Finding All Solutions if You can Find One

Rina Dechter[*]
Information and Computer Science
University of California, Irvine
dechter@ics.uci.edu

Alon Itai
Computer Science Department
Technion, Haifa, Israel
and
AT&T Bell Laboratories
itai@research.att.com

Septemeber, 1992

Abstract

We address the problem of enumerating (producing) all models of a given theory. We show that the enumeration task can be performed in time proportional to the product of the number of models and the effort needed to generate each model in isolation. In other words, the requirement of generating a new solution in each iteration does not in itself introduce substantial complexity. Consequently, it is possible to decide whether any tractably satisfiable formula has more than K solutions in time polynomial in the size of the formula and in K. In the special cases of Horn formulas and 2-CNFs, although counting is #P-complete, to decide whether the count exceeds K, is polynomial in K.

[*]This work was partially supported by NSF grant IRI-9157636 and by the Air Force Office of Scientific Research, AFOSR 900136.

1 Introduction

Finding all satisfying models for a formula in *conjunctive normal form (CNF)*, or even deciding whether a satisfying model exists (the *satisfiability* task), is known to be NP-hard. There are, however, special cases, such as 2-CNFs and *Horn formulas*, for which satisfiability is tractable, and a natural question is whether the task of finding all models is tractable as well.

Counting is important for several reasons. First, counting is often the most natural way of verifying equivalence between two theories. Second, it can provide degree of closeness between a theory and its approximation [10, 1]. Third, counting can provide heuristics for guiding planning and search, where we wish to estimate the probability that a given search avenue would lead to a goal. The number of solutions found in a simplified version of the problem description can then serve as an estimate of this probability [3].

In this paper we will explicate the relationship between the complexity of finding *one* model and finding *all* models. We will show that the time required for computing all models is proportional to the product of three factors: *the number of models*, *the time to find one model*, and *the number of literals* in the given theory. This result is then extended to *constraint satisfaction problems*, where theories are stated as conjunctions of relations on multivalued variables.

From the result above, we can immediately conclude that *counting* the number of models (or solutions) can also be accomplished in time proportional to the number of models. This result somewhat mitigates the negative finding of [8] stating that in many cases (e.g., Horn theories and 2-CNFs) counting is #P-complete. Whereas this negative result suggests that counting cannot be accomplished without enumeration, our result shows that enumeration itself is no harder then listing its output.

2 Preliminaries

We denote propositional symbols, also called *variables*, by uppercase letters $P, Q, R, X, Y, Z, ...$, propositional literals (i.e., $P, \neg P$) by lowercase letters $p, q, r, x, y, z, ...$, and disjunctions of literals, or *clauses*, by $\alpha, \beta, ...$. The complement operator $\sim$ over literals is defined as usual: If $p = \neg Q$, then $\sim p = Q$; If $p = Q$, then $\sim p = \neg Q$. Alternatively, we will allow the notation $P = 1$ (same as P) or $P = 0$ (same as $\neg P$). A *formula* in CNF is a set of clauses $\varphi = \{\alpha_1, ..., \alpha_t\}$, and it denotes their conjunction $(\alpha_1 \wedge, ..., \wedge \alpha_t)$. The *models* of a formula φ, $M(\varphi)$, (also called *solutions*) is the set of all satisfying truth assignments to all its symbols. A Horn formula is a CNF formula in which each clause has at most one positive literal. A k-CNF formula is a CNF formula in which clauses are all of length k or less.

A *constraint network* [11, 4], to be defined below, is a generalization of CNF formulas.

First, each variable X_i, instead of being either 0 or 1, may take a value from a finite domain D_i. Next, a *relation* over the variables $X_{i_1}, \ldots, X_{i_j}$ is a subset of the Cartesian product $D_{i_1} \times \ldots \times D_{i_j}$. (Each element of the relation is called a *tuple*.)

A constraint network is a set $\{R_1, \ldots, R_m\}$ of such relations, and its models or solutions are all tuples $(a_1, \ldots, a_n) \in D_1 \times \ldots \times D_n$ such that for every j, if R_j is over the variables $X_{j_1}, \ldots, X_{j_r}$, then $(a_{j_1}, \ldots, a_{j_r}) \in R_j$. We say that the network represents the unique relation $rel(N)$ over X, which is all its consistent assignments, (called *solutions*).

A constraint network can be associated with a *constraint graph* in which each node represents a variable and variables appearing in the same constraint are connected.

Each CNF formula is a constraint network with each $D_i = \{0, 1\}$, and each clause is the relation consisting of all the tuples for which at least one literal of the clause received the value 1. The solutions of the formula is the set of all satisfying truth assignments.

3 The Complexity of Finding All Solutions

Let $\varphi = \varphi(x_1, ..., x_n)$ be a CNF formula. We will show an algorithm, *find-all-solutions (φ)*, for enumerating the set of all φ's models. The running time $T_{all}(\varphi)$ of find-all-solutions(φ) is proportional to the number of models $|M(\varphi)|$, to the time for finding one model $T_1(\varphi)$, and to the number of literals n. *Find-all-solutions (φ)* computes all models of φ using a procedure for finding one model, *solve*(φ), as its basic subroutine. If φ is satisfiable, then *solve*(φ) returns a satisfying truth assignment; otherwise, it returns *false*. The notation *solve*$(\varphi, l_1, ..., l_i)$ is a shorthand for *solve*$(\varphi \cup \{l_1, ..., l_i\})$.

The algorithm *find-all-solutions*(φ), described below, uses a procedure, *find-next-sol(φ, $l_1, .., l_n$)*, that enumerates the solutions in a *dynamic lexicographic (DL)* ordering. According to this ordering, the first solution is chosen arbitrarily, and the solution generated following $l^i = (l_1, ..., l_n)$ has the largest *common prefix* with l^i relative to all the remaining solutions. In formal terms;

Definition 1: *(common-prefix)*
Let $s = (s_1, ..., s_n)$ and $t = (t_1, ..., t_n)$ be two n-tuples of values from a common domain. We say that *common-prefix*$(s, t) = p$ iff $s_i = t_i$,$\forall\ 1 \leq i \leq p$, and $s_{p+1} \neq t_{p+1}$.

Definition 2: *(DL ordering)*
Given a relation ρ on n variables, a *DL* ordering of the tuples in ρ is constructed as follows:

1. Select the first tuple t_1 arbitrarily.

2. Given that $t_1, t_2, ..., t_{i-1}$ were already selected, choose t_i such that

$$common\text{-}prefix(t_{i-1}, t_i) = \max_{j \in \rho - \{t_1, ..., t_{i-1}\}} (common\text{-}prefix(t_{i-1}, t_j)).$$

```
find-all-solutions ( $\varphi$ )
    1. for i=1 to n $mark_i = 0$. (Initialize marking)
    2. old-solution $\Longleftarrow$ $true$
    3. until old-solution $\neq$ $nil$ do
        • new-solution $\Longleftarrow$ find-next-sol( $\varphi$ , old-solution)
        • print new-solution
        • old-solution $\Longleftarrow$ new-solution
find-next-sol($\varphi$, $l_1, ..., l_n$)
    1. for $i = n$ to 1 do
        • if $mark_i = 0$ do
        • if solve($\varphi$ , $l_1, ..., l_{i-1}, \sim l_i$) $\neq$ $false$ then
        • $mark_i \Longleftarrow 1$ and $\forall\ j > i\ \ mark_j \Longleftarrow 0$. (Update marking), and
        • return solve($\varphi$ , $l_1, .., l_{i-1}, \sim l_i$)
        • else, $i = i - 1$
    2. end.
    3. return $nil$ (no next solution)
```

Figure 1: Algorithm *find-all-solutions*(φ) and algorithm *find-next-sol*($\varphi, l_1, ..., l_n$)

In order to ensure that each solution is produced only once, the algorithm uses a global marking vector of length n whose entries are 0,1. When a marking of entry i, $mark_i$, is 0, it indicates that for the current solution $l_1, l_2, ..., l_n$, the set of models beginning with $l_1, l_2, ..., l_{i-1} \sim l_i$ has not yet been generated, and 1, otherwise. We can view the vector $mark$ as an n-bit binary counter, forcing the solutions to be produced in lexicogrpahical order. To prove that the algorithm *find-all-solutions*(φ) in Figure 1 is correct, we need two more definitions.

Definition 3: *(literal-closure)*
Let φ be a CNF formula, and let $l_1, ..., l_t$ be a subset of literals in its language. The *literal-closure* of φ relative to $l_1, ..., l_t$, denoted $lc(\varphi, l_1, ..., l_t)$, is a CNF formula φ', obtained by repeatedly applying the following two operations to φ, for each literal l_i and for each clause $\alpha \in \varphi$:

1. If $l_i \in \alpha$, delete α from φ (the clause is already satisfied).

2. If $\sim l_i \in \alpha$, then eliminate $\sim l_i$ from α ($\sim l_i$ cannot satisfy α).

Lemma 1: *Let $\varphi' = lc(\varphi, l_1, ..., l_t)$. Then,*

1. *The models of φ' coincide with the models of $\varphi \cup \{l_1, ... l_t\}$ after projecting out the propositional symbols in $\{l_1, ..., l_t\}$.*

2. *The formula φ' is shorter than φ.*

3. *$lc(\varphi, l_1, ..., l_t)$ can be constructed in linear time.*

Proof: Clear. □

Definition 4: *(literal-closed)*
A class of CNF formulas, Φ, is *literal-closed* iff $\forall \varphi \in \Phi$ and $\forall \{l_1, ..., l_r\}$ in the language of φ, the formula $lc(\varphi, l_1, ..., l_r) \in \Phi$.

Theorem 1: *Let φ be a CNF formula in Φ that is literal-closed, and let $solve(\varphi)$ be a procedure for answering satisfiability of any formula in Φ. Then, algorithm* find-all-solutions (φ) *enumerates all the models of φ.*

Proof: The procedure $solve(\varphi, l_1, ..., l_i)$ can be executed by first constructing the formula $\varphi' = lc(\varphi, l_1, ..., l_t)$ and then applying $solve(\varphi')$. Since Φ is literal-closed, $solve(\varphi') \in \Phi$ is well defined. Since the algorithm generates the models in a DL ordering, it is guaranteed not to miss any model. □

Theorem 2: *Let Φ be a class of literal-closed CNF's whose satisfiability can be determined in $O(T(|\varphi|))$, when $|\varphi|$ is the size of φ. Then, the complexity of finding all models of any member φ of Φ is O($|M(\varphi)| \cdot T(|\varphi|) \cdot n$).*

Proof: There are at most n failures between consecutive solution generations (see step 2 of *find-next-sol*). Each such trial is a satisfiability task of the formula $\varphi \cup \{l_1, ..., l_i\}$. This can be accomplished by first constructing $\varphi' = lc(\varphi, l_1, ..., l_i)$ and then solving φ'. Since $\varphi' \in \Phi$ is shorter than φ, this step takes $O(T(|\varphi|) + |\varphi|)$. But, since satisfiability is linear at best, it amounts to $O(T(|\varphi|))$. Consequently, the total time for finding all solutions is O($|M(\varphi)| \cdot T(|\varphi|) \cdot n$). □

We can now apply Theorem 2 to classes of tractable formulas such as Horn formulas and 2-CNFs.

Corollary 1: *Let φ be either a Horn formula or a 2-CNF formula defined on n variables. Then, enumerating the models of φ takes $O(|\varphi| \cdot |M(\varphi)| \cdot n)$.*

Proof: Since Horn formulas and 2-CNFs are literal-closed, it follows from Theorem 2 and from the fact that satisfiability is linear for Horn formulas [7] and 2-CNFs [5], that the claim holds. □

We next generalize these results to constraint networks while avoiding some of the details. We will use n to denote the number of variables, k to bound the domain sizes, and c as the number of constraints.

Definition 5: *(instantiation-closed)*
A class of constraint networks C is *instantiation-closed* iff, $\forall\ N \in C$ and for every partial instantiation $\overline{x} = \{(X_1 = x_1), ..., (X_i = x_i)\}$, the network $N' = N \cup \{(X_j = x_j) | (X_j = x_j) \in \overline{x}\}$ can be transformed in linear time to a shorter equivalent network N'' that is also in C.

Theorem 3: *Let C be a class of* constraint networks*, such that for all $N \in C$, the consistency of N can be determined in time $T(|N|)$. If C is instantiation-closed, then the complexity of finding all solutions of any $N \in C$ is bounded by $O(|rel(N)| \cdot T(|N|) \cdot n(k-1))$.*

Proof: The DL ordering can be generalized for the multi-valued case by imposing an ordering on the domain of each variable. Then, algorithm *find-all-solutions* can be modified to generate all the solutions in this DL ordering. In this case the entries of the marking vector will contain viable candidate values that still need to be tried for each variable, relative to its past. The marking updating is essentially the same, except that if relation R_i is over the domain D_i, then $mark_i$ assumes the values $0..|D_i| - 1$, and instead of a binary counter we use a counter of mixed radix. The algorithm's complexity is O($|rel(N)| \cdot T(|N|) \cdot n(k-1)$). □

Corollary 2: *Let C be a class of instantiation-closed* constraint networks*, and K be a constant. If the consistency of $N \in C$ can be decided in polynomial time, then deciding whether N has at least K solutions is polynomial as well.* $\square$

Some classes of tractable constraint networks have tighter bounds for counting then those suggested by Theorem 3. For example, it is known that constraint networks whose constraint graphs have an r-vertex cycle-cutset[1] can decide consistency in $O(nk^{r+2})$ steps, while their solutions can be enumerated in $O(|rel(N)||N| + nk^{r+2})$. Similarly, constraint networks having an induced width[2] that is bounded by r can decide consistency in $O(nk^{r+1})$ steps, while their solutions can be enumerated in $O(|rel(N)||N| + nk^{r+1})$ [2, 4]. Theorem 3, however, yields bounds that are much higher. For instance, for networks having a cycle-cutset of size less or equal to r, Theorem 3 bounds the enumeration task by $O(n \cdot k^{r+2} \cdot |rel(N)| \cdot n(k-1))$. Likewise, the complexity of enumerating all solutions for constraint networks N having a bounded induced width r is $O(n \cdot k^{r+1} \cdot |rel(N)| \cdot n(k-1))$.

4 Conclusion

The main result of this paper is a method of enumerating all models of a given theory in time proportional to the product of the number of models and the effort needed to generate each model in isolation. This yields a polynomial time procedure to decide whether any tractably satisfiable formula has more then K solutions. Thus, in the special cases of Horn formulas and 2-CNFs, although counting is #P-complete, to decide whether the count exceeds K, is polynomial.

The significance of this result shows up in theory formation applications where it is required to find a Horn expression that approximates a stream of observations [1]. In this application counting enables us to ascertain whether the approximate theory describes the observations precisely.

[1]A cycle-cutset of a graph is a set of nodes that break all the graph's cycles.

[2]The width of a chordal graph is the size of its maximal clique. The induced width of an arbitrary graph is the minimal width among all the chordal graphs within which the input graph can be embedded.

References

[1] Dechter R. and Pearl J., "Structure identification in relational data". In *The Canadian Artificial Intelligence Conference*, May 1992, Vancouver, British Columbia

[2] Dechter R., "Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition". In *Artificial Intelligence*, 41(3), 1990, 273-312.

[3] Dechter R., "Network-Based heuristics for constraint satisfaction problems" *Artificial Intelligence* 34(1), 1987, 1-38.

[4] Dechter R., "Constraint Networks" In *Encyclopedia of Artificial Intelligence*, (2nd Ed.), 1992, Wiley and Sons, pp. 276-285.

[5] Even S., Itai A., and Shamir A., "On the Complexity of Timetable and Multi-Commodity Flow". *SIAM J. on Computing*, 5, 691-703, (1976).

[6] Garey M. R., and Johnson D. S., *Computers and Intractability – A guide to the theory of NP-completeness*, W.H. Freeman, 1978.

[7] Itai A., and Makowsky J. A., "Unification as a Complexity Measure for Logic Programming". *J. of Computational Logic* 4(2), 105-117, (1987).

[8] Valiant L. G., "The complexity of enumeration and reliability problems", *SIAM J. Comput*, 8(3), 1979, 410-421.

[9] Valiant L. G., "The complexity of computing the permanent", *Theoretical computer science*, 8, 1979, 181-201.

[10] Sellman B. and Kautz H., "Knowledge compilation using Horn approximation". In *Proceedings of AAAI-91*, Anaheim, CA, 1991.

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