ELSEVIER Artificial Intelligence 82 (1996) 273-302

Artificial Intelligence

On the hardness of approximate reasoning [*]

Dan Roth [*]

Aiken Computation Laboratory, Harvard University, 33 Oxford St., Cambridge, MA 02138, USA

Received April 1994; revised October 1994

Abstract

Many AI problems, when formalized, reduce to evaluating the probability that a propositional expression is true. In this paper we show that this problem is computationally intractable even in surprisingly restricted cases and even if we settle for an approximation to this probability.

We consider various methods used in approximate reasoning such as computing degree of belief and Bayesian belief networks, as well as reasoning techniques such as constraint satisfaction and knowledge compilation, that use approximation to avoid computational difficulties, and reduce them to model-counting problems over a propositional domain.

We prove that counting satisfying assignments of propositional languages is intractable even for Horn and monotone formulae, and even when the size of clauses and number of occurrences of the variables are extremely limited. This should be contrasted with the case of deductive reasoning, where Horn theories and theories with binary clauses are distinguished by the existence of linear time satisfiability algorithms. What is even more surprising is that, as we show, even approximating the number of satisfying assignments (i.e., "approximating" approximate reasoning), is intractable for most of these restricted theories.

We also identify some restricted classes of propositional formulae for which efficient algorithms for counting satisfying assignments can be given.

1. Introduction

Investigating the computational cost of tasks that are of interest to AI has been argued [22, 39] to be essential to our understanding and our ability to characterize these

[*] Supported by NSF grants CCR-89-02500 and CCR-92-00884 and by DARPA AFOSR-F4962-92-J-0466. A preliminary version of this paper appeared in the Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93), Chambery, France.

[*] E-mail: danr@das.harvard.edu.

tasks and to finding knowledge representation systems adequate for them. The problem discussed most extensively in this context is the problem of propositional satisfiability, the typical NP-hard problem, which is of special concern to AI because of its direct relationship to deductive reasoning. Many other forms of reasoning, including default reasoning, planning and others which make direct appeal to satisfiability, have also been shown to be NP-hard. In practice, there is a fundamental disagreement about the implications of this. There is no debate, however, that something has to be given up: restrict the form of the statements in the knowledge base, settle for approximate output and so on. One consequence of the intensive research in that direction is the identification of restricted languages for which propositional satisfiability can be solved efficiently (e.g., Horn theories).

In this paper we consider a related problem, that of counting satisfying assignments of propositional formulae. We argue that the role played by satisfiability problems in many AI problems in which *deduction* is of special concern, is replaced by that of counting satisfying assignments when *approximate reasoning* techniques are used. To support this argument we show that various methods used in approximate reasoning, such as computing degree of belief and Bayesian belief networks, are equivalent, computationally, to solving *counting* problems. We also show that considering the problem of counting solutions is a valuable approach in evaluating techniques that use approximations in an effort to avoid computational difficulties, such as constraint satisfaction and knowledge compilation,

We analyze the computational complexity of counting satisfying assignments of propositional languages, and prove that this is intractable even for Horn and monotone formulae, and even when the size of clauses and number of occurrences of a variable in the formula are extremely limited. This should be contrasted with the case of deductive reasoning, where Horn theories and theories with binary clauses are distinguished by the existence of linear time algorithms for their satisfiability. What is more surprising is that, as we show, even approximating the number of satisfying assignments (that is, "approximating" approximate reasoning), is intractable for most of those restricted theories. We identify some restricted classes of propositional formulae for which we develop efficient algorithms for counting satisfying assignments.

While our positive results can sometimes be used to find tractable languages for the approximate reasoning technique discussed, we believe that the main contribution of this paper is the presentation of these widely applicable and surprising hardness results. This implies that research should be directed away from investigating structural constraints of the "world" and towards investigating distributional constraints, and suggests that we reconsider how we model the reasoning problem. We discuss these issues further in Section 5.

In the next section we give background material from the computational complexity of counting problems, and in Section 3 we present our positive and negative results on exact and approximate counting of satisfying assignments. The main results are presented in Section 4, where we put the model-counting results in the context of various approximate reasoning techniques, by reducing those techniques to counting problems. Proofs of the technical results appear in the appendix.

2. The computational complexity of counting problems

We give in this section a brief overview of the computational complexity of counting problems and the related problems of approximate counting and random generation of solutions. For a detailed discussion consult [10,14,37,38].

With a large number of decision problems we can naturally associate a counting problem. For example, counting the number of satisfying assignments of a Boolean formula, counting the number of perfect matchings in a bipartite graph and counting the number of cycles in a graph. Clearly, the counting version is at least as hard as the decision problem but in many cases, even when the decision problem is easy, no computationally efficient method is known for counting the number of distinct solutions. The class #P was introduced by Valiant [37,38] in an effort to explain these phenomena. A counting problem belongs to #P if there is a non-deterministic algorithm such that for any instance I of the associated decision problem, the number of "guesses" that lead to acceptance of I is equal to the number of distinct solutions to I, and such that the algorithm is polynomial in the size of I.[1] As usual, the hardest problems in the complexity class are *complete* in the class.

In particular, it was shown that counting the number of satisfying assignments of a CNF formula as well as the counting versions of many other NP-complete problems are complete for #P. More significantly, it was also shown that the counting versions of many problems in P are also complete for the same class. Examples of the latter include counting the number of satisfying assignments of a DNF formula, counting the number of cycles in a graph and many other problems [29,37,38].

Problems that are #P-complete are at least as hard as NP-complete problems, but probably much harder. Evidence to the hardness of problems in #P is supplied by a result of [36] which implies that one call to a #P oracle suffices to solve any problem in the polynomial hierarchy in deterministic polynomial time. This may serve also as indication that #P is outside of the polynomial hierarchy. It is therefore natural to consider the problem of approximate counting. The notion of approximation we use is that of *relative approximation* [14,15,35]. Let M, M' be two positive numbers and $\delta \geqslant 0$. We say that *M' approximates M* within δ when

$$M'/(1+\delta) \leqslant M \leqslant M'(1+\delta).$$

Indeed, approximating a solution to a #P problem might be easier than finding an exact solution. In fact, it is no harder than solving NP-hard problems [35]. For example, there exists a polynomial time randomized algorithm that approximates the number of satisfying assignments of a DNF formula within any constant ratio [14,15]. It is possible, though, for a #P-complete problem, even if its underlying decision problem is easy, to resist even an efficient *approximate solution*. An example for that was given in [14], and in this paper we exhibit a similar phenomenon. We prove, for various propositional languages for which solving satisfiability is easy, that it is NP-hard to approximate the number of satisfying assignments even in a very weak sense.

[1] In [37] the definition is given in terms of "counting Turing machines".

We use the notion of relative approximation to discuss probabilities as well. It is worth noticing therefore that this notion of approximation is preserved when computing ratios of quantities. Assume we can approximate M_1 and M_2 to within δ. That is, we can find M_1', M_2' such that $M_1'/(1+\delta) \leqslant M_1 \leqslant (1+\delta)M_1'$ and $M_2'/(1+\delta) \leqslant M_2 \leqslant (1+\delta)M_2'$. Then,

$$\frac{M_1'}{M_2'}\frac{1}{(1+\delta)^2} \leqslant \frac{M_1}{M_2} \leqslant (1+\delta)^2\frac{M_1'}{M_2'}.$$

Thus, this yields a relative approximation of the ratio M_1/M_2 as well (within $2\delta+\delta^2$). In particular, when computing the conditional probability $P(Y=y \mid X=x)$, of the event $Y=y$ given evidence $X=x$, since

$$P(Y=y \mid X=x) = \frac{P(Y=y, X=x)}{P(X=x)}$$

we conclude that:

Proposition 2.1. *The complexity of computing relative approximation of the conditional probability $P(Y=y \mid X=x)$ is polynomially related*[2] *to that of computing relative approximation of the unconditional probability $P(Y=y)$.*

We note that a related class of problems of interest to AI, that of randomly generating solutions from a uniform distribution, was shown in [14] to be equivalent to randomized approximate counting, for a wide class of problems. (All natural problems considered here, e.g., finding satisfying assignments of Boolean formulae and various graph problems are in this class.) It is therefore enough, from the computational complexity point of view, to consider the problems of exact and approximate counting, as we do here.

3. Summary of model-counting results

In this section we summarize our results on exact and approximate counting of satisfying assignments of propositional languages. Those include hardness results for exact and approximate counting and positive results for exact counting. Complete proofs of the results are given in the appendix.

Let $\#(SAT, \mathcal{L})$ ($\tilde{\#}(SAT, \mathcal{L})$) denote the problem of counting (approximating, respectively) the number of satisfying assignments of a given formula from the propositional language $\mathcal{L}$. Given the problem $\#(SAT, \mathcal{L})$, a problem hierarchy is obtained whenever we place additional restrictions or relaxations on the allowed instances. Given propositional languages $\mathcal{L}_1$ and $\mathcal{L}_2$, define $\mathcal{L}_1 \subseteq \mathcal{L}_2$ if every instance of $\mathcal{L}_1$ is also an instance of $\mathcal{L}_2$. (e.g., HORN $\subseteq$ CNF.) Clearly, if we can solve the problem $\#(SAT, \mathcal{L}_2)$ we can solve the problem $\#(SAT, \mathcal{L}_1)$; to prove hardness results it is therefore enough to

[2] By that we mean that a procedure that relatively approximates unconditional probabilities and is, say, polynomial in δ, yields a procedure that can relatively approximate conditional probabilities, and is also polynomial in δ (and of course, vice versa).

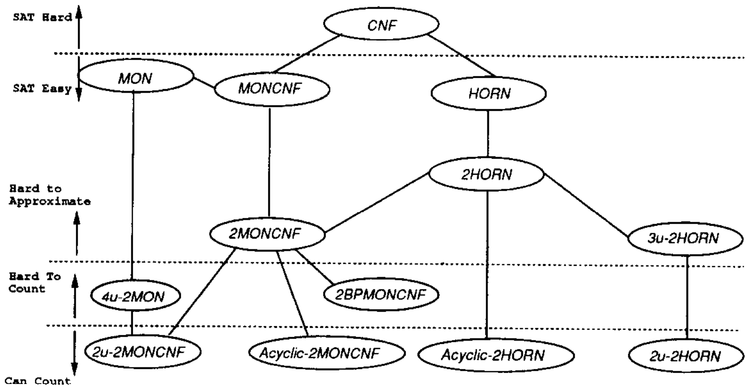

Fig. 1. Complexity of (approximately) counting satisfying assignments.

consider the most restricted languages. The same argument holds for the corresponding approximation problem.

We use the following notations and conventions in denoting propositional languages: if LANG is a class of Boolean formulae and k, l are integers, then kLANG denotes the subclass of formulae in LANG in which a clause consists of up to k literals; $l\mu$-LANG denotes the class of all LANG formulae in which no variable occurs more than l times. l is the *degree* of the formulae. For example, 2MONCNF consists of monotone CNF formulae with clauses of length 2; 3μ-2HORN consists of Horn formulae with clauses of length 2 in which no variable occurs more than 3 times.

- SAT: Any Boolean formulae.
- MON: Boolean formulae in which all variables are unnegated (monotone formulae).
- CNF: Boolean formulae in Conjunctive Normal Form.
- MONCNF: Monotone CNF.
- HORN: A CNF in which clauses have up to one unnegated variable (Horn clauses).
- 2BPMONCNF: A 2MONCNF in which the set of variables can be divided into two sets, and every clause contains one variable from each.
- Acyclic-2MONCNF: Given a 2MONCNF theory ϕ, let G be an undirected graph containing a vertex for every variable in ϕ and an edge connecting two vertices if and only if the corresponding variables appear in the same clause. ϕ is Acyclic-2MONCNF if this corresponding graph is acyclic.
- Acyclic-2HORN: Given a 2HORN theory ϕ, let G be a directed graph containing a vertex for every literal in ϕ and an edge from every vertex corresponding to a literal in the body of a rule (i.e., negative variable in the clause representation of the rule) to the vertex corresponding to a literal in the head of a rule (i.e., positive variable in the clause representation of the rule). Two special nodes T and F are added for clauses with empty body or empty head. ϕ is Acyclic-2HORN if every connected component of this corresponding graph is a directed tree.

Fig. 1 summarizes our results; it presents a hierarchy of propositional languages along with a classification according to the complexity of $\#(SAT, \mathcal{L})$ and $\tilde{\#}(SAT, \mathcal{L})$.

Based on the above comment these results imply similar results on other, less restricted languages.[3]

It is noticeable that for various propositional languages having efficient algorithms for satisfiability, and even for very restricted versions of these (e.g., 3μ-2HORN), exact counting is complete for #P. In fact, for the case of Horn theories, the situation is fully understood, and we give an efficient algorithm for the only possible case, 2μ-2HORN. The situation for approximate counting is even more surprising: for very restricted classes of Horn theories (e.g., 3μ-2HORN) it is NP-hard to approximate the number of satisfying assignments even within, say, $2^{n^{9/10}}$ (for formulae over n variables). Similar results hold for 2MONCNF theories, for which the bounded degree case is open. Our positive results virtually complete the complexity picture and can be directly applied in some of the reasoning techniques considered.

3.1. Statements of results

We now formally state the technical results outlined above. We state the results only for some of the important languages; results for other languages can be easily deduced by inclusion, as described above. Proofs are given in the appendix.

Theorem 3.1 (Hardness of Exact Counting). *Let $\Sigma \in \mathcal{L}$ be a propositional formula on n variables. If $\mathcal{L}$ is one of the following propositional languages, counting the number of satisfying assignments of Σ is complete for* #P:

(1) $\mathcal{L}$ = 2MONCNF [38],
(2) $\mathcal{L}$ = 2BPMONCNF [29],
(3) $\mathcal{L}$ = 2HORN,
(4) $\mathcal{L}$ = 3μ-2HORN,
(5) $\mathcal{L}$ = 4μ-2MON.

Theorem 3.2 (Hardness of Approximation). *Let $\Sigma \in \mathcal{L}$ be a propositional formula on n variables, and let $\varepsilon > 0$ be any constant. If $\mathcal{L}$ is one of the following propositional languages, approximating the number of satisfying assignments of Σ to within $2^{n^{1-\varepsilon}}$ is NP-hard*:

(1) $\mathcal{L}$ = 2MONCNF,
(2) $\mathcal{L}$ = 3μ-2HORN.

Theorem 3.3 (Positive results). *Let $\Sigma \in \mathcal{L}$ be a propositional formula on n variables. If $\mathcal{L}$ is one of the following propositional languages, then there exists an efficient algorithm for counting the number of satisfying assignments of Σ.*

(1) $\mathcal{L}$ = 2μ-2MONCNF,
(2) $\mathcal{L}$ = 2μ-2HORN,

[3] Notice that we place the language 2HORN above 2MONCNF even though 2HORN does not contain 2MONCNF. 2HORN contains, however, 2anti-MONCNF (where all the variables in each formula are negated rather than unnegated) and thus, clearly, all the counting results that hold for 2MONCNF hold also for 2anti-MONCNF.

(3) $\mathcal{L}$ = Acyclic-2MONCNF,
(4) $\mathcal{L}$ = Acyclic-2HORN.

4. Reducing approximate reasoning to counting

In this section we consider various techniques for approximate reasoning and show that in each case inference is equivalent to solving a counting problem. We start by considering the case of computing degree of belief, the underpinning of approximate reasoning. We then consider Bayesian belief networks, reasoning with approximate theories and constraint satisfaction problems. Finally, we discuss some previous work that relates approximate reasoning to counting problems, for which our results here also apply.

4.1. Degree of belief

The inference of a degree of belief is a generalization of deductive inference which can be used when the knowledge base is augmented by, e.g., statistical information, or in an effort to avoid the computationally hard task of deductive inference.

Consider a knowledge base consisting of a propositional theory Σ and assume we would like to assign a *degree of belief* to a particular statement α. This situation is natural in various AI problems such as planning, expert systems and others, where the actions an agent takes may depend crucially on this degree of belief. In [24] it is suggested that the kind of reasoning used in expert systems is the following: "we are given a knowledge base of facts (possibly, with their associated probabilities); we want to compute the probability of some sentence of interest. ... According to *probabilistic logic*, the probability of a sentence is the sum of the probabilities of the sets of possible worlds in which that sentence is true ...".

Indeed, the general approach to computing degree of belief is that of assigning equal degree of belief to all basic "situations" consistent with the knowledge base, and computing the fraction of those which are consistent with the query. Much work has been done on how to apply this principle, and how to determine what are the basic situations [1,2,4].

We consider here the question of *computing* the degree of belief in a restricted case, in which the knowledge base consists of a propositional theory and contains no statistical information. The hardness results we get in this restricted case just highlight the computational difficulties in the more general cases.[4] Using the above approach, all possible models of the theory are given equal weight and we are interested in the computational complexity of computing the degree of belief of a propositional formula, that is, the fraction of models that are consistent with a propositional query.

[4] The first-order version of this problem was considered in [12] where it was shown that almost all problems one might want to ask are highly undecidable. In some cases, though, it was shown that the asymptotic conditional probabilities exist, and can be computed.

Given a propositional theory Σ on n variables, the *probability that Σ is satisfied, P_Σ*, is computed over the uniform distribution over the set $\{0,1\}^n$ of all possible assignments of Σ.

$$P_\Sigma = Prob\{\Sigma \equiv T\} = |\mathcal{M}(\Sigma)|/2^n,$$

where $\mathcal{M}(\Sigma)$ denotes the set of all satisfying assignments of Σ, and $|\mathcal{M}(\Sigma)|$ denotes its size, and T stands for the truth value.

Given a propositional theory Σ and a propositional statement α, the *degree of belief in α*, is the *conditional probability* of α with respect to Σ, $P_{\alpha|\Sigma}$, that is, the fraction of satisfying assignments of Σ that satisfy α:

$$P_{\alpha|\Sigma} = Prob\{\alpha \wedge \Sigma \equiv T \mid \Sigma \equiv T\} = \frac{|\mathcal{M}(\alpha \wedge \Sigma)|}{|\mathcal{M}(\Sigma)|}.$$

The observation that $|\mathcal{M}(\alpha)| = P_{\alpha|p\vee\bar{p}}$ for any variable p, together with the discussion in Section 2 (Proposition 2.1) implies:

Proposition 4.1. *The complexity of computing (approximating) the degree of belief in a propositional statement with respect to a propositional theory, is polynomially related to the complexity of computing (approximating) the number of models of a propositional statement.*

Together with the results in Theorem 3.1 and Theorem 3.2 we have:

Theorem 4.2. *The problem of computing the degree of belief in a propositional statement (over n variables) with respect to a propositional theory, is complete for* #P.[5] *For every fixed $\varepsilon > 0$, approximating this probability within $2^{n^{1-\varepsilon}}$ is NP-hard.*

4.2. Bayesian belief networks

Bayesian belief networks provide a natural method for representing probabilistic dependencies among a set of variables and are considered an efficient and expressive language for representing knowledge in many domains [13]. We consider here the class of *multiple connected belief network*, i.e., networks that contain at least one pair of nodes (variables) that have more than one undirected path connecting them. It has been argued that the expressiveness of these networks is required for representing knowledge in several domains, like medicine. We first present briefly a general class of belief networks and the associated inference problem and then show how to reduce the inference problem to that of counting satisfying assignments of a propositional formula. For definitions and an elaborate discussion of Bayesian belief networks, the expressiveness of this representation, and the type of inference one can utilize using it, see [26].

[5] Strictly speaking the problem of computing the degree of belief is not in #P, but easily seen to be equivalent to a problem in this class. We keep the same loose interpretation in the rest of the paper.

A Bayesian belief network (causal network) consists of a graphical structure augmented by a set of probabilities. The graphical structure is a directed acyclic graph (DAG) in which nodes represent random variables (domain variables) and edges represent the existence of direct causal influence between the linked variables. A conditional probability is associated with the group of edges directed toward each node (and not with each single edge). Prior probabilities are assigned to source nodes (i.e., any node without any incoming edge). We represent the belief network as a triple (V, E, P), where V is the set of vertices (variables), E is the set of edges and P is the set of probabilities. In particular, P consists of prior probability functions, $P(X_i = x_i)$, for every source node X_i and conditional probabilities functions, $\{P(X_i \mid Y_{ij})\}_{Y_{ij} \in p_{X_i}}$, for each node X_i with a set p_{X_i} of direct predecessors. (We use the notation $P(X_i = x_i)$ and $P(x_i)$ inadvertently, when it is clear that we refer to the variable X_i.) See the construction in the proof of Theorem 4.3 for an example of a belief network. In general, not every probability distribution can be represented by a Bayesian belief network. However, given a DAG it is easy to specify consistently the conditional probabilities. One needs only to make sure that the conditional probabilities attached satisfy, for every node X_i, $\sum_{x_i} P(X_i = x_i \mid p_{X_i}) = 1$.

For complexity analysis, we take our complexity parameter to be n, the number of nodes in the belief network. Notice that the conditional probabilities tables associated with the network might be exponential in n. Practitioners of Bayesian belief networks try to avoid this case, of course. In our reduction the conditional probabilities tables will have concise representations, polynomial in the number of nodes in the network, and in this sense one can think of our complexity measure as if it is the total size of network, including the conditional probabilities tables.

The general *inference problem* using belief network is that of calculating the posterior probability $P(S_1 \mid S_2)$, where S_1 (S_2) is either a single instantiated variable or a conjunction of instantiated variables. The most restricted form of probabilistic inference, determining $P(Y = T)$ for some propositional variable Y (with no explicit conditioning information), was analyzed by Cooper [5] who proved that it is NP-hard. This hardness result for the exact inference problem shows that one cannot expect to develop general-purpose algorithms for probabilistic inference that have a polynomial running time and therefore there is a need to divert attention toward trying to construct *approximation algorithms* for probabilistic inference. Our results show that this is not the case:

Theorem 4.3. *The problem of computing the probability that a node in a Bayesian belief network is true is complete for* #P. *Moreover, for every fixed* $\varepsilon > 0$, *approximating this probability within* $2^{n^{1-\varepsilon}}$ *(where* n *is the size of the network) is NP-hard.*

Proof. The proof is based on the reduction from [5]. The two major differences are that (1) we reduce the problem of *counting* satisfying assignments of a propositional formula to that of computing the probability that a node in a Bayesian belief network is true, and (2) based on the results from Section 3 we can start our reduction from a restricted propositional formula, yielding a more restricted network topology.

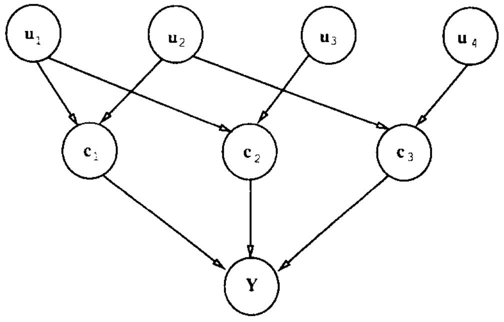

Fig. 2. The belief network generated from the 2MONCNF formula Σ.

In the following we reduce the problem of counting satisfying assignments of a 2MONCNF[6] formula to that of computing the probability that a node in a Bayesian belief network is true. Since our reduction preserves the number of satisfying assignments this reduction holds for the problem of approximating the probability as well.

Consider an instance of 2MONCNF, $\Sigma = \{c_1, c_2, \dots, c_m\}$ where c_i are clauses on a set $U = \{u_1, u_2, \dots, u_n\}$ of n Boolean variables. We construct a belief network $BN = (V, E, P)$ containing variable Y such that

$$2^n P(Y = T) = |\mathcal{M}(\Sigma)|.$$

To construct $BN = (V, E, P)$ we show how to define the vertices V, the edges E and the set of prior and conditional probabilities P. V is defined to be a set of $n + m + 1$ vertices, one for each variable u_i, one for every clause c_j and one for Y. The set of edges E consists of up to $3m$ edges: a variable u_i is connected to all clauses c_j in which it appears (i.e., total of up to $2m$ edges, since $\Sigma \in$ 2MONCNF); Y is connected to all clauses c_j. Fig. 2 depicts the BN generated using the above procedure for the instance of 2MONCNF in which $U = \{u_1, u_2, u_3, u_4\}$, and

$$\Sigma = \{(u_1 \vee u_2), (u_1 \vee u_3), (u_2 \vee u_4)\}.$$

The set of probabilities P is defined as follows: Each of the source nodes u_i, $1 \leqslant i \leqslant n$, is given a prior probability of $1/2$ to be T. For incoming edges to the node corresponding to the clause c_j we define the conditional probability such that the node becomes T only when it is satisfied by the assignment to the variables in the clause. Formally, if $c_j = u_{j1} \vee u_{j2}$ $(1 \leqslant j \leqslant m)$, define the conditional probabilities by:

$$\begin{aligned} &P(c_j = T \mid u_j^1 = v_1, u_j^2 = v_2) \\ &\quad = \begin{cases} 1, & \text{if the assignment } u_j^1 = v_1,\ u_j^2 = v_2 \text{ satisfies } c_j, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

[6] This is not possible in [5], since the results there hinge on the hardness of solving satisfiability, which can be done in polynomial time for 2MONCNF.

Finally, the conditional probability for the edges coming into the node Y is defined by

$$P(Y=T \mid c_1, c_2, \ldots, c_m) = \begin{cases} 1, & \text{if } c_1 = T,\ c_2 = T, \ldots, c_m = T, \\ 0, & \text{otherwise.} \end{cases}$$

It is easy to see that the structure (V, E, P) defined is indeed a Bayesian belief network. Also, it is clear that the construction of a belief network from a given 2MONCNF instance can be accomplished in time that is polynomial in the size of the 2MONCNF instance.[7]

We now compute the probability $P(Y=T)$. Let $u = (u_1, \ldots, u_n)$ be an assignment of the n input variables (that is, $u \in \{0,1\}^n$), and $c = (c_1, \ldots, c_m)$ be an assignment of the m clauses (that is, $c \in \{0,1\}^m$).

By the construction above we then have that

$$P(Y=T) = \sum_{s=0}^{2^n-1} \sum_{t=0}^{2^m-1} P(Y=T \mid c=t) P(c=t \mid u=s) P(u=s). \tag{1}$$

Suppose Σ is satisfiable, and let $s_1, s_2, \ldots, s_k$ be the satisfying assignments. Clearly, for $i = 1, \ldots, k$, $P(u = s_i) = 1/2^n$. Also, by the definition of the conditional probability for $i = 1, \ldots, k$, we have that $P(c = c(s_i) \mid u = s_i) = P(Y = T \mid c = c(s_i)) = 1$, and for any other assignment, these terms are equal to 0. Thus, the internal sum in Eq. (1) is equal to $1/2^n$ when s is a satisfying assignment of Σ, and is equal to 0 otherwise. We get,

$$P(Y=T) = \frac{|\mathcal{M}(\Sigma)|}{2^n}.$$

Applying now the results in Theorem 3.1 and Theorem 3.2 completes the proof. □

We have considered the computational complexity of computing the probability of a node in a Bayesian belief network being true. To compute a conditional probability, that is, $P(Y=y \mid X=x)$, where X, Y might be sets of nodes in the network, we notice that

$$P(Y=y \mid X=x) = \frac{P(Y=y, X=x)}{P(X=x)}.$$

It is clear that exact computation of the conditional probability is as hard as computing the unconditional probability (taking, e.g., X to be a single source node). Based on Proposition 2.1 this is the case also for the problem of approximating the conditional probability, and therefore we can conclude:

Theorem 4.4. *The problem of computing the conditional probability of a node given evidence in a Bayesian belief network, is complete for* #P. *Moreover, for every fixed* $\varepsilon > 0$, *approximating this probability within* $2^{n^{1-\varepsilon}}$ (*where* n *is the size of the network*) *is NP-hard.*

[7] This relies on the fact that we can define the conditional probabilities concisely. In general, every node can be associated with a conditional probability table that is exponential in the size of the network.

Finally we note that, as in [5], this reduction can be modified to hold for restricted network topology (limited in-degree, out-degree, etc.). Further restrictions to the topologies of the network can be utilized if we reduce problems of counting satisfying assignments of syntactically restricted CNF formulae to that of computing the probability that a node in the network is true. In light of the results in Section 3, this can yield even stronger hardness results. Recently, Dagum and Luby [6] presented an even stronger result, implying the hardness of computing an *absolute* approximation of probabilities in Bayesian networks. The results here are different in that we show that the inference is equivalent to counting, and combined with the results in Section 3, it implies hardness results even for restricted network topologies.

4.3. Reasoning with approximate theories

The theory of reasoning with *approximate theories* was introduced by Selman and Kautz in a series of papers [17, 18, 32] as a new approach to developing efficient knowledge representation systems.

The goal is to speed up inference by replacing the original theory by two theories that belong to a different propositional language $\mathcal{L}$ and approximate the original theory. One approximate theory implies the original theory (a lower bound) and the other one is implied by it (an upper bound). While reasoning with the approximations instead of the original theory does not guarantee exact reasoning, it can sometimes provide a quick (but not necessarily complete, see below) answer to the inference problem. This can happen when $\mathcal{L}$ allows for efficient deduction, e.g., if $\mathcal{L}$ is the class of propositional Horn formulae.[8] Of course, computing the approximations is a hard computational problem, and this is why it is suggested as an "off-line" *compilation process*. Some computational aspects of computing theory approximations and reasoning with them are studied also in [3, 11, 21]. In the following we concentrate on discussing Horn approximation.

For notational convenience, when no confusion can arise, we identify in this section the propositional theory Σ with the set of its models (satisfying assignments). Observe that the connective "implies" ($\models$) used between Boolean functions (propositional formulae) is equivalent to the connective "subset or equal" ($\subseteq$) used for subsets of models. That is, $\Sigma_1 \models \Sigma_2$ if and only if $\Sigma_1 \subseteq \Sigma_2$.

Consider a propositional theory Σ. The Horn theories Σ_{lb}, Σ_{ub} are a Horn lower bound and Horn upper bound, respectively, of Σ, if and only if

$$\Sigma_{lb} \models \Sigma \models \Sigma_{ub}$$

or, equivalently, in subset notations,

$$\Sigma_{lb} \subseteq \Sigma \subseteq \Sigma_{ub}.$$

Σ_{glb} and Σ_{lub}, the greatest Horn lower bound and least Horn upper bound, respectively, of Σ, are called *Horn approximations* of the original theory Σ.

[8] The implication problem for Horn theories can be solved in linear time in the combined length of the theory and the query. This remains true for even a broader class of queries such as DNF formulae where each disjunct contains at most one negative literal and arbitrary CNF formulae.

In order to answer $\Sigma \models \alpha$, we use a Horn approximation based inference procedure in the following way: (1) test if $\Sigma_{lub} \models \alpha$. If the answer to (1) is "yes", then the inference procedure answers "yes", $\Sigma \models \alpha$. Otherwise, (2) test if $\Sigma_{glb} \models \alpha$. If the answer to (2) is "no", then the inference procedure answers "no", $\Sigma \not\models \alpha$. Otherwise, the inference procedure returns "don't know".

Aside from the two computational problems related to Horn approximations, namely, computing the approximations and the question of the size of the formula representing the approximations (see e.g., [3, 17, 18, 21, 32]) a third major question, that is harder to analyze, is the question of evaluating the utility of reasoning with the approximate theories. Clearly, if for a given query we have either $\Sigma_{lub} \models \alpha$ or $\Sigma_{glb} \not\models \alpha$, the answer to the question $\Sigma \models \alpha$ is correct. The total performance of the inference procedure is determined, though, by how many queries are answered "don't know", forcing the procedure to resort to the original inference problem in order to answer the query.

Consider a theory Σ, and let Σ_{lub} be its least upper bound approximation.

Proposition 4.5. *The number of queries for which the reasoning with approximate theories returns "don't know" is at least exponential in* $|\Sigma_{lub} \setminus \Sigma|$.

Proof. Let $S = \Sigma_{lub} \setminus \Sigma$. For every subset $s \subset S$ define the query $\alpha_s = \Sigma \cup s$ (that is, the set of models of α consists of the models of Σ and the models in the set s). Then, for all $s \subset S$, $\Sigma_{glb} \models \alpha_s$ (since $\Sigma_{glb} \models \Sigma$), and $\Sigma_{lub} \not\models \alpha_s$. Therefore, for all the $2^{|S|} - 1$ queries α_s, reasoning with approximate theories returns "don't know". □

In [16] it is shown that, for a family of propositional languages $\mathcal{L}$ which consists of kHorn formulae (all Horn formulae with up to k literals in a clause), one can construct examples of theories Σ for which $|\Sigma_{lub} \setminus \Sigma|$ is exponential in the number of variables, where Σ_{lub} is the least upper bound of Σ in $\mathcal{L}$. (Surprisingly, one can even construct $(k+1)$Horn theories with these properties.) Using Proposition 4.5, this leads to a double exponential number of queries for which reasoning with approximate theories returns "don't know". In [21] tools are developed that allow for a construction of such examples for every language $\mathcal{L}$ with respect to which we want to consider theory approximation. We briefly describe one example, for the case of Horn approximation:

Consider the theory

$$\Sigma = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{n-1} \vee x_n).$$

It is easy to see that the number of models of Σ is $3^{n/2}$. However, the least upper approximation of Σ with respect to Horn, Σ_{lub}, can be shown to contain all the models in $\{0,1\}^n$, that is, it is of size 2^n. This can be argued from the fact that the set of models of any Horn formula is closed under intersection (bitwise "and") (see, e.g., [9]). Therefore, the size of $\Sigma_{lub} \setminus \Sigma$ is exponential in the number n of variables.

This question is partially addressed in [11], where learning techniques are used to find a locally-optimal approximation. However, in [11], as is done in general in the theory of Horn approximations, an approximation is defined in terms of *containment*, (that is, logical strength), and there is no guarantee that this approximation is "close" to the optimal one, nor that the optimal one approximates the original theory within any reasonable bound, in the sense that it answers some fraction of the queries correctly.

Taking the "counting" approach, as we suggest in this paper, can shed some light on this problem. As argued above, a more reasonable way to estimate the utility of reasoning with approximate theories is to define it in terms of proximity in the number of models, since this correlates positively with the number of queries answered correctly (i.e., not answered with "don't know") by the approximations. The examples presented indicate that, when no restriction are imposed on the queries, even syntactically similar propositional languages (e.g., $(k+1)$Horn formulae and kHorn formulae) can be far enough in terms of the number of models to produce unacceptable behavior.

The argument presented above, as exhibited in Proposition 4.5, shows that the heart of the problem is the fact that the queries presented to the reasoner are unrestricted. Thus it motivates an investigation in the direction of reasoning with restricted queries, where it might be possible to avoid these difficulties. Indeed, in [19] an experimental analysis is presented in which, under severe restrictions on the classes of queries allowed, reasoning with approximate theories is shown to succeed on a large percentage of the queries. In [21] a general analysis is developed and it is shown, in particular, that reasoning with the least upper bound of a theory Σ with respect to $\mathcal{L}$ is always correct if and only if the queries belong to the language $\mathcal{L}$.

4.4. Constraint satisfaction problems

Constraint satisfaction problems (CSP) provide a convenient way of expressing declarative knowledge, by focusing on local relationships among entities in the domain.

A *constraint satisfaction problem* [7] involves a set of n variables $x_1, \ldots, x_n$ having domains $D_1, \ldots, D_n$, where each D_i defines the set of values that variable x_i may assume. An *n-ary relation* on these variables is a subset of the Cartesian product $D_1 \times D_2 \times \cdots \times D_n$. A *constraint* between variables is a subset of the Cartesian product of their domains. The set of all n-tuples satisfying all the constraints are the solutions to the CSP. The problem is either to find all the solutions, or one solution.

In general, constraint satisfaction problems are hard, as a generalization of satisfiability of CNF formulae. A CNF formula is a constraint satisfaction problem in which $D_i = \{0, 1\}$ for each i, and each clause is the relation consisting of all the tuples for which at least one literal is 1. The set of solutions of the CSP is the set of satisfying assignments of the formula. In particular, if we consider a network of binary constraints[9] over $D_i = \{0, 1\}$, as is usually done, the problem can be represented as a 2SAT formula.

Finding all the solutions is clearly an enumeration problem, and based on the results in Section 3, even the associated counting problem is #P-complete for almost all nontrivial

[9] Not every n-ary relation can be represented by a network of binary constraints with n variables [23].

cases. [10]

However, even finding one solution of a constraint satisfaction problem is known to be hard in general, as discussed above, and different heuristics techniques have been used to find approximate solutions. To discuss those under the counting point of view presented here, we first observe that the above discussion implies that computing (approximating) the number of solutions of a (binary) CSP problem is at least as hard as computing (approximating) the number of solutions of a 2MONCNF formula.

Together with the results in Theorem 3.1 and Theorem 3.2 we have:

Theorem 4.6. *The problem of computing the number of solutions of a (binary) CSP problem is complete for #P. For every fixed $\varepsilon > 0$, approximating this number within $2^{n^{1-\varepsilon}}$ is NP-hard.*

Search techniques were traditionally used to solve CSPs. These techniques require, in the worst case, exponential search time, and analyzing those techniques in order to get some performance guarantees is usually hard.

We exemplify how the counting point of view taken here can be used to evaluate one class of heuristics [8] and restrict its feasibility. Dechter and Pearl suggest to use counting to guide the search according to an estimate of the confidence we have that a specific solution can be extended further to a full solution.

More specifically, it is suggested to (1) simplify the pending subproblems (i.e., make some simplifying assumptions about the continuing portion of the search graph), (2) count the number of consistent solutions in each simplified subproblem, and (3) order the candidates according to these counts and use them to decide among the pending subproblems.

The intuition behind the heuristics is that "the number of solutions found in the simplified version of the problem is a good estimate to the number of solutions in the original problem and thus is indicative of the chance to retain at least one surviving solution."

The results we present in Section 3 therefore restrict the utility of these heuristics in two ways: First, the simplified subproblem must be a tree, or another syntactically constraint structure (see Theorem 3.3), in order to be able to get a count of the solutions of the simplified version. More significantly, in case the original problem possesses a non-trivial structure, the number of solutions of the simplified version is not indicative at all to the number of solutions of the original problem. If it were, this could be used to approximate the number of solutions of the original problem, which we have shown to be hard. To summarize:

Corollary 4.7. *Using counting heuristics to constraint satisfaction problem is computationally intractable.*

[10] We comment, though, that Valiant's results ([38], Fact 7) imply that under simple conditions (e.g., when finding one solution is easy and the problem satisfies a form of self-reducibility), enumerating the solutions is polynomial in their number even when the counting problem is hard. These conditions trivially hold for Horn formulae, and therefore for subclasses of CSP as well.

On the other hand, the positive results (Theorem 3.3) can be used to identify restricted domains for which these counting techniques can be shown useful.

4.5. Related results

The most related result to the work presented here is a reduction of yet another approximate reasoning technique to a counting problem. Orponen [25] shows, by reduction, that the problem of computing Dempster's rule of combination, the main tool in the Dempster-Shafer theory of evidence is at least as hard as the problem of computing the number of satisfying assignments of a propositional CNF formula.[11] Those results can be now strengthened using Theorem 3.1 and Theorem 3.2. As immediate results we get that (1) even the approximate version of Dempster's rule of combination is hard to compute and (2) the hardness result for the Dempster-Shafer theory still holds even if we severely restrict the *basic probability assignments* allowed as evidence in the Dempster-Shafer theory.

5. Discussion

We have put results on the complexity of counting and approximating the number of satisfying assignments of propositional formulae in the context of various approximate reasoning techniques. The significance of this approach was illustrated by showing that various, supposedly different methods in approximate reasoning are equivalent to counting. It is worth noticing, for example, that while there is an active discussion in the approximate reasoning community as for differences in the semantical basis of the Dempster-Shafer theory and the Bayesian approach (see, e.g., [27,31,33]) we show here that there is one computational problem underlying both approaches: computing inference is equivalent to counting satisfying assignments of a theory. Moreover, we have shown that this approach is valuable in evaluating techniques that use approximations in an effort to avoid computational difficulties. This was exemplified by analyzing heuristics used in constraint satisfaction problems and the utility of reasoning with approximate theories. We believe that the approach developed here can be found useful in the analysis of other problems of interest to AI.

Our hardness results indicate that for the problems of computing degree of belief, probabilistic reasoning and other approximate reasoning techniques, one cannot expect to develop general-purpose algorithms that have a polynomial running time. Moreover, even computing approximate inference was proved to be intractable.

These results do not rule out the possibility of developing efficient algorithms that apply in restricted cases, as our positive results suggest; identifying more positive results and investigating how they apply to various techniques might be one direction to extend this work. We mention in particular the problem of (approximately) counting the number of satisfying assignments in bounded degree 2MONCNF formulae. The problem is left

[11] A similar result, using a different reduction, was proved independently by Provan [30]. We thank Greg Provan for bringing [25,30] to our attention.

open (see Fig. 1) and its solution might be used to develop efficient algorithms for constraint satisfaction problems, for example. The positive results presented here are important therefore not only for pointing out the tractability frontiers, but also since they provide a collection of techniques that can be used to further enhance our understanding of these problems and develop new results, possibly, for other problems of interest to AI.

However, the extent to which the hardness results apply (very restricted propositional languages, restricted topologies of Bayesian networks, etc.), imply that research should be directed away from investigating structural constraints on the "world" and towards investigating distributional constraints, or limiting our reasoning tasks rather than the "world" we reason about. The first might include constraining the distributions we can represent in our belief networks (e.g., [28]), while the second could imply studying restrictions on the type of queries we need to respond to. This is motivated also by the results in Section 4.3 that suggest that a possible approach to allow for efficient reasoning might be to constrain the *queries* (rather than the "world"). Indeed, partly motivated by these results, in [21] it is shown how constraining the queries gets around the difficulties presented in Section 4.3 and leads to correct reasoning with approximate theories.

A possible interpretation of the surprising and widely applicable results presented here is that we need to reconsider the way we model the reasoning problem. One way to get around the difficulties presented here is to allow the reasoner other ways to access the "world", instead, or in addition to the fixed (formula-based, Bayesian network-based, etc.) knowledge-based approach that we analyze here. Promising results in this direction are presented in [20].

Appendix A. Proofs

In this section we formally state and prove the results outlined in Section 3. The results are stated only for some of the important languages in Fig. 1 and results for other languages can be easily deduced by inclusion, as described in Section 3. The results in this section are summarized in three theorems: in Section A.2 (Theorem 3.1) we prove results on the hardness of exact counting. In Section A.3 (Theorem 3.2) we prove results on the hardness of approximate counting. In Section A.4 (Theorem 3.3) we give positive results on exact counting by describing efficient algorithms for counting satisfying assignments for formulae in those languages.

A.1. Preliminaries

For a Boolean function f we denote by $\mathcal{M}(f)$ the set of satisfying assignments of f and by $|\mathcal{M}(f)|$ its size. We denote $\{1,2,\dots,n\} = [n]$.

Lemma A.1. *The problem of counting the number of satisfying assignments of a* 2MONCNF *is equivalent to the problem of counting the number of independent sets in a graph.*

Proof. Given a graph $G(V, E)$ we associate with it a monotone CNF Φ_G on variables $\{x_1, \ldots, x_n\}$ as follows:

$$\Phi_G = \bigwedge_{(v_i,v_j)\in E} (x_i \vee x_j).$$

The inverse mapping is defined in the same manner: given a formula $\Phi \in$ 2MONCNF on $\{x_1, \ldots, x_n\}$, construct a graph $G_\Phi(V, E)$ on $\{v_1, \ldots, v_n\}$ as follows:

$$(v_i, v_j) \in E \quad \text{iff } (x_i \vee x_j) \text{ is a clause in } \Phi.$$

Now, let $S_I = \{v_j : j \in I\}$ be an independent set in G, then the assignment defined by

$$x_i = \begin{cases} 0, & \text{if } i \in I, \\ 1, & \text{otherwise,} \end{cases}$$

satisfies Φ_G. The reason is that in every clause $x_i \vee x_j$ at least one of the variables is assigned to 1, since otherwise, by the definition of Φ_G, $(v_i, v_j) \in E$, but both v_i and v_j are in the independent set S_I.

For the other direction, assume that x is a satisfying assignment of Φ, and let $I = \{i \in [n] : x_i = 0\}$, then $S_I = \{v_j : j \in I\}$ is an independent set in G_Φ. The reason is that by the definition of the graph G_Φ, no two vertices which share an edge are in I, since otherwise we have a clause in Φ that x does not satisfy. □

Corollary A.2. *The problem of counting the number of satisfying assignments of a* $k\mu$-2MONCNF *is equivalent to the problem of counting the number of independent sets in a graph of degree k.*

Proof. The mapping defined in Lemma A.1 maps graphs of degree k to $k\mu$ formulae. □

A.2. Exact counting

Theorem 3.1 (Hardness of Exact Counting). *Let $\Sigma \in \mathcal{L}$ be a propositional formula on n variables. If $\mathcal{L}$ is one of the following propositional languages, counting the number of satisfying assignments of Σ is complete for* #P:
(1) $\mathcal{L}$ = 2MONCNF,
(2) $\mathcal{L}$ = 2BPMONCNF,
(3) $\mathcal{L}$ = 2HORN,
(4) $\mathcal{L}$ = 3μ-2HORN,
(5) $\mathcal{L}$ = 4μ-2MON.

Proof. (1) and (2) are well known: (1) is proved in [38]; (2) is from [29]. We can get (3) from (2) by negating all the variables in one of the bipartite sets.

To prove (4), given a formula Φ in 2HORN we rewrite it, without changing the number of solutions, as a 3μ-2HORN formula. Let Φ be a 2HORN formula on $\{x_1, \ldots, x_n\}$ and assume x_i appears $m(i)$ times in Φ (negated or unnegated). For every $i \in [n]$

introduce $m(i)$ new variables, $\{x_i^{(j)}\}_{j=1,m(i)}$ and replace the jth appearance of x_i in Φ by $x_i^{(j)}$ to get Σ, a μ-2HORN formula. We then conjoin Σ with $\bigwedge_i \Gamma_i$, where Γ_i is the following 2μ-2HORN formula:

$$\Gamma_i = x_i^{(1)} \rightarrow x_i^{(2)} \rightarrow \cdots \rightarrow x_i^{(m-1)} \rightarrow x_i^{(m)} \rightarrow x_i^{(1)}.$$

Clearly

$$\Phi \equiv \Sigma \wedge \bigwedge_i \Gamma_i.$$

Thus, the number of satisfying assignments of the 3μ-2HORN formula $\Sigma \wedge \bigwedge_i \Gamma_i$ is equal to the number of satisfying assignments of the original 2HORN formula, and the counting problems for those languages are therefore equivalent.

For (5) we use a different rewriting technique. Given a 3μ-2CNF formula Φ we rewrite it, while preserving the number of satisfying assignments as $\Sigma \wedge \neg\Gamma$ where Σ and Γ are both monotone, and Σ and $\Sigma \wedge \Gamma$ are both in 4μ-2MON. Since $|\Sigma \wedge \neg\Gamma| = |\Sigma| - |\Sigma \wedge \Gamma|$, the hardness of exact counting for 4μ-2MON formulae results from the hardness of counting for 3μ-2CNF formulae (cf. (4)).

In rewriting Φ, given a variable x_i, which appears both negated and unnegated in Φ, we replace its (up to 2) unnegated occurrences by y_i and its (up to 2) negated occurrences by z_i. The resulting formula is a 3μ-2MONCNF formula Φ'. To force that $\forall i$, $\overline{y_i} = z_i$ we denote

$$\Phi'' = \bigwedge_i (y_i \vee z_i), \qquad \Gamma = \bigvee_i (y_i \wedge z_i).$$

It is clear that

$$\Phi \equiv \Phi' \wedge \Phi'' \wedge \neg\Gamma = \Sigma \wedge \neg\Gamma.$$

Since $\Sigma = \Phi \wedge \Phi''$ is a 3μ-2MONCNF formula and Γ is a 1μ-2MON formula (in a DNF form), the result follows. □

A.3. Approximate counting

Theorem 3.2 (Hardness of Approximation). *Let $\Sigma \in \mathcal{L}$ be a propositional formula on n variables. If $\mathcal{L}$ is one of the following propositional languages, approximating the number of satisfying assignments of Σ to within a factor of $2^{n^{1-\varepsilon}}$, for any fixed ε, is NP-hard.*

(1) $\mathcal{L} =$ 2MONCNF,
(2) $\mathcal{L} = 3\mu$-2HORN.

Proof. To get (2) from (1) we use the rewriting technique for nonmonotone clauses as in (4) of Theorem 3.1. (This technique leaves the number of solutions the same but might increase the number of variables up to n^2. This can easily be handled as we do below.) Notice that the rewriting technique used in (5) of that theorem does not extend for approximations.

The next lemma provides the main step in the proof of (1). The proof is based on the "blow-up" technique developed in [14]. A different version of this lemma appears also in [34].

Lemma A.3. *For any ε, approximating the number of independent sets of a graph on n vertices within $2^{n^{1-\varepsilon}}$ is NP-hard.*

Proof. We use the "blow-up" technique introduced in [14], to reduce the problem of approximating the number of independent sets in G to the k-INDEPENDENT-SET problem [10]. Given $G(V,E)$, where $|V| = n$, we construct a graph $G'(V',E')$ such that approximating the number independent sets in G' to within $2^{n^{1-\varepsilon}}$ can be used to solve k-INDEPENDENT-SET in G. G' is defined as follows: each vertex $v \in V$ is blown-up to a "cloud" $c(v)$ of m vertices in G'. If $(u,v) \in E$, in G' we construct a complete bipartite graph on $c(v) \cup c(u)$); otherwise, there are no edges connecting $c(v)$ to $c(u)$). Formally,

$$V' = \{v^j : v \in V;\ j \in \{1,\dots,m\}\},$$

$$E' = \{(v^i,u^j) : (v,u) \in E;\ i,j \in \{1,\dots,m\}\}.$$

Assume now that G contains an independent set I of size k. Then, $I' = \{v^j : v \in I;\ j \in \{1,\dots,m\}\}$ is an independent set of size km in G'. Since all the subsets of an independent set are also independent sets, there are are least $N_{\min} = 2^{km}$ independent sets in G'.

On the other hand, if G contains no independent set of size k, an independent set in G', contains vertices from up to $k-1$ "clouds", since otherwise, the corresponding vertices in G (the "projection" of the clouds) generate in G an independent set of size larger than $k-1$. In particular, the largest independent set in G' is of size $\leqslant (k-1)m$ (there might be, however, many different independent sets of that size). Thus, there are no more than $N_{\max} = \binom{n}{k-1}2^{(k-1)m}$ independent sets in G'.

Finally, let $k = n/2$; in this case, k-INDEPENDENT-SET is NP-hard [10, p. 194]. Given $\varepsilon > 0$, choose r large enough such that $1-(r-2)/(r+1) < \varepsilon$, and let $m = n^r$. The "blow-up" graph G' is of size $|V'| = nm = n^{r+1}$. We have that,

$$\begin{aligned}\sqrt{N_{\min}/N_{\max}} &= \sqrt{\frac{2^{km}}{\binom{n}{k-1}2^{(k-1)m}}} = \sqrt{\frac{2^m}{\binom{n}{k-1}}} = \sqrt{\frac{2^{n^r}}{\binom{n}{n/2-1}}} \\ &\geqslant \sqrt{\frac{2^{n^r}}{2^{n/2}}} = 2^{(2n^r-n)/4} \geqslant 2^{n^{r-2}} = 2^{|V'|^{(r-2)/(r+1)}}.\end{aligned}$$

Notice also that this "blow-up" procedure is polynomial in the size of the original graph. Therefore, if we can approximate the number of independent sets in G' within $2^{|V'|^{1-\varepsilon}} < 2^{|V'|^{(r-2)/(r+1)}}$ we can use this approximation to decide whether the graph G' has more than $N_{\min}$ or less than $N_{\max}$ independent sets. As argued above, this leads to deciding $n/2$-INDEPENDENT SET. □

We have proved in Lemma A.1 that counting the number of satisfying assignments of a 2MONCNF is equivalent to the problem of counting the number of independent sets in a graph. This, together with Lemma A.3 implies the theorem. □

A.4. Positive results

Theorem 3.3 (Positive results). *Let $\Sigma \in \mathcal{L}$ be a propositional formula on n variables. If $\mathcal{L}$ is one of the following propositional languages, there exists an efficient algorithm for counting the number of satisfying assignments of Σ.*

(1) $\mathcal{L} = 2\mu$-2MONCNF,
(2) $\mathcal{L} = 2\mu$-2CNF,
(3) $\mathcal{L} =$ Acyclic-2MONCNF,
(4) $\mathcal{L} =$ Acyclic-2HORN.

Proof. We prove (1) by developing a closed form formula that is easy to evaluate for the number of independent sets in graphs of degree 2. For the other cases we develop efficient algorithms. (3) is the problem of counting independent sets of trees, for which we give an efficient recursive algorithm. The algorithms for (2) and (4) are more elaborate. In both cases we start by constructing chains of the form $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_r$, from the original theory. We then show that the original theory can be represented as a composition of these chains, and develop compositions rules that allow us to count the number of satisfying assignments of the composite chains. The difference between (2) and (4) is the type of compositions allowed. We note that (1) and (3) are subcases of (2) and (4), respectively, but we give for them separate proofs, since those cases are considerably simpler.

Proof of (1). Based on Corollary A.2 it is enough to count the number of independent sets of a degree-2 graph. In the following we consider the empty set to be an independent set.

Let G be a connected graph of maximal degree 2 on n vertices. G can be either a *cycle*, in case all its vertices are of degree 2, or an *arm* if exactly 2 of its vertices are of degree 1. We have:

Lemma A.4. *Let IS^c_n denote the number of independent sets in a cycle of length n, and IS^a_n the number of independent sets in an arm of length n. Then,*

(i) $IS^a_n = 1 + \sum_{j=1}^{\lceil n/2 \rceil} \binom{n-j+1}{j}$,
(ii) $IS^c_n = 1 + \sum_{j=1}^{\lfloor n/2 \rfloor} \binom{n-j}{j} + \binom{n-j-1}{j-1}$.

Proof. We denote by $IS^a_{n,j}$ (respectively, $IS^c_{n,j}$) the number of independent sets of size j in an arm (respectively, cycle) of length n. $IS^a_{n,j}(v)$ counts those independent sets that contain a fixed vertex, v, in the cycle.

(i) The problem of computing $IS^a_{n,j}$ reduces to the following combinatorial problem: find the number of selections of j integers from the set $\{1, \ldots, n\}$, such that no two consecutive numbers are selected. To count this number, consider any selection of j different numbers from among $\{1,\ldots,n-j+1\}$. The mapping which adds 0 to the first number selected, 1 to the second, $\ldots, j-1$ to the jth, is a 1-1 correspondence between those selections and the legal selections we count. Thus, we get:

$$IS^a_{n,j} = \binom{n-j+1}{j}.$$

To get the total number we sum on j, $j \leqslant \lceil n/2 \rceil$, and add 1 for the empty set.

(ii) We claim that $IS^c_{n,j} = IS^a_{n-1,j} + IS^a_{n-1,j}(v)$. To see that, consider the cycle as an arm with end points v_1, v_n. (I.e, v_1, v_n are adjacent in the cycle.) The first term corresponds to the independent sets of the cycle that do not contain, say, v_n, while the second term correspond to those that contain it. (The latter group cannot contain v_1, so we get them by shifting each set that contains v_1 by one place). Also, $IS^a_{n-1,j}(v) = IS^a_{n-3,j-1}$ as we can just add two adjacent vertices as prefix, one that is not selected and v. We get:

$$IS^c_{n,j} = \binom{n-j}{j} + \binom{n-j-1}{j-1}.$$

To get the total number we sum on j, $j \leqslant \lfloor n/2 \rfloor$, and add 1 for the empty set. □

As an immediate consequence of Lemma A.4 we get:

Lemma A.5. *Let $G(V,E)$ be a graph of maximal degree 2, and assume G has r connected components, of sizes $n_1,\ldots,n_r$, respectively. The number of independent sets in G is*

$$\prod_{i=1}^{r} IS^x_{n_i},$$

where $x \in \{a,c\}$ depends on whether the component is an arm or a cycle.

Proof of (2). All the clauses in a 2μ-2CNF theory are of the form $l_1 \rightarrow l_2$ where every literal l_i might be a variable $x \in X_n$ or its negation, and every *variable* appears no more than twice in the given theory. (I.e., either a literal appears twice and its negation never appears in the theory or that the literal and its negation appear once each.) Notice that in the implication representations every clause $\overline{l_1} \vee l_2$ has two equivalent representation, $l_1 \rightarrow l_2$ and $\overline{l_2} \rightarrow \overline{l_1}$. Since it will be convenient to use the implication representation we assume that we hold both representations and use the one that it more convenient. If a literal l appears both as an antecedent and as a consequent in two clauses, e.g., $l_1 \rightarrow l_2$ and $l_2 \rightarrow l_3$ we can combine then the chain $l_1 \rightarrow l_2 \rightarrow l_3$, which now contains the only occurrence of $var(l_2)$ in the theory.

We call the theory $C = l_1 \rightarrow l_2 \rightarrow \cdots \rightarrow l_r$ a *simple chain*. In this case, l_1 is the *antecedent* of C and l_r its *consequent*. The antecedent and the consequent are the only literals of degree 1 in the theory; all other literals are of degree 2. The antecedent and the consequent literals might have the same underlying variable, but this cannot be the case for other literals in the chain. We say that a simple chain $l_1 \longrightarrow l_2 \longrightarrow \cdots \longrightarrow l_r$ in a 2μ-2CNF theory is maximal if it cannot be extended, that is, the theory contains no other clause with l_1 as consequent and no other clause with l_r as antecedent. Notice that two maximal simple chains C_1 and C_2 in a theory cannot have as consequents l_1 and $\overline{l_1}$, respectively. The reason in that in this case we can "reverse" C_2 and negate all its literals, to get an equivalent chain that has l_1 as antecedent. The new chain can be concatenated to C_1, contradicting its maximality. A simple chain is called a *cycle* if $var(x_1) = var(x_r)$; in this case all variables are of degree 2.

Let $C_1, C_2, \ldots, C_k$ be maximal simple chains in a 2μ-2CNF theory Σ. Assume C_1 and C_2 both have l_1 as antecedent. In this case we can *compose* the simple chains, and say that $C_1 \wedge C_2$ is a *composite chain*. Similarly, C_1 and C_2 can be composed if they have a common consequent. Notice that if a literal l appears in two maximal simple chains, it must appear in it both as an antecedent or as a consequent (being internal to both contradicts the degree requirement while being a consequent in one and an antecedent in the other contradicts the maximality of the chain). Thus, we can repeat this process of composition until there are no two chains in Σ, simple or composite, that share a *variable*.

If two chains share both antecedents and consequents, composing them results in a *closed* composite chain. Every composite chain C that is not closed has exactly two literals of degree 1, say l_i and l_j. We decide arbitrarily to denote $l_i = t(C)$, the *tail* of C and $l_j = h(C)$, the *head* of C if $i < j$. The tail and head of a composite chain can be both antecedents, both consequents or any other combination.

Given a chain C_1 on variables $\{x_1, x_2, \ldots, x_k\}$ (i.e., all k variables appear in C_1), denote by N_{C_1} the number of assignments of $\{x_1, x_2, \ldots, x_k\}$ that satisfy C_1. Likewise, for $b \in \{0, 1\}$ denote by $N_{C_1|h=b}, N_{C_1|t=b}$ the number of assignments of $\{x_1, x_2, \ldots, x_k\}$ that satisfy C_1, given that we force the head (respectively, tail) literal to 0 or 1, and by $N_{C_1|t=b_1,h=b_1}$ when we force both head and tail to some value in $\{0, 1\}$.

Notice that given the values N_C, $N_{C|t=1}$, $(N_{C|h=1})$ and $N_{C|t=1,h=1}$ one can determine all the possible values $N_{C|t=*,h=*}$. Therefore, it will be necessary to compute all these values for a composite chain. For a closed chain, it will be enough to compute N_C, since the chain will not be composed any more.

Given a 2μ-2CNF theory, to count the number of its satisfying assignments we first decompose it to simple chains and cycles. The number of satisfying assignments of these simple theories is given in Lemma A.7. As argued above, if a variable x is common to two chains it must be an antecedent in both or a consequent in both. We derive, in Lemma A.8 and Lemma A.9 a composition rule that shows how to compute the number of satisfying assignments of the conjunction of two theories, under the restriction that these two theories are part of a 2μ-2CNF theory. This composition rule is applied also for conjuncting composite chains, until there are no more compositions to be made. At this point the theory is represented as a conjunction of disjoint theories, and we use Lemma A.6 to compute its total number of satisfying assignments. We now describe the algorithm in some more details, and then prove some lemmas that show its correctness.

A.4.1. Algorithm: Count-2μ-2HORN

Let $\Sigma = (l_{i_1} \vee l_{j_1}) \wedge \cdots \wedge (l_{i_m} \vee l_{j_m})$ be a CNF theory in 2μ-2CNF such that $var(l_{i_k}), var(l_{j_k}) \in X_n$. The following procedure counts the number of satisfying assignments of Σ:

Construct simple chains:

- Represent each clause[12] as

$$l_1 \vee l_2 \equiv \overline{l_1} \rightarrow l_2 \equiv \overline{l_2} \rightarrow l_1.$$

- Fix an order of the clauses. Start from the first clause and greedily combine $l_i \rightarrow l_j$ and $l_j \rightarrow l_k$ to $l_i \rightarrow l_j \rightarrow l_k$. (That is, for each clause, check if one of its representations can be combined in that way, if any.) Go on until you end up with a maximal simple chain.
- Starting from the next available clause, repeat the above procedure, using only clauses that are not already part of a previously constructed chains. Go on until no more combination can be made. (I.e, no variable occurs both as consequent and as an antecedent.) Make sure in this process that the constants T and F are never internal to a chain.
- For each simple chain of the form

$$C = l_{i_1} \rightarrow l_{i_2} \rightarrow \cdots \rightarrow l_{i_r},$$

compute, using Lemma A.7 the value of $N_C, N_{C|t=1}, N_{C|h=1}, N_{C|t=1,h=1}$.

Notice that in the above process no more than n combination steps are required (since the degree of a variable in Σ is at most 2, and that the resulting chains are uniquely defined.

Combine chains:

- Given Σ, represented as a conjunction of simple chains, combine chains C_1 and C_2 if they have a common variable as antecedent, as consequent, or both.
- At each combination step, resulting in a composite chain C, compute, using Lemma A.8 and Lemma A.9, the values of $N_C, N_{C|t=1}, N_{C|h=1}, N_{C|t=1,h=1}$.
- Go on until there are no chains with common variables. No more than n combinations steps are required, since the degree of a variable in Σ is at most 2. The process results in disjoint composite chains.

Compute the number of satisfying assignments:

- Let $\{C_1, C_2, \ldots, C_k\}$ be the set of disjoint composite chains given by the previous stage. Let N_{C_i} be the number of assignments that satisfy C_i, counted only over the

[12] Clauses of length 1 can also be represented in this way, e.g., $x \equiv (T \rightarrow x)$ and $\bar{x} \equiv (x \rightarrow F)$. The fact that the constants T and F might appear in the theory more than twice will not affect the correctness of the algorithm (see remark on that later).

variables in C_i. If only r variables from X_n are used in $\{C_1, C_2, \ldots, C_k\}$ Then, using Lemma A.6 the number of satisfying assignments of Σ is

$$|\mathcal{M}(\Sigma)| = 2^{n-r} \prod_{j=i}^{k} N_{C_j}.$$

A.4.2. Correctness

It is clear that the maximal simple chains constructed by the algorithm are unique, and so are the composite chains. It is also clear that the construction is efficient. We just need to show how to derive the number of satisfying assignments in that process. We show that in the next lemmas.

Lemma A.6. *Let $\Sigma = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ be a formula on X_n such that for each i, x_i appears in exactly one of the conjuncts C_j. Let N_{C_i} be the number of independent sets of C_i and N_Σ the number of independent sets of Σ. Then,*

$$N_\Sigma = \prod_{i=1}^{k} N_{C_i}.$$

Proof. Clear from the fact that C_i are variable-disjoint. □

The initial step in computing the number of satisfying assignments is given by the next lemma:

Lemma A.7. *Consider the simple chain*

$$C = l_1 \to l_2 \to \cdots \to l_r.$$

(i) *If all the underlying variables in C are different then $N_C = r + 1$, $N_{C|t=1} = 1$, $N_{C|h=1} = r$, $N_{C|t=1,h=1} = 1$.*
(ii) *If $l_1 = l_r$ then $N_C = 2$.*
(iii) *If $l_1 = \overline{l_r}$ then $N_C = 1$.*

Proof. For (i) we note that if $l_i = 1$ then $l_j = 1$ for $j > i$, so we need to consider only the first index i such that $l_i = 1$. There are r possibilities for that and one satisfying assignment in which all variables are 0. The other statements follow similarly (here $l_1 = t(C); l_r = h(C)$). In (ii), either all literals are 1 or all are 0. In (iii) we must have that: $l_1 = l_2 = \cdots = l_{r-1} = 0$. □

The next lemma shows how to compute the number of satisfying assignments when composing two chains.

Lemma A.8. *Let C_1, C_2 be two composite chains that have a exactly on variable, x, in common. We assume, without loss of generality, that the common variable is the tail of both C_1 and C_2, that the tail of the composite chain $C_1 \wedge C_2$ is the degree-1 variable coming from C_1 and the head is the degree-1 coming from C_2.*

(i) *If x appears as a positive (negative, respectively) variable in both chains it must be either an antecedent in both chains or a consequent in both. In this case,*
- $N_{C_1 \wedge C_2} = N_{C_1|t=1} N_{C_2|t=1} + N_{C_1|t=0} N_{C_2|t=0}$,
- $N_{(C_1 \wedge C_2)|t=1} = N_{C_1|t=1,h=1} N_{C_2|t=1} + N_{C_1|t=0,h=1} N_{C_2|t=0}$,
- $N_{(C_1 \wedge C_2)|h=1} = N_{C_1|t=1} N_{C_2|t=1,h=1} + N_{C_1|t=0} N_{C_2|t=0,h=1}$,
- $N_{(C_1 \wedge C_2)|t=1,h=1} = N_{C_1|t=1,h=1} N_{C_2|t=1,h=1} + N_{C_1|t=0,h=1} N_{C_2|t=0,h=1}$.

(ii) *Assume x appears as a positive variable in C_1 and as a negative variable in C_2. In this case x must be an antecedent in one of the chains (say, C_1), and consequent in the other. We have:*
- $N_{C_1 \wedge C_2} = N_{C_1|t=1} N_{C_2|t=0} + N_{C_1|t=0} N_{C_2|t=1}$,
- $N_{(C_1 \wedge C_2)|t=1} = N_{C_1|t=1,h=1} N_{C_2|t=0} + N_{C_1|t=0,h=1} N_{C_2|t=1}$,
- $N_{(C_1 \wedge C_2)|h=1} = N_{C_1|t=0} N_{C_2|t=1,h=1} + N_{C_1|t=1} N_{C_2|t=0,h=1}$,
- $N_{(C_1 \wedge C_2)|t=1,h=1} N_{C_1|t=1,h=1} N_{C_2|t=0,h=1} + N_{C_1|t=0,h=1} N_{C_2|t=1,h=1}$.

Proof. The proof is immediate from the notational assumption made and the observation that all the possible satisfying assignments are counted that way, and no other satisfying assignment is possible. □

Lemma A.9. *Let C_1, C_2 be two composite chains that have exactly two variables, x, y, in common. We assume, without loss of generality, that the literals whose variable is x are in the tail of both C_1 and C_2, and those whose variable is y are in the head of both chains. Since the result of this composition is a closed chain it is sufficient to compute $N_{C_1 \wedge C_2}$.*

(i) *If both x and y appear as a positive (negative, respectively) variables in both chains (i.e., each must be either an antecedent in both chains or a consequent in both) we have:*

$$N_{C_1 \wedge C_2} = \sum_{b_1,b_2 \in \{0,1\}} N_{C_1|t=b_1,h=b_2} N_{C_2|t=b_1,h=b_2}.$$

(ii) *If both occurrences of x are positive and y appears once as a positive variable and once negated we have:*

$$N_{C_1 \wedge C_2} = \sum_{b_1,b_2 \in \{0,1\}} N_{C_1|t=b_1,h=b_2} N_{C_2|t=b_1,h=\overline{b_2}}.$$

(iii) *If both x and y appear as positive variable in one chain and negated in the other, then we have:*

$$N_{C_1 \wedge C_2} = \sum_{b_1,b_2 \in \{0,1\}} N_{C_1|t=b_1,h=b_2} N_{C_2|t=\overline{b_1},h=\overline{b_2}}.$$

As an example, consider the case of composing two simple maximal chains

$$C_1 \wedge C_2 = (x_1 \to x_2 \to \cdots \to x_{r_1-1} \to z) \wedge (y_1 \to y_2 \to \cdots \to y_{r_2-1} \to z),$$

where the x_i's are different from the y_j's, z is the tail variable in both chains, and x_1, y_1 are the tail and the head, respectively of the composite chain. It is easy to see that

$$N_{C_1 \wedge C_2} = r_1 r_2 + 1,$$
$$N_{C_1 \wedge C_2|t=1} = N_{C_1 \wedge C_2|x_1=1} = r_2,$$
$$N_{C_1 \wedge C_2|h=1} = N_{C_1 \wedge C_2|y_1=1} = r_1,$$
$$N_{C_1 \wedge C_2|t=1,h=1} = 1.$$

If also $x_1 = y_1$ then $N_{C_1 \wedge C_2} = (r_1 - 1)(r_2 - 1) + 2$.

Proof. The proof is immediate from the notational assumption made and the observation that all the possible satisfying assignments are counted that way, and no other satisfying assignment is possible. □

With the observations that the computation above can be done in time polynomial in the size of the formula, this completes the correctness proof for algorithm **Count-2μ-2CNF**. We note that in the case of clauses that contain a constant, T or F, since all of them appear either as tail or head of a chain the algorithm can handle multiple occurrences of them. This in fact is true in general. The composition rule holds if we require only that the degree of variables that appear internal to a simple chain is at most 2, while the degree of all other variables, those whose all occurrences in the theory iseither positive or negative, is not restricted. The problem is that in this case every composite chain can have more than two possible connection points, and the number of N_*'s we need to keep track of, in order to implement the algorithm, grows exponentially. Therefore, we can allow no more than a logarithmic (in n) number of variables with unrestricted degree.

Proof of (3). By Lemma A.1, given $\Sigma \in$ Acyclic-2MONCNF, it is sufficient to count the number of independent sets in the corresponding graph which is, by the definition of an Acyclic-2MONCNF formula (Section 3), an acyclic graph. We first consider the case of a connected acyclic graph, a tree.

Lemma A.10. *Let T be a tree on n vertices. The number of independent sets of T can be computed in time* $O(n)$.

Proof. Let T be a tree with root r. For a vertex $x \in T$, we denote by T_x the subtree of T with x as root. $c(x)$ denotes the set of all vertices which are children of x in T, and $gc(x)$ the grandchildren of x in T. We denote by IS_x the number[13] of independent sets of the tree rooted at x. Among these independent sets, $IS_x(x)$ denotes the number of those which contain the root x, and $IS_x(\not{x})$ denotes the number of those which do not contain the root x.

Notice that, for all $y \in c(x)$ and any independent set I of T_x, $I_y = \{z \in I \cap T_y\}$ is an independent set in T_y. We use this in the next claim to represent the number of independent sets in T_x in terms of the number of independent sets of subtrees of T_x.

[13] As before, we count the empty set as one of the independent sets.

```
Count-IS-Tree(T):
  For all $x \in R_0$, $IS_x = 2$.
  For all $x \in R_1$, $IS_x = 1 + 2^{|c(x)|}$.
  For $i = 3, 4, \ldots, r$ do:
    For all $x \in R_i$, $IS_x = \prod_{z \in gc(x)} IS_z + \prod_{y \in c(x)} IS_y$.
End
```

Fig. A.1. Counting independent sets of a tree.

Claim A.11. *For $IS_x, IS_x(x), IS_x(\not x)$ as defined above, we have*:

(i) $IS_x(\not x) = \prod_{y \in c(x)} IS_y$,
(ii) $IS_x(x) = \prod_{z \in gc(x)} IS_z$,
(iii) $IS_x = IS_x(x) + IS_x(\not x) = \prod_{z \in gc(x)} IS_z + \prod_{y \in c(x)} IS_y$.

Proof. If $c(x) = \{y_1, y_2, \cdots, y_k\}$ and $\{I_{y_i} \subseteq T_{y_i}\}_{i=1}^k$ any collection of independent sets (in the respective trees) then $\bigcup_{i=1}^k I_{z_i}$ is an independent set in T_x that does not contain x. For the other direction, clearly any independent set $I \subseteq T_x$ that does not contain x can be decomposed uniquely as above. For (ii), similarly, if I is an independent set in T_x and $x \in I$, then clearly $\forall z \in gc(x)$, $I_z = \{z \in I \cap T_z\}$ is an independent set in T_z. For the other direction, if $gc(x) = \{z_1, z_2, \ldots, z_k\}$ and $\{I_{z_i} \subseteq T_{z_i}\}_{i=1}^k$ any collection of independent sets in T_{z_i}, then clearly $\bigcup_{i=1}^k I_{z_i} \cup \{x\}$ is an independent set in T_x, since it contains no vertex from $c(x)$. (iii) is immediate from (i) and (ii). □

We now present an algorithm, **Count-IS-Tree**, that computes the number of independent sets of a given tree. We denote by $r(x)$ the *rank* of the vertex $x \in T$. The rank of x is defined as follows: If x is a leaf, $r(x) = 0$. If x is an internal node in the tree we define, $r(x) = 1 + \max_{y \in c(x)} r(y)$. We denote by R_i the set of all vertices $x \in T$ such that $r(x) = i$, and assume w.l.o.g. that the tree T is representated as a collection of its sets R_i. In the algorithm, we compute the number IS_x of independent sets of a tree rooted at $x \in R_i$, given the values IS_y for all $y \in R_j$, for $j < i$.

For the boundary conditions, notice that if the tree contains a single vertex x, then $IS_x = 2$, and if the children of the root are leaves, then $IS_x = 1 + 2^{\#\text{leaves}}$. Thus, the correctness of the algorithm follows from Claim A.11 and the discussion above, and this completes the proof of the lemma. □

Since an acyclic undirected graph is a union of disjoint trees, using Lemma A.6 completes the proof.

Proof of (4). As in the proof of (3) we assume that in the graph that corresponds to the 2HORN formula Σ every connected component is a *tree*. The counting algorithm is very similar to the one presented for the acyclic monotone case. We prove a claim that is analog to Claim A.11, and use it to count the satisfying assignments as in the algorithm **Count-IS-Tree**(T) above.

Claim A.12. *Let N_x denote the number of assignments that satisfy the conjunction of clauses that correspond to a tree rooted at x (with respect to these variables only). $N_x(0)$ (respectively, $N_x(1)$) denotes those assignments in which x is assigned* 0 *(respectively,* 1*). We have that:*

(i) $N_x(0) = \prod_{y\in c(x)} N_y$,
(ii) $N_x(1) = \prod_{y\in c(x)} N_y(1)$,
(iii) $N_x = N_x(0) + N_x(1) = \prod_{y\in c(x)} N_y + \prod_{y\in c(x)} N_y(1)$.

Proof. To prove (i) we observe that since x is assigned 0, there are no restrictions on the satisfying assignments of the tree rooted at $y \in c(x)$. Since the subtrees rooted at different elements of $c(x)$ are disjoint, we get the result. We get (ii) by observing that an assignment satisfies the formula corresponding to T_x, where x is assigned 1, iff all $y \in c(x)$ are assigned 1. (iii) is immediate from (i) and (ii). □

Noticing that if the corresponding tree is of depth 1, the number of satisfying assignments is $1 + 2^{\#\text{leaves}}$, serves as the boundary condition for the procedure, that uses Claim A.12 to count the number of satisfying assignments of the Acyclic-2HORN formula Σ. Similar to the algorithm **Count-IS-Tree(*T*)** in the proof of (3) above we get an algorithm that computes the number of satisfying assignments efficiently. □

Acknowledgements

I am very grateful to Les Valiant for very helpful discussions and for his comments on an earlier draft of this paper. I would also like to thank Karen Daniels, Roni Khardon and Salil Vadhan for their comments on an earlier draft of this paper.

References

[1] F. Bacchus, *Representing and Reasoning with Probabilistic Knowledge: A Logical Approach to Probabilities* (MIT Press, Cambridge, MA, 1990).

[2] F. Bacchus, A. Grove, J.Y. Halpern, and D. Kolle, From statistics to beliefs, in: *Proceedings AAAI-92*, San Jose, CA (1992) 602–608.

[3] M. Cadoli, Semantical and computational aspects of Horn approximations, in: *Proceedings IJCAI-93*, Chambery, France (1993) 39–44.

[4] R. Carnap, *Logical Foundations of Probability* (University of Chicago Press, Chicago, IL, 1950).

[5] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, *Artif. Intell.* **42** (1990) 393–405.

[6] P. Dagum and M. Luby, Approximating probabilistic inference in Bayesian belief networks is NP-hard, *Artif. Intell.* **60** (1993) 141–153.

[7] R. Dechter, Constraint networks, in: G.S. Shapiro, ed., *Encyclopedia of Artificial Intelligence* (Wiley, New York, 1992).

[8] R. Dechter and J. Pearl, Network-based heuristics for constraint-satisfaction problems, *Artif. Intell.* **34** (1988) 1–38.

[9] R. Dechter and J. Pearl, Structure identification in relational data, *Artif. Intell.* **58** (1992) 237–270.

[10] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, CA, 1979).

[11] R. Greiner and D. Schuurmans, Learning useful Horn approximations, in: *Proceedings International Conference on the Principles of Knowledge Representation and Reasoning*, Cambridge, MA (1992) 383-392.

[12] A. Grove, J.Y. Halpern and D. Koller, Asymptotic conditional probabilities for first-order logic, in: *Proceedings 24th ACM Symposium of the Theory of Computing* (1992) 294-305.

[13] S. Holtzman, *Intelligent Decision Systems* (Addison-Wesley, Reading, MA, 1989).

[14] M.R. Jerrum, L.G. Valiant and V.V. Vazirani, Random generation of combinatorial structures from a uniform distribution, *Theoret. Comput. Sci.* **43** (1986) 169-188.

[15] R. Karp and M. Luby, Monte-Carlo algorithms for enumeration and reliability problems, in: *Proceedings 24th IEEE Symposium of Foundations of Computer Science* (1983) 56-64.

[16] H. Kautz, M. Kearns and B. Selman, Horn approximations of empirical data, *Artif. Intell.* **74** (1995) 129-146.

[17] H. Kautz and B. Selman, A general framework for knowledge compilation, in: *Proceedings International Workshop on Processing Declarative Knowledge*, Kaiserslautern, Germany (1991).

[18] H. Kautz and B. Selman, Forming concepts for fast inference, in: *Proceedings AAAI-92*, San Jose, CA (1992) 786-793.

[19] H. Kautz and B. Selman, An empirical evaluation of knowledge compilation by theory approximation, in: *Proceedings AAAI-94*, Seattle, WA (1994) 155-161.

[20] R. Khardon and D. Roth, Learning to reason, in: *Proceedings AAAI-94*, Seattle, WA (1994) 682-687; also: Technical Report TR-02-94, Aiken Computation Lab., Harvard University, Cambridge, MA (1994).

[21] R. Khardon and D. Roth, Reasoning with models, in: *Proceedings of the National Conference on Artificial Intelligence* (1994) 1148-1153. Submitted for publication. Full version: Technical Report TR-01-94, Aiken Computation Lab., Harvard University (1994).

[22] H. Levesque, Making believers out of computers, *Artif. Intell.* **30** (1986) 81-108.

[23] U. Montanari, Networks of constraint: fundamental properties and applications to picture processing, *Inf. Sci.* **7** (1974) 95-132.

[24] N.J. Nilsson, Probabilistic logic, *Artif. Intell.* **28** (1986) 71-87.

[25] P. Orponen, Dempster's rule of combination is #P-complete, *Artif. Intell.* **44** (1990) 245-253.

[26] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann, San Mateo, CA, 1988).

[27] J. Pearl, Reasoning with belief functions: an analysis of compatibility, *Int. J. Approximate Reasoning* **4** (1990) 343-389.

[28] D. Poole, Average-case analysis of a search algorithm for estimating prior and posterior probabilities in Bayesian networks with extreme probabilities, in: *Proceedings IJCAI-93*, Chambery, France (1993) 607-612.

[29] J.S. Provan and M.O. Ball, The complexity of counting cuts and of computing the probability that a graph is connected, *SIAM J. Comput.* **12** (4) (1983) 777-788.

[30] M.G. Provan, A logical-based analysis of Dempster-Shafer theory, *Int. J. Approximate Reasoning* **4** (1990) 451-498.

[31] M.G. Provan, The validity of Dempster-Shafer belief functions, *Int. J. Approximate Reasoning* **6** (1992) 389-399.

[32] B. Selman and H. Kautz, Knowledge compilation using Horn approximations, in: *Proceedings AAAI-91*, Anaheim, CA (1991) 904-909.

[33] G. Shafer, Perspectives of the theory and practice of belief functions, *Int. J. Approximate Reasoning* **4** (1990) 323-362.

[34] A. Sinclair, Randomized algorithms for counting and generating combinatorial structures, Ph.D. Thesis, Department of Computer Science, University of Edinburgh (1988).

[35] L. Stockmeyer, On approximation algorithms for #P, *SIAM J. Comput.* **14** (1985) 849-861.

[36] S. Toda, On the computational power of PP and $\oplus$P, in: *Proceedings 30th IEEE Symposium of Foundations of Computer Science* (1989) 514-519.

[37] L.G. Valiant, The complexity of computing the permanent, *Theoret. Comput. Sci.* **8** (1979) 189-201.

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

[39] L.G. Valiant, A theory of the learnable, *Commun. ACM* **27** (11) (1984) 1134-1142.