From Weighted to Unweighted Model Counting [*] [†] [‡]

Supratik Chakraborty
Indian Institute of Technology,
Bombay

Dror Fried, Kuldeep S. Meel, Moshe Y. Vardi
Department of Computer Science,
Rice University

Abstract

The recent surge of interest in reasoning about probabilistic graphical models has led to the development of various techniques for probabilistic reasoning. Of these, techniques based on weighted model counting are particularly interesting since they can potentially leverage recent advances in unweighted model counting and in propositional satisfiability solving. In this paper, we present a new approach to weighted model counting via reduction to unweighted model counting. Our reduction, which is polynomial-time and preserves the normal form (CNF/DNF) of the input formula, allows us to exploit advances in unweighted model counting to solve weighted model counting instances. Experiments with weighted model counters built using our reduction indicate that these counters performs much better than a state-of-the-art weighted model counter.

1 Introduction

Probabilistic reasoning on graphical models has seen an unwavering interest in recent years, with compelling applications from diverse domains like medical diagnostics, behavioral studies, natural language processing and the like [Getoor and Taskar, 2007]. Several computational techniques have therefore been developed to address problems that arise in reasoning about probabilistic graphical models [Koller and Friedman, 2009]. One such important problem is that of probabilistic inference, wherein we are given a graphical model describing conditional dependencies between variables of interest, and we are required to compute the conditional probability of an event, i.e. valuations of a subset of variables, given some evidence in the form of valuations of another subset of variables. It is known [Cooper, 1990; Roth, 1996; Chavira and Darwiche, 2008] that there is a deep connection between probabilistic inference and computing the aggregate weight of all models (or satisfying assignments) of a weighted propositional formula. Owing to the spectacular and sustained improvements in the capabilities of modern tools for propositional satisfiability solving and model counting [Belov *et al.*, 2014; Thurley, 2006], probabilistic inference via weighted model counting promises to be a strong alternative to other probabilistic inference techniques, such as variable elimination, belief propagation, and the like. This motivates investigating further the problem of computing aggregate weights of models of propositional formulas.

Given a propositional formula and a weight function that assigns a non-negative weight to every assignment of values to variables, *weighted model counting* (henceforth WMC) concerns summing weights of assignments that satisfy the formula. If every assignment has weight 1, the corresponding problem is often simply called *model counting*; for clarity of presentation, we use *unweighted model counting* (henceforth UMC) to denote this variant of the problem.

Both UMC and WMC have been extensively studied in theoretical and practical contexts. UMC for propositional formulas in both conjunctive normal form (CNF) and disjunctive normal form (DNF) were shown to be #P-complete in [Valiant, 1979]. Subsequently, #P-completeness of WMC for CNF formulas was stated in [Roth, 1996].

On the practical front, both UMC and WMC have applications in diverse areas even beyond probabilistic reasoning, viz. network reliability estimation, statistical physics, program synthesis and system verification [Roth, 1996; Sang *et al.*, 2004; Domshlak and Hoffmann, 2007; Xue *et al.*, 2012]. Several tools and algorithms for model counting, in both the weighted and unweighted settings, have therefore been proposed in the recent past [Bayardo and Schrag, 1997; Darwiche, 2004; Sang *et al.*, 2005; Thurley, 2006].

Many applications, including probabilistic inference, of WMC arising from real-world can be expressed by a *literal-weighted* representation, in which the weight of an assignment is the product of weights of its literals [Chavira and Darwiche, 2008]. We use this representation throughout this paper, and use *literal-weighted* WMC to denote the corresponding WMC problem. Note that literal-weighted WMC problems for both CNF and DNF formulas arise in real-life

[*]The author list has been sorted alphabetically by last name; this should not be used to determine the extent of authors' contributions.

[†]The full version is available at `http://www.cs.rice.edu/CS/Verification/Projects/CUSP/`

[‡]This work was supported in part by NSF grants CNS 1049862, CCF-1139011, by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering", by BSF grant 9800096, by a gift from Intel, by a grant from the Board of Research in Nuclear Sciences, India, Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097.

applications; e.g., DNF formulas are used in problems arising from probabilistic databases [Dalvi and Suciu, 2007], while CNF is the de-facto form of representation for probabilistic-inference problems [Chavira and Darwiche, 2008].

Recent approaches to WMC have focused on adapting UMC techniques to work in the weighted setting [Sang *et al.*, 2005; Choi and Darwiche, 2013; Chakraborty *et al.*, 2014a]. Such adaption requires intimate understanding of the implementation details of the UMC techniques, and on-going maintenance, since some of these techniques evolve over time. In this paper, we flip this approach and present an efficient reduction of literal-weighted WMC to UMC. The reduction preserves the normal form of the input formula, i.e. it provides the UMC formula in the same normal form as the input WMC formula. Therefore, an important contribution of our reduction is to provide a WMC-to-UMC module that allows any UMC solver, viewed as a *black box*, to be converted to a WMC solver. This enables the automatic leveraging of progress in UMC solving to progress in WMC solving.

We have implemented our WMC-to-UMC module on top of state-of-the-art exact unweighted model counters to obtain exact weighted model counters for CNF formulas with literal-weighted representation. Experiments on a suite of benchmarks indicate that the resulting counters scale to significantly larger problem instances than what can be handled by a state-of-the-art exact weighted model counter [Choi and Darwiche, 2013]. Our results suggest that we can leverage powerful techniques developed for SAT and related domains in recent years to handle probabilistic inference queries for graphical models encoded as WMC instances. Furthermore, we demonstrate that our techniques can be extended to more general representations where weights are associated with constraints instead of individual literals.

The remainder of the paper is organized as follows. We introduce notation and preliminaries in Section 2, and discuss related work in Section 3. We present our main technical contribution – a polynomial-time reduction from WMC to UMC – in Section 4. Ising our reduction, we have implemented a literal-weighted exact model counter module called WeightCount. In Section 5, we present results of our experiments using WeightCount on top of state-of-the-art UMC solvers, and compare them with SDD – a state-of-the-art exact weighted model counter. We then demonstrate, in Section 6, that our reduction can be extended to more general representation of associating weights with constraints. Finally, we discuss future work and conclude in Section 7.

2 Notation and Preliminaries

Let F be a Boolean formula and let X be the set of variables appearing in F. For a variable $x \in X$, we denote the assignment of x to *true* by x^1 and the assignment of x to *false* by x^0. A *satisfying assignment* or a *witness* of F is an assignment of variables in X that makes F evaluate to *true*. We denote the set of all witnesses of F by R_F. If σ is an assignment of variables in X and $x \in X$, we use $\sigma(x)$ to denote the value assigned to x in σ. For notational convenience, whenever the formula F is clear from the context, we omit mentioning it.

As mentioned earlier, we focus on literal-weighted WMC, in which weights are assigned to literals, and the weight of an assignment is the product of weights of its literals. For a variable x of F and a weight function $W(\cdot)$, we use $W\big(x^1\big)$ and $W\big(x^0\big)$ to denote the weights of the positive and negative literals, respectively. Adopting terminology used in [Sang *et al.*, 2005], we assume that every variable x either has an *indifferent* weight, i.e. $W\big(x^0\big) = W\big(x^1\big) = 1$, or a *normal* weight, i.e. $W\big(x^0\big) = 1 - W\big(x^1\big)$, where $0 \leq W\big(x^1\big) \leq 1$. This assumption stands well, as many real-world applications, and in particular probabilistic inference, can be efficiently reduced to literal-weighted WMC where every variable has either indifferent or normal weight [Chavira and Darwiche, 2008]. For notational convenience, henceforth whenever we say "literal-weighted WMC", we mean literal-weighted WMC with indifferent or normal weights of variables. Note that having a variable with a normal weight of $W\big(x^1\big) = 1$ (resp. $W\big(x^1\big) = 0$) makes the variable x redundant for WMC, since in this case only assignments σ with $\sigma(x) = \mathit{true}$ (resp. $\sigma(x) = \mathit{false}$) can contribute to the total weight. Thus, we can assign true (resp. false) to x without changing the overall weight of models of the formula. This suggests that we can further assume $0 < W\big(x^1\big) < 1$ for every variable x with a normal weight.

To avoid notational clutter, we overload $W(\cdot)$ to denote the weight of a literal, assignment or formula, depending on the context. Given a set Y of assignments, we use $W(Y)$ to denote $\sum_{\sigma \in Y} W(\sigma)$. Given a formula F, we use $W(F)$ to denote $\sum_{\sigma \in R_F} W(\sigma)$. For example, the formula $F = (x_1 \leftrightarrow \neg x_2)$ has two satisfying assignments: $\sigma_1 = (x_1 : \mathit{true}, x_2 : \mathit{false})$, and $\sigma_2 = (x_1 : \mathit{false}, x_2 : \mathit{true})$. Thus, we have $W(\sigma_1) = W\big(x_1^1\big) \cdot W\big(x_2^0\big)$ and $W(\sigma_2) = W\big(x_1^0\big) \cdot W\big(x_2^1\big)$. The weight of F, or $W(F)$, is then $W(\sigma_1) + W(\sigma_2)$.

For every variable x_i with normal weight, we assume that $W\big(x_i^1\big)$, which is a positive fraction, is specified in binary using m_i bits. Without loss of generality, the least significant bit in the binary representation of $W\big(x_i^1\big)$ is always taken to be 1. Thus, the rational decimal representation of $W\big(x_i^1\big)$ is $k_i/2^{m_i}$, where k_i is an odd integer in $\{1, \ldots 2^{m_i} - 1\}$. It follows that $W\big(x_i^0\big)$ is $(2^{m_i} - k_i)/2^{m_i}$. Let N_F denote the set of indices of variables in X that have normal weights, and let $\widehat{m} = \Sigma_{i \in N_F} m_i$. Let $C_F = \prod_{i \in N_F} 2^{-m_i} = 2^{-\widehat{m}}$. Note that $W(\sigma)/C_F$ is a natural number for every assignment σ; hence $W(F)/C_F$ is a natural number as well.

An instance of literal-weighted WMC can be defined by a pair $(F, W(\cdot))$, where F is a Boolean formula, and $W(\cdot)$ is the weight function for literals. Given $(F, W(\cdot))$, a *weighted model counter* returns $W(F)$.

Our work uses a special class of Boolean formulas, called *chain formulas*, that are inductively defined as follows. Every literal (i.e., a variable or its complement) is a chain formula. In addition, if l is a literal and φ is a chain formula in which neither l nor $\neg l$ appears, then $(l \vee \varphi)$ and $(l \wedge \varphi)$ are also chain formulas. Note that chain formulas can be represented as $l_1\, C_1\, (l_2\, C_2\, (\cdots (l_{n-1}\, C_{n-1}\, l_n) \cdots))$, where the l_i's are literals and every C_i is either the connector "$\vee$" or the connector "$\wedge$". It is easy to see from De Morgan's laws that if φ is a chain formula, then so is $\neg\varphi$.

3 Related Work

Theoretical investigations into the problem of model counting were initiated by Valiant, who first showed that UMC is #P-complete for both CNF and DNF formulas [Valiant, 1979]. Subsequently, Roth asserted that probabilistic inference is #P-complete, which by a known connection with WMC [Cooper, 1990; Chavira and Darwiche, 2008], implies that WMC is also #P-complete for both CNF and DNF formulas [Roth, 1996].

On the practical side, the earliest efforts at WMC such as CDP [Birnbaum and Lozinskii, 1999] were inspired from DPLL-style SAT solvers and consisted of incrementally counting the number of solutions after a partial solution was found. Subsequently, heuristics such as component caching, clause learning, no-good learning and the like improved upon the initial approach and ensuing counters such as Relsat [Bayardo and Schrag, 1997], Cachet [Sang *et al.*, 2004], and sharpSAT [Thurley, 2006] have been shown to scale to larger formulas. These approaches were later *manually* adapted for WMC in Cachet [Sang *et al.*, 2005].Again, alternative approaches based on BDDs and their variants [Löbbing and Wegener, 1996; Xue *et al.*, 2012] have also been proposed for UMC. Similar to SAT-based approaches, the resulting solvers have also been *manually* adapted for WMC, resulting in solvers like SDD [sdd, 2014].

In this paper, we adopt a different approach and propose to solve WMC by reducing it to UMC. Our key contribution lies in showing that this reduction is efficient and effective, thereby making it possible to solve weighted model counting problems using *any* unweighted model counter as a black-box. Our reduction makes use of chain formulas to encode each weighted variable. Interestingly, these formulas can be viewed as adaptations of switching circuits proposed by [Wilhelm and Bruck, 2008] in the context of stochastic switching networks. Chain formulas are also reminiscent of the log-encoding approach of encoding variables with bounded domains in the CSP literature [Eén and Sörensson, 2006; Frisch *et al.*, 2006; Gavanelli, 2007; Huang, 2008; Walsh, 2000]. Indeed, a chain formula encoding a variable with weight $k/2^m$ is logically equivalent to the constraint $(X \geq 2^m - k)$, where X is an unsigned integer represented using m boolean variables, as described in [Gavanelli, 2007; Walsh, 2000]. The use of log-encoding for exact counting weighted models of Boolean formulas is novel, to the best of our knowledge.

4 From Literal-weighted WMC to UMC

In this section, we first show how chain formulas can be used to represent normal weights of variables. We then present two polynomial-time reductions from WMC to UMC using chain formulas. These constitute the key technical contributions of the paper. These reductions, though related, are motivated by the need to preserve different normal forms (CNF and DNF) of the input formula. Finally, we discuss the optimality of our reductions with respect to number of variables in the unweighted formula.

Representing Weights using Chain Formulas

The central idea of our reduction is the use of chain formulas to represent weights. Let $m > 0$ be a natural number, and $k < 2^m$ be a positive odd number. Let $c_1 c_2 \cdots c_m$ be the m-bit binary representation of k, where c_m is the least significant bit. We then construct a chain formula $\varphi_{k,m}(\cdot)$ on m variables $a_1, \ldots a_m$ as follows. For every j in $\{1, \ldots m-1\}$, let C_j be the connector "$\vee$" if $c_j = 1$, and the connector "$\wedge$" if $c_j = 0$. Define

$$\varphi_{k,m}(a_1, \cdots a_m) = a_1\, C_1\, (a_2\, C_2 (\cdots (a_{m-1}\, C_{m-1}\, a_m) \cdots))$$

For example, consider $k = 5$ and $m = 4$. The binary representation of 5 using 4 bits is 0101. Therefore, $\varphi_{5,4}(a_1, a_2, a_3, a_4) = a_1 \wedge (a_2 \vee (a_3 \wedge a_4))$. We first show in Lemma 1 that $\varphi_{k,m}(\cdot)$ has exactly k satisfying assignments. Next, as a simple application of the distributive laws of Boolean algebra, Lemma 2 shows that every chain formula can be efficiently represented in both CNF and DNF. The proofs of these and other lemmas are given in the full version of the paper.

Lemma 1. *Let $m > 0$ be a natural number, $k < 2^m$, and $\varphi_{k,m}$ as defined above. Then $|\varphi_{k,m}|$ is linear in m and $\varphi_{k,m}$ has exactly k satisfying assignments.*

Recall from Section 2 that N_F denotes the set of indices of normal-weighted variables in F. For i in N_F, let $W(x_i^1) = k_i/2^{m_i}$, where k_i is a positive odd number less than 2^{m_i}. Additionally, let $\{x_{i,1}, \ldots x_{i,m_i}\}$ be a set of m_i "fresh" variables (i.e. variables that were not used before) for each i in N_F. We call the chain formula $\varphi_{k_i,m_i}(x_{i,1} \cdots x_{i,m_i})$, the *representative formula of x_i*. For notational clarity, we simply write φ_{k_i,m_i} when the arguments of the representative formula are clear from the context.

Lemma 2. *Every chain formula ψ on n variables is equivalent to a CNF (resp.,* DNF*) formula ψ^{CNF} (resp., ψ^{DNF}) having at most n clauses. In addition, $|\psi^{\mathsf{CNF}}|$ (resp., $|\psi^{\mathsf{DNF}}|$) is in $O(n^2)$.*

Polynomial-time Reductions

We now present two reductions from literal-weighted WMC to UMC. Since weighted model count is a real number in general, while unweighted model count is a natural number, any reduction from WMC to UMC must use a normalization constant. Given that all literal weights are of the form $k_i/2^{m_i}$, a natural choice for the normalization constant is $C_F = \prod_{i \in N_F} 2^{-m_i}$. Theorem 1a gives a transformation of an instance $(F, W(\cdot))$ of literal-weighted WMC to an unweighted Boolean formula $\widehat{F}$ such that $W(F) = C_F \cdot |R_{\widehat{F}}|$. This reduction is motivated by the need to preserve CNF form of the input formula. We may also allow an additive correction term when doing the reduction. Theorem 1b provides a transformation of $(F, W(\cdot))$ to an unweighted Boolean formula $\breve{F}$ such that $W(F) = C_F \cdot |R_{\breve{F}}| - 2^n + 2^{n-|N_F|}$. The motivation for this reduction comes from the need to preserve DNF form of the input formula. Both reductions take time linear in the size of F and in the number of bits required to represent the weights of normal-weighted variables in F.

Note that since $C_F = 2^{-\widehat{m}}$, computing $C_F \cdot |R_{\widehat{F}}|$ (respectively, $C_F \cdot |R_{\breve{F}}|$) amounts to computing $|R_{\widehat{F}}|$ (respectively, $|R_{\widehat{F}}|$), which is an instance of UMC, and shifting the radix point in the binary representation of the result to the left by $\widehat{m}$ positions.

Theorem 1. *Let $(F, W(\cdot))$ be an instance of literal-weighted* WMC*, where F has n variables. Then, we can construct in linear time the following unweighted Boolean formulas, each of which has $n + \widehat{m}$ variables and is of size linear in $|F| + \widehat{m}$.*

(a) $\widehat{F}$ such that $W(F) = C_F \cdot |R_{\widehat{F}}|$.

(b) $\breve{F}$ such that $W(F) = C_F \cdot |R_{\breve{F}}| - 2^n \cdot (1 - 2^{-|N_F|})$

Proof. Let $X = \{x_1, \cdots, x_n\}$ be the set of variables of F. Without loss of generality, let $N_F = \{1, \cdots r\}$ be the set of indices of the normal-weighted variables of F. For each normal-weighted variable x_i, let $\varphi_{k_i,m_i}(x_{i,1}, \cdots x_{i,m_i})$ be the representative formula, as defined above. Let $\Omega = (x_1 \leftrightarrow \varphi_{k_1,m_1}) \wedge \cdots \wedge (x_r \leftrightarrow \varphi_{k_r,m_r})$.

Proof of part (a): We define the formula $\widehat{F}$ as follows.

$$\widehat{F} = F \wedge \Omega$$

Recalling $\widehat{m} = \sum_{i \in N_F} m_i$, it is easy to see that $\widehat{F}$ has $n + \widehat{m}$, variables. From Lemma 1, we know that $\varphi_{k_i,m_i}(x_{i,1}, \cdots x_{i,m_i})$ is of size linear in m_i, for every i in N_F. Therefore, the size of Ω is linear in $\widehat{m}$, and the size of $\widehat{F}$ is linear in $|F| + \widehat{m}$.

We now show that $W(F) = C_F \cdot |R_{\widehat{F}}|$. Let $W'(\cdot)$ be a new weight function, defined over the literals of X as follows. If x_i has indifferent weight, then $W'\big(x_i^0\big) = W'\big(x_i^1\big) = 1$. If x_i has normal weight with $W\big(x_i^1\big) = k_i/2^{m_i}$, then $W'\big(x_i^1\big) = k_i$ and $W'\big(x_i^0\big) = 2^{m_i} - k_i$. By extending the definition of $W'(\cdot)$ in a natural way (as was done for $W(\cdot)$) to assignments, sets of assignments and formulas, it is easy to see that $W(F) = W'(F) \cdot \prod_{i \in N_F} 2^{-m_i} = W'(F) \cdot C_F$.

Next, for every assignment σ of variables in X, let $\sigma^1 = \{i \in N_F \mid \sigma(x_i) = \mathit{true}\}$ and $\sigma^0 = \{i \in N_F \mid \sigma(x_i) = \mathit{false}\}$. Then, we have $W'(\sigma) = \prod_{i \in \sigma^1} k_i \prod_{i \in \sigma^0} (2^{m_i} - k_i)$. Let $\widehat{\sigma}$ be an assignment of variables appearing in $\widehat{F}$. We say that $\widehat{\sigma}$ is *compatible* with σ if for all variables x_i in X, we have $\widehat{\sigma}(x_i) = \sigma(x_i)$. Observe that $\widehat{\sigma}$ is compatible with exactly one assignment, viz. σ, of variables in X. Let S_σ denote the set of all satisfying assignments of $\widehat{F}$ that are compatible with σ. Then $\{S_\sigma | \sigma \in R_F\}$ is a partition of $R_{\widehat{F}}$. From Lemma 1, we know that there are k_i witnesses of φ_{k_i,m_i} and $2^{m_i} - k_i$ witnesses of $\neg\varphi_{k_i,m_i}$. Since the representative formula of every normal-weighted variable uses a fresh set of variables, we have from the structure of $\widehat{F}$ that if σ is a witness of F, then $|S_\sigma| = \prod_{i \in \sigma^1} k_i \prod_{i \in \sigma^0} (2^{m_i} - k_i)$. Therefore $|S_\sigma| = W'(\sigma)$. Note that if σ is not a witness of F, then there are no compatible satisfying assignments of $\widehat{F}$; hence $S_\sigma = \emptyset$ in this case. Overall, this gives

$$|R_{\widehat{F}}| = \sum_{\sigma \in R_F} |S_\sigma| + \sum_{\sigma \notin R_F} |S_\sigma| = \sum_{\sigma \in R_F} |S_\sigma| + 0 = W'(F) \,.$$

It follows that $W(F) = C_F \cdot W'(F) = C_F \cdot |R_{\widehat{F}}|$. This completes the proof of part (a).

Proof of part (b): We define the formula $\breve{F}$ as follows.

$$\breve{F} = \Omega \to F$$

Clearly, $\breve{F}$ has $n + \widehat{m}$ variables. Since the size of Ω is linear in $\widehat{m}$, the size of $\breve{F}$ is linear in $|F| + \widehat{m}$.

We now show that $W(F) = C_F \cdot |R_{\breve{F}}| - 2^n \cdot (1 - 2^{-|N_F|})$. First, note that $\breve{F}$ is logically equivalent to $\neg\Omega \vee (F \wedge \Omega) = \neg\Omega \vee \widehat{F}$, where $\widehat{F}$ is as defined in part (a) above. Since $\widehat{F}$ and $\neg\Omega$ are mutually inconsistent, it follows that $|R_{\breve{F}}|$ is the sum of $|R_{\widehat{F}}|$ and the number of satisfying assignments (over all variables in $\breve{F}$) of $\neg\Omega$. By definition, Ω does not contain any variable in $X \setminus N_F$. Hence, the number of satisfying assignments (over all variables in $\breve{F}$) of $\neg\Omega$ is $2^{n-|N_F|} \cdot |R_{\neg\Omega}|$. To calculate $|R_{\neg\Omega}|$, observe that $|R_{(x_i \leftrightarrow \varphi_{k_i,m_i})}| = 2^{m_i}$, and the sub-formulas $(x_i \leftrightarrow \varphi_{k_i,m_i})$ and $(x_j \leftrightarrow \varphi_{k_j,m_j})$ have disjoint variables for $i \neq j$. Therefore, $|R_\Omega| = \prod_{i \in N_F} 2^{m_i} = 2^{\widehat{m}}$, and $|R_{\neg\Omega}| = 2^{\widehat{m}+|N_F|} - 2^{\widehat{m}} = 2^{\widehat{m}} \cdot (2^{|N_F|} - 1)$. From part (a) above, we also know that $|R_{\widehat{F}}| = W(F)/C_F$. Hence, $|R_{\breve{F}}| = |R_{\widehat{F}}| + 2^{n-|N_F|} \cdot |R_{\neg\Omega}| = W(F)/C_F + 2^{n+\widehat{m}} \cdot (1 - 2^{-|N_F|})$. Rearraging terms, we get $W(F) = C_F \cdot \big(|R_{\breve{F}}| - 2^{n+\widehat{m}} \cdot (1 - 2^{-|N_F|})\big)$. Since $C_F = 2^{-\widehat{m}}$, we obtain $W(F) = C_F \cdot |R_{\breve{F}}| - 2^n \cdot (1 - 2^{-|N_F|})$. This completes the proof of part (b). $\square$

Preservation of Normal Forms

The representative formula of a normal-weighted variable is a chain formula, which is generally neither in CNF nor in DNF. Therefore, even if the input formula F is in a normal form (CNF/DNF), the formulas $\widehat{F}$ and $\breve{F}$ in Theorem 1 may be neither in CNF nor in DNF. We wish to ask if our reductions can be adapted to preserve the normal form (CNF/DNF) of F. Theorem 2 answers this question affirmatively.

Theorem 2. *Let $(F, W(\cdot))$ be an instance of literal-weighted* WMC*, where F is in* CNF *(resp.,* DNF*) and has n variables. We can construct in polynomial time a* CNF *(resp.,* DNF*) formula $F^\star$ such that $W(F) = C_F \cdot |R_{F^\star}|$ (resp., $C_F \cdot |R_{F^\star}| - 2^n \cdot (1 - 2^{-|N_F|})$). Moreover, $F^\star$ has $n + \widehat{m}$ variables and its size is linear in $(|F| + \sum_{i \in N_F} m_i^2)$.*

The key step in the proof is an application of Lemma 2 to show that Ω can be expressed in CNF form in size $O(\sum_{i \in N_F} m_i^2)$. The rest of the proof involves Boolean algebraic manipulations of $\widehat{F}$ and $\breve{F}$.

Optimality of Reductions

We now ask if there exists an algorithm that reduces literal-weighted WMC to UMC and gives unweighted Boolean formulas that have significantly fewer variables than $\widehat{F}$ or $\breve{F}$. We restrict our discussion to reductions that use C_F as a normalization constant, and perhaps use an additive correction term $D(n, W(\cdot))$ that is agnostic to F, and depends only on the number of variables in F and on the weight function. An example of such a term is $-2^n \cdot (1 - 2^{-|N_F|})$, used in Theorem 1, where N_F can be determined from $W(\cdot)$ by querying the weights of individual literals.

Theorem 3. *Let* Reduce$(\cdot)$ *be an algorithm that takes as input an instance* $(F, W(\cdot))$ *of literal-weighted* WMC*, and returns an unweighted Boolean formula* $\widetilde{F}$ *such that* $W(F) = C_F \cdot |R_{\widetilde{F}}| + D(n, W(\cdot))$*, where* $D(\cdot,\cdot)$ *is a real-valued function and* n *is the number of variables in* F*. Then* $\widetilde{F}$ *has at least* $n - 1 + \widehat{m} - 2|N_F|)$ *variables.*

Observe that the number of variables in $\widehat{F}$ and $\breve{F}$ in Theorem 1 differ from the lower bound given by Theorem 3 by $2|N_F| + 1$, which is independent of n as well as the number of bits ($\widehat{m}$) used to represent weights.

5 Experimental Analysis

The construction outlined in the proof of Theorem 1 naturally suggests an algorithm for solving WMC using a UMC solver as a black-box. This is particularly important in the context of weighted model counting, since state-of-the-art unweighted model counters (viz. sharpSAT [Thurley, 2006], DSharp [Muise *et al.*, 2012]) scale to much larger problem sizes than existing state-of-the-art weighted model counters (viz. SDD [Darwiche, 2011]). To investigate the practical advantages of using the reduction based approach, we developed a literal-weighted model counter module called WeightCount, that takes as input an instance $(F, W(\cdot))$ of literal-weighted WMC and reduces it to an instance $F^\star$ of UMC, as outlined in Theorem 1 and 2. The WeightCount module then invokes an underlying state-of-the-art exact UMC solver, to count the witnesses of $F^\star$. Finally, WeightCount computes $C_F \cdot |R^\star_F|$ as the weighted model count of $(F, W(\cdot))$. In our experiments we employed both sharpSAT and DSharp as the underlying exact UMC solver.

We conducted experiments on a suite of diverse CNF benchmarks to compare the performance of WeightCount with that of SDD. We also tried to compare our tool with the weighted variant of Cachet [Sang *et al.*, 2005], but despite extensive efforts, we have not been able to run this tool on our system.[1] We focused on CNF formulas because of the availability of CNF model counters and the lack of DNF model counters in the public-domain. The suite of benchmarks used in our experiments consisted of problems arising from probablistic inference in grid networks, synthetic grid-structured random interaction Ising models, plan recognition, DQMR networks, bit-blasted versions of SMTLIB benchmarks, ISCAS89 combinational circuits with weighted inputs, and program synthesis examples. We used a high performance cluster to conduct multiple experiments in parallel. Each node of the cluster had a 12-core 2.83 GHz Intel Xeon processor, with 4GB of main memory, and each of our experiments was run on a single core. Note that normal weights of variables in our benchmarks typically correspond to (conditional) probabilities of events in the original problem from which the benchmark is derived. To allow specification of probabilities with a precision of up to to two decimal places, we rounded off the weights such that all weights were of the form $k/2^i$ ($1 \leq i \leq 7$). A uniform timeout of 5 hours was used for all tools in our experiments.

Table 1 presents the results of comparing the performances of WeightCount and SDD on a subset of our benchmarks. [2] In this table, the benchmarks are listed in Column 1. Columns 2 and 3 list the number of variables and clauses, respectively, for each benchmark. Columns 4 through 8 present our experimental observations on running WeightCount via either sharpSAT or DSharp as UMC solvers. Specifically, columns 4 and 5 give the total number of variables and clauses of the unweighted formula obtained after applying our reduction. Note that these numbers are larger than the corresponding numbers in the original problem, since all normal-weighted variables in the original problem have been replaced by their respective representative formulas. The run-time of WeightCount via sharpSAT is the sum of the transform time taken to reduce a WMC instance to an instance of UMC, as presented in Column 6, and the counting time taken by sharpSAT to solve an instance of UMC, as presented in Column 7. The run-time of WeightCount via DSharp is the sum of the transform time as presented in Column 6, and and the counting time taken by DSharp to solve an instance of UMC, as presented in Column 8. Finally, run-time for SDD to solve the same instance of WMC is presented in column 9. A "-" in a column indicates that the corresponding experiment either did not complete within 5 hours or ran out of memory.

Overall, out of 79 benchmarks for which the weighted model count could be computed by either SDD or WeightCount, SDD timed/spaced out on 30 benchmarks, WeightCount via sharpSAT timed out on 1 benchmarks, and WeightCount via DSharp timed out on 11 benchmarks . Table 1 clearly shows that on most benchmarks WeightCount via either sharpSAT or DSharp outperformed SDD in terms of running time by 1 to 3 orders of magnitude. Moreover, WeightCount could generate weighted counts for a large class of benchmarks for which SDD timed out. Thus, our reduction helps in solving instances of literal-weighted WMC that are otherwise beyond the reach of a state-of-the-art weighted model counter. Significantly, column 6 of Table 1 demonstrates that the overhead for reducing a WMC problem to a UMC instance is very small. The comparison between WeightCount via sharpSAT and WeightCount via DSharp is interesting but beyond the scope of this work.

Overall, our experiments demonstrate that state-of-the-art UMC solvers can be augmented with an implementation of our reduction to obtain literal-weighted model counts on formulas with tens of thousands of variables – problems that are clearly beyond the reach of existing weighted model counters. Significantly, our approach requires no modification of the implementation of the UMC solver, which can therefore be treated as a black-box.

6 Beyond Literal Weights

While literal-weighted representation is typically employed in applications of WMC, richer forms of representations of

[1]The weighted variant of Cachet is broken, and its authors have acknowledged this and expressed their inability to fix the tool. We tried fixing it ourselves and have communicated with several other researchers in this respect, but to no avail.

[2]An extended version of Table 1 is will be available in full version.

			WeightCount					SDD
Benchmark	**Orig #vars**	**Orig #clas**	**Final #vars**	**Final #claus**	**Transform time (s)**	sharpSAT **counting time (s)**	DSharp **counting time (s)**	**Overall time (s)**
case_1_b11_1	340	1026	550	1266	0.03	92.16	1059.82	64.3
s1196a_15_7	777	2165	867	2285	0.06	0.54	8.88	–
case_2_b12_2	827	2725	917	2845	0.06	34.11	714.37	735.68
squaring1	891	2839	981	2959	0.04	10.02	97.86	–
cliquen30	930	1800	2517	3821	0.11	300.86	–	–
BN_63	1112	2661	1272	2853	0.04	0.68	8.68	–
BN_55	1154	2692	1314	2884	0.1	1.11	–	–
BN_47	1336	3376	1406	3460	0.11	0.11	1.49	170.92
BN_61	1348	3388	1418	3472	0.05	0.2	1.77	157.88
squaring9	1434	5028	1524	5148	0.07	32.68	721.14	–
squaring16	1627	5835	1723	5963	0.07	–	2623.12	–
BN_43	1820	3806	2240	4286	0.34	8393.12	–	–
BN_108	2289	8218	11028	19105	0.27	2.14	8.66	270.31
smokers_20	2580	3740	6840	8860	0.33	224.25	–	–
treemax	24859	103762	26353	105754	1.5	3.93	338.16	–
BN_26	50470	93870	276675	352390	244.29	68.99	259.42	693.09

Table 1: Performance comparison of WeightCount vis-a-vis SDD

weights are increasingly used in a wide variety of applications. Of these, associating weights to constraints instead of literals has been widely employed in probabilistic programming, verification, and the like [Ver, 2015; Pfeffer, 2009]. For example, Figaro, a popular probabilistic programming framework, contains a construct called *setConstraint* that associates weights with constraints. We now demonstrate that our techniques can be generalized to handle such representations as well.

Define ConstraintWMC to be a variant of WMC, wherein the weight of an assignment is specified using a set of constraints. Specifically, given a formula F, a set $\boldsymbol{G} = (G_1, \cdots G_r)$ of Boolean constraints, and a weight function $W(\cdot)$ over $\boldsymbol{G}$, the weight of an assignment σ is defined as the product of the weights of constraints in $\boldsymbol{G}$ that are satisfied by σ. The weight of every constraint G_i is assumed to be of the form $k_i/2^{m_i}$, where k_i is an odd integer between 1 and $2^{m_i} - 1$. In case σ satisfies none of the constraints in $\boldsymbol{G}$, the weight of σ is defined to be 1. The ConstraintWMC problem is to compute the sum of the weights of all witnesses of F.

By an extension of the reasoning used in the proof of Theorem 1a, we can obtain an efficient reduction from ConstraintWMC to UMC. We do not yet know how to do this preserving the normal form of the input formula.

Theorem 4. *Let $(F, \boldsymbol{G}, W(\cdot))$ be an instance of* ConstraintWMC*, where $|\boldsymbol{G}| = r$ and $\varphi_{k_i,m_i}(x_{i,1}, \cdots x_{i,m_i})$ is the chain formula that describes $W(G_i)$. Then by defining $\widehat{F} = F \wedge (G_1 \rightarrow \varphi_{k_1,m_1}) \wedge \cdots \wedge (G_r \rightarrow \varphi_{k_r,m_r})$, we get a linear-time reduction from* ConstraintWMC *to* UMC*, such that $W(F) = C_{\boldsymbol{G}} \cdot |R_{\widehat{F}}|$, where $C_{\boldsymbol{G}} = \prod_{i=1}^{r} 2^{-m_i}$.*

7 Discussion and Future Work

The recent surge of interest in probabilistic reasoning has propelled WMC to emerge as a key challenging problem. In this paper, we presented new polynomial-time reductions from WMC to UMC that preserve the normal form of the input formula. This provides a framework that allows any UMC solver, viewed as a black box, to be converted to a WMC solver. We also showed that our approach leads to an efficient practical algorithm for literal-weighted WMC for CNF formulas.

The proposed reductions open up new research directions. While we focused on exact WMC in this paper, the computational difficulty of exact inferencing in complex graphical models has led to significant recent interest in approximate WMC [Chakraborty *et al.*, 2014a]. Approximate WMC concerns computing the weighted model count of a formula within a specified multiplicative tolerance and with a desired minimum confidence. In this context, it is worth noting that the reduction proposed in Theorem 1a allows us to lift approximation guarantees from the unweighted to the weighted setting for CNF formulas. Unfortunately, this is not the case for the reduction proposed in Theorem 1b, which is required for DNF formulas. The question of whether there exists an approximation-preserving reduction from WMC to UMC that also preserves DNF is open. The practical feasibility of solving approximate WMC problems by reducing them to their unweighted counterpart, even in the case of CNF formulas, requires further detailed investigation. This is particularly challenging since the current reductions introduce extra variables, which is known to adversely affect XOR-hashing-based state-of-the-art approximation techniques [Ermon *et al.*, 2014; Chakraborty *et al.*, 2014b].

Another interesting direction of research is CNF/DNF-preserving reductions from ConstraintWMC to UMC. Investigations in this direction can lead to improvements in both modeling and inferencing techniques in probabilistic programming frameworks. The design of practically efficient unweighted DNF model counters is also a fruitful line of research, since our reduction allows us to transform any such tool to a weighted DNF model counter.

References

[Bayardo and Schrag, 1997] R. J. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. of AAAI*, pages 203–208, 1997.

[Belov *et al.*, 2014] A. Belov, D. Diepold, M. J. Heule, M. Järvisalo, et al. Proceedings of SAT competition. 2014.

[Birnbaum and Lozinskii, 1999] E. Birnbaum and E. L. Lozinskii. The good old Davis-Putnam procedure helps counting models. *Journal of Artificial Intelligence Research*, 10(1):457–477, June 1999.

[Chakraborty *et al.*, 2014a] S. Chakraborty, D. Fremont, K. S. Meel, S. Seshia, and M. Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proc. of AAAI*, 2014.

[Chakraborty *et al.*, 2014b] S. Chakraborty, K. S. Meel, and M. Y. Vardi. Balancing scalability and uniformity in sat-witness generator. In *Proc. of DAC*, pages 60:1–60:6, 2014.

[Chavira and Darwiche, 2008] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6):772–799, 2008.

[Choi and Darwiche, 2013] A. Choi and A. Darwiche. Dynamic minimization of sentential decision diagrams. In *Proc. of AAAI*, pages 187–194, 2013.

[Cooper, 1990] G. Cooper. The computational complexity of probabilistic inference using bayesian belief networks (research note). *Artificial Intelligence*, 42(2-3):393–405, 1990.

[Dalvi and Suciu, 2007] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, 2007.

[Darwiche, 2004] A. Darwiche. New advances in compiling CNF to decomposable negation normal form. In *Proc. of ECAI*, pages 328–332. Citeseer, 2004.

[Darwiche, 2011] A. Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *Proc. of IJCAI*, volume 22, page 819, 2011.

[Domshlak and Hoffmann, 2007] C. Domshlak and J. Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *Journal of Artificial Intelligence Research*, 30(1):565–620, 2007.

[Eén and Sörensson, 2006] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into sat. *JSAT*, 2(1-4):1–26, 2006.

[Ermon *et al.*, 2014] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Low-density parity constraints for hashing-based discrete integration. In *Proc. of ICML*, pages 271–279, 2014.

[Frisch *et al.*, 2006] A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Propagation algorithms for lexicographic ordering constraints. *Artificial Intelligence*, 170(10):803–834, 2006.

[Gavanelli, 2007] M. Gavanelli. The log-support encoding of CSP into SAT. In *Proc. of CP*, pages 815–822. Springer, 2007.

[Getoor and Taskar, 2007] L. Getoor and B. Taskar. *Introduction to statistical relational learning*. MIT press, 2007.

[Huang, 2008] J. Huang. Universal booleanization of constraint models. In *Proc. of CP*, pages 144–158, 2008.

[Koller and Friedman, 2009] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[Löbbing and Wegener, 1996] M. Löbbing and I. Wegener. The number of knight's tours equals 33,439,123,484,294 – counting with binary decision diagrams. *The Electronic Journal of Combinatorics*, 3(1):R5, 1996.

[Muise *et al.*, 2012] C. Muise, S. A. McIlraith, J. C. Beck, and E. I. Hsu. Dsharp: fast d-dnnf compilation with sharpsat. In *Advances in Artificial Intelligence*, pages 356–361. Springer, 2012.

[Pfeffer, 2009] A. Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, page 137, 2009.

[Roth, 1996] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996.

[Sang *et al.*, 2004] T. Sang, F. Bacchus, P. Beame, H. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Proc. of SAT*, 2004.

[Sang *et al.*, 2005] T. Sang, P. Beame, and H. Kautz. Performing Bayesian inference by weighted model counting. In *Proc. of AAAI*, pages 475–481, 2005.

[sdd, 2014] The SDD package. `http://reasoning.cs.ucla.edu/sdd/`, 2014.

[Thurley, 2006] M. Thurley. sharpSAT: Counting models with advanced component caching and implicit bcp. In *Proc. of SAT*, pages 424–429, 2006.

[Valiant, 1979] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Ver, 2015] System Verilog. `http://www.systemverilog.org`, 2015.

[Walsh, 2000] T. Walsh. SAT v CSP. In *Proc. of CP*, pages 441–456. Springer, 2000.

[Wilhelm and Bruck, 2008] D. Wilhelm and J. Bruck. Stochastic switching circuit synthesis. In *Proc. of ISIT*, pages 1388–1392, 2008.

[Xue *et al.*, 2012] Y. Xue, A. Choi, and A. Darwiche. Basing decisions on sentences in decision diagrams. In *Proc. of AAAI*, 2012.