network has a unique minimal tuple. In this case the minimal model can be generated by joining all the minimal tuples of all the relations.*[3]

Proof: Clearly, if t is a unique minimal model of the tree network, its projection on each relation in the tree is a unique minimal model in that relation.

The other direction is proved by induction on the number of relations in the tree $\rho(S_1), ..., \rho(S_n)$ (assume this is a tree ordering). Suppose each relation S_i has a unique minimal tuple. By induction, $\rho(S_1), ..., \rho(S_{n-1})$ has a unique minimal model m obtained by joining those tuples. Take t to be the projection of m on S_{n-1}, and $P = S_{n-1} \bigcap S_n$. By induction, t is the minimal tuple of $\rho(S_{n-1})$, and so there is no tuple in S_n which is smaller than t when both are projected on P. Let t' be the unique minimal tuple of $\rho(S_n)$. Since the network is pair-wise consistent and t' is unique, it must be the case that $t_P = t'_P$. Moreover, since t' is a unique minimal model in $\rho(S_n)$, t' must be the only tuple that extends t to get a minimal model. So the whole tree has a unique minimal model. $\Box$

5.2 Listing all minimal models

As we noted above, algorithm **min1** does not necessarily produce all minimal models. We now present a second algorithm, **min2** , that computes all the minimal models. More accurately, the algorithm computes all *conditional minimal models* of the network, when conditioning is with respect to the variables that are common to each node and its neighbors in the tree. Once all conditional minimal models of the root node are available the set of all

[3]remember that we assume pair-wise consistency all along.

minimal models can be generated by minimizing over their union. We will show that the algorithm is output polynomial (almost linear) w.r.t. the set of all *conditional minimal models*, but may not be optimal relative to the overall set of minimal models. The reason is that the set of all conditional minimal models associated with the root node might be quite large and may not be included in the final set of minimal models. It seems that the source of complexity for this task even for trees is that the notion of minimal models cannot be captured by a simple numerical function. In contrast, it was shown that finding the *minimal cardinality* models, can be computed in linear time for acyclic theories (see [DD88, FD94]).

In contrast to **min1**, algorithm **min2** computes partial conditional minimal models recursively while traversing the join-tree *bottom up*. When it visits a node S_i, and for each of its conditional minimal tuple s_i (conditioned on the variables common to the node and its neighbors), the algorithm prunes partial models in the subtree rooted at S_i that agree with s_i and which cannot be extended to a minimal model since they are not conditionally minimal. Let T_i be the network rooted at node S_i, let I_i be the set of all variables that S_i shares with its parent node, and let F_i be the set of all variables that S_i shares with its neighbors (i. e. children and parents). We associate each node S_i with two relations, θ_i, and Θ_i. The relation θ_i denotes the set of all minimal models in T_i conditioned on F_i. Namely,

$$\theta_i = min(rel(T_i)|F_i).$$

The relation Θ_i denotes all minimal models conditioned on I_i. Namely,

$$\Theta_i = min(rel(T_i)|I_i).$$

Since $I_i \subseteq F_i$, Θ_i can be computed from θ_i using:

$$\Theta_i = min(\theta_i|I_i) \qquad (2)$$

Note that for the root node, S_0, θ_0 is the set of all conditional minimal models relative to the set of variables in S_0 that appear in the children of S_0, while Θ_0 is the set of *all* minimal models (conditioning is on the empty set).

Lemma 5.17 *The relation θ_i can be expressed recursively as a function of $\Theta_{i_1}, ..., \Theta_{i_b}$ where $S_{i_1}, ..., S_{i_b}$ are S_i's children in the tree:*

$$\theta_i = min(\rho(S_i)|F_i) \bowtie \Theta_{i_1} ... \bowtie \Theta_{i_b}. \qquad (3)$$

Proof: Assume S_i is the root node and $S_{i_1}, ..., S_{i_b}$ are its children. By definition, for each $1 \leq j \leq b$

$$\Theta_{i_j} = min(rel(T_{i_j})|I_{i_j}) \tag{4}$$

Suppose $t \in min(\rho(S_i)|F_i) \bowtie \Theta_{i_1} ... \bowtie \Theta_{i_b}$, and we will show that $t \in \theta_i = min(rel(T_i)|F_i)$. Suppose by contradiction that there exist $t' \in rel(T_i)$ such that $t'_{F_i} = t_{F_i}$ and $t' \prec t$. Since t and t' agree on F_i, either $t'_{S_i} \prec t_{S_i}$, which is impossible because θ_i was computed by joining $min(\rho(S_i)|F_i)$, or there must be a relation $rel(T)$ where T is among $\{T_{i_1}, ..., T_{i_k}\}$ such that $t'_S \prec t_S$, where S is the set of variables in the subtree T. Since t and t' agree on F_i and $I_i \subseteq F_i$, it must be the case that $t'_{S-I_i} \prec t_{S-I_i}$, but that's a contradiction to (4).

To prove the other direction, suppose $t \in min(rel(T_i)|F_i)$, we want to show that $t \in min(\rho(S_i)|F_i) \bowtie \Theta_{i_1} ... \bowtie \Theta_{i_b}$. It is clear that

$$t_{S_i} \in min(\rho(S_i)|F_i),$$

and it is also clear that for each T_{i_j} in $\{T_{i_1}, ..., T_{i_k}\}$

$$t_{S_{i_j}} \in min(rel(T_{i_j})|I_{i_j})$$

where S_{i_j} is the set of variables in the subtree T_{i_j}. Hence t is in Θ_{i_j} and hence in θ_i as defined in (3). $\Box$

Lemma 5.17 above allows a bottom-up computation of Θ_i starting at the leaf nodes.

Algorithm **min2** is summarized in Figure 6.

Example 5.18 *Consider again the tree-network of Example 5.12. Algorithm* ***min2*** *will perform the following computations:*

$$\begin{aligned}
&\theta_{AB} = \Theta_{AB} = min(\rho(AB)|\{B\}) = \{00, 01\}, \\
&\theta_{CD} = \Theta_{CD} = min(\rho(CD)|\{C\}) = \{01, 10\}, \\
&\theta_{BC} = min(\rho(BC)|\{BC\}) \bowtie \Theta_{AB} \bowtie \Theta_{CD} = \\
&\{(ABCD) = 0001, 0010, 0101\}. \\
&\Theta_{BC} = min(\{0001, 0010, 0101\}) = \{0010, 0001\}.
\end{aligned}$$

We see that although the theory has 7 models, only 3 intermediate models were generated, each conditional minimal relative to one consistent tuple of BC.

min2(T)
Input: A pair-wise consistent join tree T_N, and a tree ordering $S_0, ..., S_r$.
Output: All minimal models of N, and all conditional minimal models, θ_i.

1. For $i = n$ to 1 traverse the tree bottom up. Let[4] $S_{i_1}, ..., S_{i_b}$ be the child nodes of S_i.

$$\theta_i = min(\rho(S_i)|F_i) \bowtie \bowtie_{i=1}^{b} \Theta_{i_j} \tag{5}$$

$$\Theta_i = min(\theta_i|I_i) \tag{6}$$

2. Output: Θ_0, the set of all minimal models.

Figure 6: Algorithm **min2**

Example 5.19 *Consider the following network having 5 variables $X_1, ..., X_5$, with the relations:*
$\rho(X_1X_3) = \{01, 10, 11\}$
$\rho(X_2X_3) = \{00, 01, 11\}$
$\rho(X_3X_5) = \{01, 10, 11\}$
$\rho(X_4X_5) = \{00, 01, 11\}$
The network is clearly acyclic and we consider the join-tree in which X_4X_5 is the parent of X_3X_5, and X_3X_5 is the parent of both X_1X_3 and X_2X_3. The tree is rooted at node X_4X_5. For the leaf nodes X_1X_3 and X_2X_3 we have that: (For abbreviation we use θ_{ij} for $\theta_{X_iX_j}$.)
$\theta_{13} = \Theta_{13} = min(\rho(X_1X_3)|\{X_3\}) = \{01, 00\}$
$\theta_{23} = \Theta_{23} = min(\rho(X_2X_3|\{X_3\}) = \{00, 01\}$
$\theta_{35} = min(\rho(X_3X_5)|\{X_3, X_5\}) \bowtie \Theta_{13} \bowtie \Theta_{23} =$
$\rho(X_3X_5) \bowtie \Theta_{13} \bowtie \Theta_{23} =$
$\{(X_1X_2X_3X_5) = \{0010, 1001, 0011\}\ \}.$
$\Theta_{35} = min(\theta_{35}|\{X_5\}) = \{(X_1X_2X_3X_5) = \{0010, 1001, 0011\}\ \}.$
$\theta_{45} = min(\rho(X_4X_5)|\{X_5\}) \bowtie \Theta_{35} = (X_1X_2X_3X_4X_5) = \{00100, 10001, 00101\}.$
$\Theta_{45} = min(\theta_{45}) = \{00100, 10001\}.$

We see that during computation we had at most 3 conditional minimal models associated with each node, while this network has totally 19 models.

Theorem 5.20 *Algorithm **min2** computes all and only the minimal models of its input theory.*

Proof: We have shown that the algorithm compute the conditional minimal models of the root relation. Since this is a superset of all the minimal models, the minimization operation at the root ensure that *all* minimal models be returned. □

We will bound the complexity of **min2** (without the tree-clustering preprocessing step). First note that for each node S_i there is a subset of tuples of that node whose conditional minimal models will definitely be part of the final set; those that are conditioned on minimal models within their own relation.

Consider now those conditional minimal models that are conditioned on non-minimal models in their own relations. Some of them will end up to be globally minimal while others will be pruned. Can we bound the number of these conditional minimal models that will be pruned? We conjecture that the size of the pruned set can be bounded as a polynomial function of the output. We summarize with the following theorem.

Theorem 5.21 *Let m be a bound on the number of conditional minimal models associated with any tuple in any node in the join-tree. Let s denotes the amount of space used by **min2**. Then, the time complexity of the algorithm is $O(s^2)$ and its space complexity is $s = O(n \cdot m \cdot logm)$.*

Proof: The time complexity of the algorithm can be bounded as follows. Before computing a relation θ_i, pair-wise consistency is applied implicitly between S_i and each child node, requiring at most $O(s \cdot logs)$ steps[5] (each relation size is bounded by s). Afterwards, the join operation is applied starting from the parent relation ρ_i and then joining with the Θ_{ij} of the child node. This order of the join operations guarantees an output linear performance. Thus, relation θ_i is computed in linear time in its input and output. The minimization operation applied when computing Θ_i (equation 5) can be implemented in $O(s^2)$ when s bounds the size θ_i.

The space complexity of the algorithm is determined by the sizes of the relations θ_i and Θ_i in each node. Since for each S_i $\theta_i \supseteq \Theta_i$, the space

[5]Pair-wise consistency can be enforced in $O(r \cdot k \cdot logk)$, when k bounds the size of each relation.

complexity is bounded by the space of θ_i's which is $O(n \cdot m)$. The parameter m bounds the size of θ_i, the set of all conditional minimal models of S_i. $\Box$.

Corollary 5.22 *If for every node S, the ratio between the number of minimal models, Θ_i, and the number of conditional minimal models θ_i is bounded by constant c, then the algorithm complexity is output polynomial.*

Unfortunately, we do not have a way of determining in advance when this condition will be satisfied. One possibility is to compute the number of solutions associated with each tuple (which can be done in linear time for trees) and use those numbers as bounds on the conditional minimal models.

We would like to argue at this point that the task of computing the conditional minimal models as a primary task, is important for its own right. When working in a distributed and dynamically changing environment, one wish to keep around all conditional minimal models. Adding just one relation to the join-tree or changing an existing one may make a complete set of conditional minimal models, that were not globally minimal before, globally minimal in the updated network.

6 Other related work

The idea of exploiting algorithms for Horn theories for doing inference was already suggested in [Lov91] where it was shown how SLD resolution for first-order Horn theories can be modified to be efficient for near-Horn theories. The virtue of our approach (relative to satisfiability solving) is that it identifies parameters of the theories by which the worst-case complexity can be bounded in advance. In [Cad92] there is a different partition of the set of propositional theories into classes for which the problem of finding one minimal model is tractable or NP-hard.

The properties of acyclic theories were also investigated in the past, primarily in relational databases [BFMY83] and in constraint networks [DP89]. It was shown that such theories are tractable for satisfiability and also for the task of finding models with *minimum number of positive literals* [FD94]. A new tractable class for finding one minimal model, based on certain dependencies between positive and negative literals in each clause, was recently introduced in [BEP94].

7 Conclusion

The task of finding all or some of the minimal models of a theory is at the heart of many knowledge representation systems. This paper presents several algorithms for this task and identifies new tractable classes. In particular, it presents new algorithms for finding minimal models of a propositional theory. The first group is effective for almost-Horn theories. In this group, we have presented algorithm MinSAT which is efficient for theories with only few non-Horn clauses, and algorithm VC-minSAT, which is efficient when clauses in the theory are almost Horn — that is, have very few positive literals. The second group of the algorithms is effective for theories that can be represented as trees of small-arity relations. Algorithm $min1$ is capable of generating a subset of the minimal models, while algorithm $min2$ generates all the minimal models.

Horn theories are used extensively in deductive databases and logic programs (for surveys, see [GM92, KH92]). In *disjunctive* deductive databases, we use rules of the form

$$A_1 \wedge ... \wedge A_n \longrightarrow B_1 \vee ... \vee B_m \tag{7}$$

where the A's and the B's are atoms in some first-order language. Disjunctive databases permits disjunctive information and nondeterministic choices in queries in a natural way (for examples, see [BEP94]). By most semantics it is agreed that the set of all minimal models of a disjunctive database of the form 7 above is the set of its intended models, and hence, for example, a clause is entailed by a deductive database if it is **true** in all the minimal models of the database [Min82]. Consequently, our almost Horn algorithms can be used for query answering in deductive databases. Specifically, algorithm MinSAT presented in Section 4.1, will be effective for deductive database having only few disjunctive rules. Indeed, it is likely that only a small fraction of the database will consist of disjunctive rules, since these rules are quite expressive and are saved for rare occasions (see also [RLS92]).

Acyclic networks and almost tree networks are likely to appear when the knowledge is relatively sparse or specially structured. Areas like model-based circuit diagnosis or knowledge-bases involving temporal information like planning and scheduling, are likely candidates. For instance, it was shown that a theory describing an n-bit adder can be represented by a chain-tree where each relation has arity at most 5. For more information see [FD94].

The algorithms outlined here and elsewhere provide the theoretical foundation for computing minimal models. The ultimate value of these algorithms should be evaluated empirically, on a set of real-world problems (e.g, in diagnosis or logic programming).

Appendix

A Tree-clustering

Constraint-based reasoning is a paradigm for formulating knowledge in terms of a set of constraints on some entities, without specifying methods for satisfying such constraints. Some techniques for testing the satisfiability of such constraints, and for finding a setting that will satisfy all the constraints specified, exploit the structure of the problem through the notion of a *constraint graph*.

The problem of the satisfiability of a propositional theory can be also formulated as a constraint satisfaction problem (CSP). For a propositional theory, the constraint graph associates a node with each propositional letter and connects any two nodes whose associated letters appear in the same propositional sentence.

Various parameters of constraints graph were shown as crucially related to the complexity of solving CSP and hence to solving the satisfiability problem. These include the *induced width*, w^*, the *size of the cycle-cutset*, the *depth of a depth-first-search spanning tree* of this graph, and the *size of the non-separable components* ([Fre85]),[DP88], [Dec90]). It can be shown that the worst-case complexity of deciding consistency is polynomially bounded by any one of these parameters. Since these parameters can be bounded easily by a simple processing of the graph, they can be used for assessing complexity ahead of time. For instance, when the constraint graph is a tree, satisfiability can be answered in linear time.

The tree-clustering scheme has a *tree-building* phase and a *query-processing* phase. The complexity of the former is exponentially dependent on the sparseness of the constraint graph, while the complexity of the latter is always linear in the size of the database generated by the tree-building preprocessing phase. Consequently, even when building the tree is computationally expensive, it may be justified when the size of the resulting tree is manageable and many queries on the same theory are expected. The algorithm is summarized in Figure 7. It uses the *triangulation* algorithm, which transforms any graph into a chordal[6] graph by adding edges to it [TY84]. The triangulation

[6]A graph is *chordal* if every cycle of length at least four has a chord.

Tree building(T, G)

input: A propositional theory T and its constraint graph G.

output: A tree representation of all the models of T.

1. Use the *triangulation algorithm* to generate a chordal constraint graph.
2. Identify all the *maximal cliques* in the graph. Let $C_1, ..., C_t$ be all such cliques indexed by the rank of their highest nodes.
3. Connect each C_i to an ancestor C_j $(j < i)$ with whom it shares the largest set of letters. The resulting graph is called a **join tree**.
4. Compute $\mathcal{M}_i$, the set of models over C_i that satisfy the set of all sentences from T composed only of letters in C_i.
5. For **each** C_i and for *each* C_j adjacent to C_i in the join tree, delete from $\mathcal{M}_i$ every model M that has no model in $\mathcal{M}_j$ that agrees with it on the set of their common letters (this amounts to performing *arc consistency* on the join tree). $\Box$

Figure 7: Propositional-tree-clustering:Tree-building phase

algorithm consists of two steps:

1. Select an ordering for the nodes (various heuristics for good orderings are available).
2. Fill in edges recursively between any two nonadjacent nodes that are connected via nodes higher up in the ordering.

Since the most costly operation within the *tree-building* algorithm is generating all the submodels of each clique (step 5), the time and space complexity of this preliminary phase is $O(|T| * n * 2^{|C|})$, where $|C|$ is the size of the largest clique, $|T|$ the size of the theory and n is the number of letters used in T. It can be shown that $|C| = w^* + 1$, where w^* is the *width* [7] of

[7] The *width* of a node in an ordered graph is the number of edges connecting it to nodes

the ordered chordal graph (also called *induced width*). As a result, for classes having a *bounded induced width*, this method is tractable.

Once the tree is built it always allows an efficient query-answering process, that is, the cost of answering many types of queries is linear in the size of the tree generated. The query-processing phase is described below (m bounds the number of submodels for each clique):

Propositional Tree-Clustering - Query Processing

1. T is satisfiable if none of its $\mathcal{M}_i$'s is empty, a property that can be checked in $O(n)$.

2. To see whether there is a model in which some letter P is true (false), we arbitrarily select a clique containing P and test whether one of its models satisfies (does not satisfy) P. This amounts to scanning a column in a table, and thus will be linear in m. To check whether a set of letters A is satisfied by some common model, we test whether all the letters belong to one cluster C_i. If so, we check whether there is a model in $\mathcal{M}_i$ that satisfies A. Otherwise, if the letters are scattered over several cliques, we temporarily eliminate from each such clique all models that disagree with A, and then re-apply arc consistency. A model satisfying A exists iff none of the resulting $\mathcal{M}_i$'s becomes empty. The complexity of this step is $O(n * m * logm)$. $\square$

lower in the ordering. The width of an ordering is the maximum width of nodes in that ordering, and the width of a graph is the minimal width of all its orderings

Acknowledgments

We thank Yousri El Fattah, Itay Meiri, and Judea Pearl for useful discussions and helpful comments on earlier drafts of this paper. We have benefited from discussions with Adam Grove and Daphne Koller. Thanks also to Michelle Bonnice for editing, and to one anonymous referee for detailed and helpful comments. In particular, the example in comment 4.8 is due to that referee. The first author also wishes to thank the computer science department at Tel-Aviv University, Israel for allowing her to use its resources while staying in Tel-Aviv.

References

[BEP94] Rachel Ben-Eliyahu and Luigi Palopoli. Reasoning with minimal models: Efficient algorithms and applications. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *KR-94: Proceedings of the fourth international conference on principles of knowledge representation and reasoning*, pages 39–50, San Francisco, CA, 1994. Morgan Kaufmann.

[BFMY83] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[BNNS91] C. Bell, A. Nerode, R.T. Ng, and V.S. Subrahmanian. Computation and implementation of non-monotonic deductive databases. Technical Report CS-TR-2801, University of Maryland, 1991.

[Cad92] Marco Cadoli. On the complexity of model finding for nonmonotonic propositional logics. In A. Marchetti Spaccamela, P. Mentrasti, and M. Venturini Zilli, editors, *Proceedings of the 4th Italian conference on theoretical computer science*, pages 125–139. World Scientific Publishing Co., October 1992.

[DD88] R. Dechter and A. Dechter. Beleif maintenance in dynamic constraint networks. In *AAAI-88: Proceedings of the 7th national conference on artificial intelligence*, pages 37–42, St. Paul, MN, USA, August 1988.

[DE92] Mukesh Dalal and David W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing Letters*, 44:173–180, 1992.

[Dec90] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[Dec92] Rina Dechter. Constraint networks. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 276–285. John Wiley, 2nd edition, 1992.

[DG84] William F. Dowling and Jean H. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[dKMR92] J. de Kleer, A.K. Mackworth, and R. Reiter. Characterizing diagnosis and systems. *Artificial Intelligence*, 56:197–222, 1992.

[Mac85] A.K. Mackworth, and E. Freuder. The complexity of some polynomilal network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[dKW87] J. de Kleer and B.C. Williams. Diagnosis multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[DP88] Rina Dechter and Judea Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[DP89] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[FD94] Y. El Fattah and R. Dechter. Diagnosing tree-decomposable circuits. Technical Report 94-18, University of California, Irvine, April 1994. A preliminary version in DX-92: Proceedings of the workshop on Principles of Diagnosis, October 1992.

[Fre85] E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4):755–761, 1985.

[Gav] F. Gavril. See page 134 in Michael R. Garey and David S. Johnson, “Computers and intractability, A guide to the theory of NP-completeness”, W. H. Freeman and Company, 1979.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GM92] John Grant and Jack Minker. Deductive database systems. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 320–328. John Wiley & Sons, 2nd edition, 1992.

[GS88] Giorgio Gallo and Maria Grazia Scutella. Polynomially solvable satisfiability problems. *Information Processing Letters*, 29:221–227, 1988.

[IM87] A. Itai and J. A. Makowsky. Unification as a complexity measure for logic programming. *Journal of Logic Programming*, 4:105–117, 1987.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, New York, 1972.

[KH92] R.A. Kowalski and C.J. Hogger. Logic programming. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 873–891. John Wiley & Sons, 2nd edition, 1992.

[Lif85] Vladimir Lifshitz. Computing circumscription. In *IJCAI-85: Proceedings of the international joint conference on AI*, pages 121–127, 1985.

[Lov91] Donald W. Loveland. Near-horn prolog and beyond. *Journal of Automated Reasoning*, 7:1–26, 1991.

[Mai83] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, MD, 1983.

[McC80] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McC86] John McCarthy. Application of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[Min82] Jack Minker. On indefinite databases and the closed world assumption. In *Proceedings of the 6th conference on automated deduction, Lecture Notes in Computer Science Vol. 138*, pages 292–308. Springer-Verlag, 1982.

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[RLS92] David W. Reed, Donald W. Loveland, and Bruce T. Smith. The near-horn approach to disjunctive logic programming. In *Proceedings of the 2nd workshop on extensions of logic programming, Lecture Notes in Artificial Intelligence Vol. 596*. Springer-Verlag, 1992.

[SES76] A. Itai A. S. Even and A. Shamir. On the complexity of timetable and multi-commodity flow. *SIAM journal of Computing*, 5:691–703, 1976.

[TY84] Robert E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.