Distribution-Aware Sampling and Weighted Model Counting for SAT * †

Supratik Chakraborty
Indian Institute of Technology, Bombay

Daniel J. Fremont
University of California, Berkeley

Kuldeep S. Meel
Department of Computer Science, Rice University

Sanjit A. Seshia
University of California, Berkeley

Moshe Y. Vardi
Department of Computer Science, Rice University

Abstract

Given a CNF formula and a weight for each assignment of values to variables, two natural problems are weighted model counting and distribution-aware sampling of satisfying assignments. Both problems have a wide variety of important applications. Due to the inherent complexity of the exact versions of the problems, interest has focused on solving them approximately. Prior work in this area scaled only to small problems in practice, or failed to provide strong theoretical guarantees, or employed a computationally-expensive most-probable-explanation (MPE) queries that assumes prior knowledge of a factored representation of the weight distribution. We identify a novel parameter, *tilt*, which is the ratio of the maximum weight of satisfying assignment to minimum weight of satisfying assignment and present a novel approach that works with a black-box oracle for weights of assignments and requires only an NP-oracle (in practice, a SAT-solver) to solve both the counting and sampling problems when the tilt is small. Our approach provides strong theoretical guarantees, and scales to problems involving several thousand variables. We also show that the assumption of small tilt can be significantly relaxed while improving computational efficiency if a factored representation of the weights is known.

1 Introduction

Given a set of weighted elements, computing the cumulative weight of all elements that satisfy a set of constraints is a fundamental problem that arises in many contexts. Known variously as weighted model counting, discrete integration and partition function computation, this problem has applications in machine learning, probabilistic reasoning, statistics, planning, and combinatorics, among other areas (Roth 1996; Sang et al. 2004; Domshlak and Hoffmann 2007; Xue, Choi, and Darwiche 2012). A closely related problem is that of sampling elements satisfying a set of constraints, where the probability of choosing an element is proportional to its weight. The latter problem, known as distribution-aware sampling or weighted sampling, also has important applications in probabilistic reasoning, machine learning, statistical physics, constrained random verification and other domains (Jerrum and Sinclair 1996; Bacchus, Dalmao, and Pitassi 2003; Naveh et al. 2006; Madras and Piccioni 1999). Unfortunately, the exact versions of both problems are computationally hard. Weighted model counting can be used to count the number of satisfying assignments of a CNF formula; hence it is #P-hard (Valiant 1979). As shown by (Toda 1989), $\mathsf{P}^{\#\mathsf{P}}$ contains the entire polynomial-time hierarchy, and thus #P-hard problems are structurally harder than NP-complete problems. Fortunately, approximate solutions to weighted model counting and weighted sampling are adequate for most applications. As a result, there has been significant interest in designing practical approximate algorithms for these problems. Before discussing approaches to the approximate weighted sampling and counting problems, we should pause to note that a fully polynomial randomized approximation scheme (FPRAS) for weighted sampling would yield an FPRAS for #P-complete inference problems (Jerrum and Sinclair 1996; Madras and Piccioni 1999) – a possibility that lacks any evidence so far. Therefore, even approximate versions of weighted sampling and counting are computationally challenging problems with applications to a wide variety of domains.

Since constraints arising from a large class of real-world problems can be modeled as propositional CNF (henceforth CNF) formulas, we focus on CNF formulas and assume that the weights of truth assignments are given by a weight function $w(\cdot)$. Roth showed that approximately counting the models of a CNF formula is NP-hard even when the structure of the formula is severely restricted (Roth 1996). By a result of Jerrum, Valiant and Vazirani 1986, we also know that approximate model counting and almost uniform sampling (a special case of approximate weighted sampling) are polynomially inter-reducible. Therefore, it is unlikely that there exist polynomial-time algorithms for either approximate weighted model counting or approximate weighted sampling (Karp, Luby, and Madras 1989). Recently, a new class of algorithms that use pairwise-independent random parity constraints and MPE (*most-probable-explanation*)-

*The full version is available at `http://arxiv.org/abs/1404.2984`

† This work was supported in part by NSF grants CNS 1049862, CCF-1139011, CCF-1139138, 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, by the Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097, and by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

queries have been proposed for solving both problems (Ermon et al. 2013c; 2014; 2013a). These algorithms provide strong theoretical guarantees (an FPRAS relative to the MPE-oracle), and have been shown to scale to medium-sized problems in practice. While this represents a significant step in our quest for practically efficient algorithms with strong guarantees for approximate weighted model counting and approximate weighted sampling, the use of MPE-queries presents issues that need to be addressed in practice. First, MPE is an optimization problem – significantly harder in practice than a feasibility query (CNF satisfiability) (Park and Darwiche 2004). Indeed, even the approximate version of MPE has been shown to be NP-hard (Ermon et al. 2013a). Second, the use of MPE-queries along with parity constraints poses scalability hurdles (Ermon et al. 2014), and it has been argued in (Ermon et al. 2013b) that the MPE-query-based weighted model counting algorithm proposed in (Ermon et al. 2013c) is unlikely to scale well to large problem instances. This motivates us to ask if we can design approximate algorithms for weighted model counting and weighted sampling that do not invoke MPE-queries at all, and do not assume any specific representation of the weight distribution.

Our primary contributions are twofold. First, we identify a novel parameter, *tilt*, defined as the ratio of the maximum weight of a satisfying assignment to the minimum weight of a satisfying assignment, which characterizes the hardness of the approximate weighted model counting and weighted sampling problems. Second, we provide an affirmative answer to the question posed above, when the tilt is small. Specifically, we show that two recently-proposed algorithms for approximate (unweighted) model counting (Chakraborty, Meel, and Vardi 2013b) and almost-uniform (unweighted) sampling (Chakraborty, Meel, and Vardi 2014; 2013a) can be adapted to work in the setting of weighted assignments, using only a SAT-solver (NP-oracle) and a black-box weight function $w(\cdot)$ when the tilt is small. A black-box weight function is useful in applications where a factored representation is not easily available, such as in probabilistic program analysis and constrained random simulation. We also present arguments why it might be reasonable to assume a small tilt for some important classes of problems; a detailed classification of problems based on their tilt, however, is beyond the scope of this work. For distributions with large tilt, we propose an adaptation of our algorithm, which requires a pseudo-Boolean satisfiability solver instead of an (ordinary) SAT-solver as an oracle.

2 Notation and Preliminaries

Let F be a Boolean formula in conjunctive normal form (CNF), and let X be the set of variables appearing in F. The set X is called the *support* of F. Given a set of variables $S \subseteq X$ and an assignment σ of truth values to the variables in X, we write $\sigma|_S$ to denote the projection of σ onto S. A *satisfying assignment* or *witness* of F is an assignment that makes F evaluate to true. We denote the set of all witnesses of F by R_F. For notational convenience, whenever the formula F is clear from the context, we omit mentioning it. Let $\mathcal{D} \subseteq X$ be a subset of the support such that there are no two satisfying assignments that differ only in the truth values of variables in $\mathcal{D}$. In other words, in every satisfying assignment, the truth values of variables in $X \setminus \mathcal{D}$ uniquely determine the truth value of every variable in $\mathcal{D}$. The set $\mathcal{D}$ is called a *dependent* support of F, and $X \setminus \mathcal{D}$ is called an *independent* support of F. Note that there may be more than one independent supports; for example, $(a \vee \neg b) \wedge (\neg a \vee b)$ has three, namely $\{a\}$, $\{b\}$, and $\{a, b\}$. Clearly, if $\mathcal{I}$ is an independent support of F, so is every superset of $\mathcal{I}$.

Let $w(\cdot)$ be a function that takes as input an assignment σ and yields a real number $w(\sigma) \in (0, 1]$ called the *weight* of σ. Given a set Y of assignments, we use $w(Y)$ to denote $\Sigma_{\sigma \in Y} w(\sigma)$. Our main algorithms (see Section 4) make no assumptions about the nature of the weight function, treating it as a black-box. In particular, we do not assume that the weight of an assignment can be factored into the weights of projections of the assignment on specific subsets of variables. The exception to this is Section 6, where we consider possible improvements when the weights are given by a known function, or "white-box". Three important quantities derived from the weight function are $w_{max} = \max_{\sigma \in R_F} w(\sigma)$, $w_{min} = \min_{\sigma \in R_F} w(\sigma)$, and the *tilt* $\rho = w_{max}/w_{min}$. Following standard definitions, the MPE (*most probable explanation*) is w_{max}. Thus an MPE-query for a CNF formula F and weight distribution $w(\cdot)$ seeks the value of w_{max}. Our algorithms require an upper bound on the tilt, denoted r, which is provided by the user. To maximize the efficiency of the algorithms, it is desirable to obtain as tight a bound on the tilt as possible.

We write $\mathsf{Pr}[X : \mathcal{P}]$ for the probability of outcome X when sampling from a probability space $\mathcal{P}$. For brevity, we omit $\mathcal{P}$ when it is clear from the context. The expected value of the outcome X is denoted $\mathsf{E}[X]$.

Special classes of hash functions, called *k-wise independent* hash functions, play a crucial role in our work (Bellare, Goldreich, and Petrank 1998). Let n, m and k be positive integers, and let $H(n, m, k)$ denote a family of k-wise independent hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$. We use $h \xleftarrow{R} H(n, m, k)$ to denote the probability space obtained by choosing a hash function h uniformly at random from $H(n, m, k)$. The property of k-wise independence guarantees that for all $\alpha_1, \ldots \alpha_k \in \{0,1\}^m$ and for all distinct $y_1, \ldots y_k \in \{0,1\}^n$, $\mathsf{Pr}\left[\bigwedge_{i=1}^{k} h(y_i) = \alpha_i : h \xleftarrow{R} H(n, m, k)\right] = 2^{-mk}$. For every $\alpha \in \{0,1\}^m$ and $h \in H(n, m, k)$, let $h^{-1}(\alpha)$ denote the set $\{y \in \{0,1\}^n \mid h(y) = \alpha\}$. Given $R_F \subseteq \{0,1\}^n$ and $h \in H(n, m, k)$, we use $R_{F,h,\alpha}$ to denote the set $R_F \cap h^{-1}(\alpha)$.

Our work uses an efficient family of hash functions, denoted as $H_{xor}(n, m, 3)$. Let $h : \{0,1\}^n \to \{0,1\}^m$ be a hash function in the family, and let y be a vector in $\{0,1\}^n$. Let $h(y)[i]$ denote the i^{th} component of the vector obtained by applying h to y. The family of hash functions of interest is defined as $\{h(y) \mid h(y)[i] = a_{i,0} \oplus (\bigoplus_{l=1}^{n} a_{i,l} \cdot y[l]), a_{i,j} \in \{0,1\}, 1 \leq i \leq m, 0 \leq j \leq n\}$, where $\oplus$ denotes the XOR operation. By choosing values of $a_{i,j}$ randomly and independently, we can effectively choose a random hash function from the family. It has been shown in (Gomes, Sabharwal,

and Selman 2007) that this family of hash functions is 3-wise independent.

Given a CNF formula F, an *exact weighted model counter* returns $w(R_F)$. An *approximate weighted model counter* relaxes this requirement to some extent: given a *tolerance* $\varepsilon > 0$ and *confidence* $1 - \delta \in (0, 1]$, the value v returned by the counter satisfies $\mathsf{Pr}[\frac{w(R_F)}{1+\varepsilon} \leq v \leq (1+\varepsilon)w(R_F)] \geq 1 - \delta$. A related kind of algorithm is a *weighted-uniform probabilistic generator*, which outputs a witness $w \in R_F$ such that $\mathsf{Pr}[w = y] = w(y)/w(R_F)$ for every $y \in R_F$. An *almost weighted-uniform generator* relaxes this requirement, ensuring that for all $y \in R_F$, we have $\frac{w(y)}{(1+\varepsilon)w(R_F)} \leq \mathsf{Pr}[w = y] \leq \frac{(1+\varepsilon)w(y)}{w(R_F)}$. Probabilistic generators are allowed to occasionally "fail" by not returning a witness (when R_F is non-empty), with the failure probability upper bounded by δ.

3 Related Work

Marrying strong theoretical guarantees with scalable performance is the holy grail of research in the closely related areas of weighted model counting and weighted sampling. The tension between the two objectives is evident from a survey of the literature. Earlier algorithms for weighted model counting can be broadly divided into three categories: those that give strong guarantees but scale poorly in practice, those that give weak guarantees but scale well in practice, and some recent attempts to bridge this gap. Techniques in the first category attempt to compute the weighted model count exactly by enumerating partial solutions (Sang, Beame, and Kautz 2005) or by converting the CNF formula to alternative representations (Darwiche 2004; Choi and Darwiche 2013). Unfortunately, none of these approaches scale to large problem instances. Techniques in the second category employ variational methods, sampling-based methods or other heuristic methods. Variational methods (Wainwright and Jordan 2008; Gogate and Dechter 2011) work extremely well in practice, but do not provide guarantees except in very special cases. Sampling-based methods are usually based on importance sampling (e.g. (Gogate and Dechter 2011)), which provide weak one-sided bounds, or on Markov Chain Monte Carlo (MCMC) sampling (Jerrum and Sinclair 1996; Madras 2002). MCMC sampling is perhaps the most popular technique for both weighted sampling and weighted model counting. Several MCMC algorithms like simulated annealing and the Metropolis-Hastings algorithm have been studied extensively in the literature (Kirkpatrick, Gelatt, and Vecchi 1983; Madras 2002). While MCMC sampling is guaranteed to converge to a target distribution under mild requirements, convergence is often impractically slow (Jerrum and Sinclair 1996). Therefore, practical MCMC sampling-based tools use heuristics that destroy the theoretical guarantees. Several other heuristic techniques that provide weak one-sided bounds have also been proposed in the literature (Gomes, Sabharwal, and Selman 2006).

Recently, Ermon et al. proposed new hashing-based algorithms for approximate weighted model counting and approximate weighted sampling (2013c; 2013a; 2013b; 2014). Their algorithms use random parity constraints as pairwise independent hash functions to partition the set of satisfying assignments of a CNF formula into cells. An oracle is then queried to obtain the maximum weight of an assignment in a randomly chosen cell. By repeating the MPE-queries polynomially many times for randomly chosen cells of appropriate expected sizes, Ermon et al. showed that they can provably compute approximate weighted model counts and also provably achieve approximate weighted sampling. The performance of Ermon et al's algorithms depend crucially on the ability to efficiently answer MPE-queries. Complexity-wise, MPE is an optimization problem and is believed to be much harder than a feasibility query (CNF satisfiability). Furthermore, even the approximation of MPE is known to be NP-hard (Ermon et al. 2013a). The problem is further compounded by the fact that the MPE-queries generated by Ermon et al's algorithms have random parity constraints built into them. Existing MPE-solving techniques work efficiently when the weight distribution of assignments is specified by a graphical model, and the underlying graph has specific structural properties (Park and Darwiche 2004). With random parity constraints, these structural properties are likely to be violated very often. In (Ermon et al. 2013b), it has been argued that an MPE-query-based weighted model-counting algorithm proposed in (Ermon et al. 2013c) is unlikely to scale well to large problem instances. Since MPE-solving is also crucial in the weighted sampling algorithm of (Ermon et al. 2013a), the same criticism applies to that algorithm as well. Several relaxations of the MPE-query-based algorithm proposed in (Ermon et al. 2013c), were therefore discussed in (Ermon et al. 2013b). While these relaxations help reduce the burden of MPE-solving, they also significantly weaken the theoretical guarantees.

In later work, Ermon et al. (2014) showed how the average size of parity constraints in their weighted model counting and weighted sampling algorithms can be reduced using a new class of hash functions. This work, however, still stays within the same paradigm as their earlier work – i.e., it uses MPE-queries and XOR constraints. Although Ermon et al's algorithms provide a 16-factor approximation in theory, in actual experiments, they use relaxations and timeouts of the MPE-solver to get upper and lower bounds on the optimal MPE solution. Unfortunately, these bounds do not come with any guarantees on the factor of approximation. Running the MPE-solver to obtain the optimal value is likely to take significantly longer, and is not attempted in Ermon et al's work. Furthermore, to get an approximation factor less than 16 using Ermon et al's algorithms requires replication of variables. This can significantly increase the running time of their algorithms. In contrast, the performance of our algorithms is much less sensitive to the approximation factor, and our experiments routinely compute 1.75-approximations of weighted model counts.

The algorithms developed in this paper are closely related to two algorithms proposed recently by Chakraborty, Meel and Vardi (2013b; 2014) The first of these (Chakraborty, Meel, and Vardi 2013b) computes the approximate (unweighted) model-count of a CNF formula, while the second algorithm (Chakraborty, Meel, and Vardi 2014) per-

forms near-uniform (unweighted) sampling. Like Ermon et al's algorithms, these algorithms make use of parity constraints as pairwise independent hash functions, and can benefit from the new class of hash functions proposed in (Ermon et al. 2014). Unlike Ermon et al's algorithms, however, Chakraborty et al. use a SAT-solver (NP-oracle) specifically engineered to handle parity constraints efficiently. This allows Chakraborty, Meel, and Vardi's algorithms to scale to much larger problems than those analyzed by Ermon et al., albeit in the unweighted setting. As we show later, this scalability is inherited by our algorithms in the weighted setting as well.

Algorithm 1 WeightMC$(F, \varepsilon, \delta, S, r)$

counter $\leftarrow 0$; $C \leftarrow$ emptyList; $\mathrm{w_{max}} \leftarrow 1$;
pivot $\leftarrow 2 \times \lceil e^{3/2}\left(1+\frac{1}{\varepsilon}\right)^2 \rceil$;
$t \leftarrow \lceil 35 \log_2(3/\delta) \rceil$;
repeat
  $(c, \mathrm{w_{max}}) \leftarrow$ WeightMCCore$(F, S, \mathrm{pivot}, r, \mathrm{w_{max}})$;
  counter $\leftarrow$ counter $+ 1$;
  if $c \neq \perp$ **then**
    AddToList$(C, c \cdot \mathrm{w_{max}})$;
until (counter $< t$)
finalCount $\leftarrow$ FindMedian(C);
return finalCount;

4 Algorithms

We now present algorithms for approximate weighted model counting and approximate weighted sampling, assuming a small bounded tilt and a black-box weight function. Recalling that the tilt concerns weights of only satisfying assignments, our assumption about it being bounded by a small number is reasonable in several practical situations. For example, when performing probabilistic inference with evidence by reduction to weighted model counting (Chavira and Darwiche 2008), every satisfying assignment of the CNF formula corresponds to an assignment of values to variables in the underlying probabilistic graphical model that is consistent with the evidence. Furthermore, the weight of a satisfying assignment is the joint probability of the corresponding assignment of variables in the probabilistic graphical model. A large tilt would therefore mean existence of two assignments that are consistent with the same evidence, but of which one is overwhelmingly more likely than the other. In several real-world situations, this is considered unlikely given that numerical conditional probability values are often obtained from human experts providing qualitative and rough quantitative data (see, e.g. Sec 8.3 of (Dıez and Druzdzel 2006)).

The algorithms presented in this section require an upper bound for the tilt ρ as part of their input. It is worth noting that although tighter upper bounds improve performance, the algorithms are sound with respect to any upper bound estimate. While an algorithmic solution to the estimation of upper bounds for ρ is beyond the scope of this work, such an estimate can often be easily obtained from the designers of probabilistic models. It is often easier for designers to estimate an upper bound for ρ than to accurately estimate w_{max}, since the former does not require precise knowledge of the probabilities of all models.

Algorithm 2 WeightMCCore$(F, S, \mathrm{pivot}, r, \mathrm{w_{max}})$

$(Y, \mathrm{w_{max}}) \leftarrow$
  BoundedWeightSAT$(F, \mathrm{pivot}, r, \mathrm{w_{max}}, S)$;
if $(w(Y)\,/\mathrm{w_{max}} \leq \mathrm{pivot})$ **then**
  return $w(Y)$;
else
  $i \leftarrow 0$;
  repeat
    $i \leftarrow i + 1$;
    Choose h at random from $H_{xor}(|S|, i, 3)$;
    Choose α at random from $\{0,1\}^i$;
    $(Y, \mathrm{w_{max}}) \leftarrow$ BoundedWeightSAT$(F \wedge (h(x_1, \ldots x_{|S|}) = \alpha), \mathrm{pivot}, r, \mathrm{w_{max}}, S)$;
  until $((0 < w(Y)\,/\mathrm{w_{max}} \leq \mathrm{pivot})$ or $(i = n))$
  if $((w(Y)\,/\mathrm{w_{max}} > \mathrm{pivot})$ **or** $(w(Y) = 0))$ **then return** $(\perp, \mathrm{w_{max}})$;
  else return $(\frac{w(Y)\cdot 2^{i-1}}{\mathrm{w_{max}}}, \mathrm{w_{max}})$;

Algorithm 3 BoundedWeightSAT$(F, \mathrm{pivot}, r, \mathrm{w_{max}}, S)$

$\mathrm{w_{min}} \leftarrow \mathrm{w_{max}}/r$; $\mathrm{w_{total}} \leftarrow 0$; $Y = \{\}$;
repeat
  $y \leftarrow$ SolveSAT(F);
  if $(y =$ UNSAT$)$ **then**
    break;
  $Y = Y \cup y$;
  $F =$ AddBlockClause$(F, y|_S)$;
  $\mathrm{w_{total}} \leftarrow \mathrm{w_{total}} + w(y)$;
  $\mathrm{w_{min}} \leftarrow min(\mathrm{w_{min}}, w(y))$;
until $(\mathrm{w_{total}}/(\mathrm{w_{min}} \cdot r) > \mathrm{pivot})$;
return $(Y, \mathrm{w_{min}} \cdot r)$;

Our weighted model counting algorithm, called WeightMC, is best viewed as an adaptation of the ApproxMC algorithm proposed by Chakraborty, Meel and Vardi (2013b) for approximate unweighted model counting. Similarly, our weighted sampling algorithm, called WeightGen, can be viewed as an adaptation of the the UniGen algorithm (Chakraborty, Meel, and Vardi 2014), originally proposed for near-uniform unweighted sampling. The key idea in both ApproxMC and UniGen is to partition the set of satisfying assignments into "cells" containing roughly equal numbers of satisfying assignments, using a random hash function from the family $H_{xor}(n, m, 3)$. A random cell is then chosen and inspected to see if the number of satisfying assignments in it is smaller than a pre-computed threshold. This threshold, in turn, depends on the desired approximation factor or tolerance ε. If the chosen cell is small enough, UniGen samples uniformly from the chosen small cell to obtain a near-uniformly generated satisfying assignment. ApproxMC multiplies the number of satisfying assignments in the cell by a suitable scaling

Algorithm 4 WeightGen(F, ε, r, S)

/*Assume $\varepsilon > 6.84$ */

1: $\mathrm{w_{max}} \leftarrow 1$; Samples $= \{\}$;
2: $(\kappa, \mathrm{pivot}) \leftarrow$ ComputeKappaPivot(ε);
3: $\mathrm{hiThresh} \leftarrow 1 + \sqrt{2}(1+\kappa)\mathrm{pivot}$;
4: $\mathrm{loThresh} \leftarrow \frac{1}{\sqrt{2}(1+\kappa)}\mathrm{pivot}$;
5: $(Y, \mathrm{w_{max}}) \leftarrow$ BoundedWeightSAT$(F, \mathrm{hiThresh}, r, \mathrm{w_{max}}, S)$;
6: **if** $(w(Y)/\mathrm{w_{max}} \leq \mathrm{hiThresh})$ **then**
7: Choose y weighted-uniformly at random from Y;
8: **return** y;
9: **else**
10: $(C, \mathrm{w_{max}}) \leftarrow$ WeightMC$(F, 0.8, 0.2)$;
11: $q \leftarrow \lceil \log C - \log \mathrm{w_{max}} + \log 1.8 - \log \mathrm{pivot} \rceil$;
12: $i \leftarrow q - 4$;
13: **repeat**
14: $i \leftarrow i + 1$;
15: Choose h at random from $H_{xor}(|S|, i, 3)$;
16: Choose α at random from $\{0,1\}^i$;
17: $(Y, \mathrm{w_{max}}) \leftarrow$ BoundedWeightSAT$(F \wedge (h(x_1, \ldots x_{|S|}) = \alpha), \mathrm{hiThresh}, r, \mathrm{w_{max}}, S)$;
18: $W \leftarrow w(Y)/\mathrm{w_{max}}$
19: **until** $(\mathrm{loThresh} \leq W \leq \mathrm{hiThresh})$ or $(i = q)$
20: **if** $((W > \mathrm{hiThresh})$ or $(W < \mathrm{loThresh}))$ **then return** $\perp$
21: **else** Choose y weighted-uniformly at random from Y; **return** y;

Algorithm 5 ComputeKappaPivot(ε)

1: Find $\kappa \in [0, 1)$ such that $\varepsilon = (1+\kappa)(7.55 + \frac{0.29}{(1-\kappa)^2}) - 1$
2: $\mathrm{pivot} \leftarrow \lceil 4.03 \left(1 + \frac{1}{\kappa}\right)^2 \rceil$; **return** (κ, pivot)

factor to obtain an estimate of the model count. ApproxMC is then repeated a number of times (depending on the desired confidence $1 - \delta$) and the median of the computed counts determined to obtain the final approximate model count. For weighted model counting and sampling, the primary modification that needs to be made to ApproxMC and UniGen is that instead of requiring "cells" to have roughly equal numbers of satisfying assignments, we now require them to have roughly equal weights of satisfying assignments.

A randomly chosen hash function from $H_{xor}(n, m, 3)$ consists of m XOR constraints, each of which has expected size $n/2$. Although ApproxMC and UniGen were shown to scale to a few thousand variables, their performance erodes rapidly beyond that point. It has recently been shown in (Chakraborty, Meel, and Vardi 2014) that by using random parity constraints on the independent support of a formula (which can be orders of magnitude smaller than the complete support), we can significantly reduce the size of XOR constraints. We use this idea in our work. For all our benchmark problems, obtaining the independent support of the CNF formulae has been easy, once we examine the domain from which the problem originated.

Both WeightMC and WeightGen assume access to a subroutine called BoundedWeightSAT that takes as inputs a CNF formula F, a "pivot", an upper bound r on the tilt and an upper bound $\mathrm{w_{max}}$ on the maximum weight of a satisfying assignment in the independent support set S. BoundedWeightSAT returns a set of satisfying assignments of F such that the total weight of the returned assignments scaled by $1/\mathrm{w_{max}}$ exceeds the "pivot". Since all weights are assumed to be in $(0, 1]$, the upper bound of $\mathrm{w_{max}}$ is set to 1 in the initial invocation of BoundedWeightSAT. Subsequently, BoundedWeightSAT returns a refined upper bound of $\mathrm{w_{max}}$ from the knowledge of r (upper bound on the tilt), and the minimum weight of all satisfying assignments seen so far. Every invocation of BoundedWeightSAT also accesses an NP-oracle, called SolveSAT, which can decide SAT. In addition, it accesses a subroutine AddBlockClause that takes as inputs a formula F and a projected assignment $\sigma|_S$, computes a blocking clause for $\sigma|_S$, and returns the formula F' obtained by conjoining F with the blocking clause.

4.1 WeightMC Algorithm

The pseudo-code for WeightMC is shown in Algorithm 1. The algorithm takes as inputs a CNF formula F, tolerance $\varepsilon \in (0, 1)$, confidence parameter $\delta \in (0, 1)$, independent support S, and upper bound r on the tilt, and returns an approximate weighted model count. WeightMC invokes an auxiliary procedure WeightMCCore that computes an approximate weighted model count by randomly partitioning the space of satisfying assignments using hash functions from the family $H_{xor}(|S|, m, 3)$. WeightMC first computes two parameters: pivot, which quantifies the size of a "small" cell, and t, which determines the number of invocations of WeightMC. The particular choice of expressions to compute these parameters is motivated by technical reasons. After invoking WeightMCCore sufficiently many times, WeightMC returns the median of the non-$\perp$ counts returned by WeightMCCore.

The pseudo-code for subroutine WeightMCCore is presented in Algorithm 2. WeightMCCore takes as inputs a CNF formula F, independent support S, parameter pivot to quantify the size of "small" cells, upper bound r on the tilt, and the current upper bound on w_{max}, and returns an approximate weighted model count and a revised upper bound on w_{max}. WeightMCCore first handles the easy case of the total weighted count of F being less than pivot in lines 1–4. Otherwise, in every iteration of the loop in lines 7–12, WeightMCCore randomly partitions the solution space of F using $H_{xor}(|S|, i, 3)$ until a randomly chosen cell is "small" i.e. the total weighted count of the cell is less than pivot. We also refine the estimate for w_{max} in every iteration of the loop in lines 7–12 using the minimum weight of solutions seen so far (computed by BoundedWeightSAT). In the event a chosen cell is "small", WeightMCCore multiplies the weighted count of the cell by the total number of cells to obtain the estimated total weighted count. The estimated total weighted count along with a refined estimate of w_{max} is finally returned in line 14.

Theorem 1. *Given a propositional formula F, $\varepsilon \in (0, 1)$, $\delta \in (0, 1)$, independent support S, and upper bound r*

of the tilt, suppose $\mathsf{WeightMC}(F, \varepsilon, \delta, S, r)$ *returns* c*. Then* $\Pr\left[(1+\varepsilon)^{-1} \cdot w(R_F)) \leq c \leq (1+\varepsilon) \cdot w(R_F))\right] \geq 1-\delta$.

Theorem 2. *Given an oracle (*SolveSAT*) for* SAT, $\mathsf{WeightMC}(F, \varepsilon, \delta, S, r)$ *runs in time polynomial in* $\log_2(1/\delta), r, |F|$ *and* $1/\varepsilon$ *relative to the oracle.*

The proofs of Theorem 1 and 2 can be found in (Chakraborty et al. 2014).

4.2 WeightGen Algorithm

The pseudo-code for WeightGen is presented in Algorithm 4. WeightGen takes as inputs a CNF formula F, tolerance $\varepsilon > 6.84$, upper bound r of the tilt, and independent support S, and returns a random (approximately weighted-uniform) satisfying assignment of F.

WeightGen can be viewed as an adaptation of UniGen (Chakraborty, Meel, and Vardi 2014) to the weighted domain. It first computes two parameters, κ and pivot, and then uses them to compute $\mathrm{hiThresh}$ and $\mathrm{loThresh}$, which quantify the size of a "small" cell. The easy case of the weighted count being less than $\mathrm{hiThresh}$ is handled in lines 6–9. Otherwise, WeightMC is called to estimate the weighted model count. This is then used to estimate a range of candidate values for i, where a random hash function from $H_{xor}(|S|, i, 3)$ must be used to randomly partition the solution space. The choice of parameters for the invocation of WeightMC is motivated by technical reasons. The loop in lines 13–19 terminates when a small cell is found; a sample is then picked weighted-uniformly at random from that cell. Otherwise, the algorithm reports a failure in line 20.

Theorem 3. *Given a CNF formula* F*, tolerance* $\varepsilon > 6.84$*, upper bound* r *of the tilt, and independent support* S*, for every* $y \in R_F$ *we have* $\frac{w(y)}{(1+\varepsilon)w(R_F)} \leq \Pr\left[\mathsf{WeightGen}(F, \varepsilon, r, X) = y\right] \leq (1+\varepsilon)\frac{w(y)}{w(R_F)}$*. Also,* WeightGen *succeeds (i.e. does not return* $\perp$*) with probability at least* 0.52*.*

Theorem 4. *Given an oracle (*SolveSAT*) for* SAT, $\mathsf{WeightGen}(F, \varepsilon, r, S)$ *runs in time polynomial in* $r, |F|$ *and* $1/\varepsilon$ *relative to the oracle.*

The proofs of Theorem 3 and 4 can be found in (Chakraborty et al. 2014).

Implementation Details: In our implementations of WeightGen and WeightMC, BoundedWeightSAT is implemented using CryptoMiniSAT (Cry), a SAT solver that handles XOR clauses efficiently. CryptoMiniSAT uses *blocking clauses* to prevent already generated witnesses from being generated again. Since the independent support of F determines every satisfying assignment of F, blocking clauses can be restricted to only variables in the set S. We implemented this optimization in CryptoMiniSAT, leading to significant improvements in performance. We used "random_device" implemented in C++11 as a source of pseudo-random numbers to make random choices in WeightGen and WeightMC.

4.3 Generalization

We have so far restricted S to be an independent support of F in order to ensure 3-wise independence of $H_{xor}(|S|, m, 3)$ over the entire solution space R_F. Indeed, this is crucial for proving the theorems presented in this section. However, similar to (Chakraborty, Meel, and Vardi 2014), our results can be generalized to arbitrary subsets S of the support of F. For an arbitrary S, Theorems 3 and 1 generalize to weighted sampling and weighted counting over the solution space projected onto S. To illustrate the projection of the solution space (R_F) over S, consider $F = (a \vee b)$ and $S = \{b\}$. Then the projection of R_F over S, denoted by $R_{F|S}$, is $\{\{0\}, \{1\}\}$. This generalization allows our algorithms to extend to formulas of the form $\exists(\cdot)F$ without incurring any additional cost. We discuss this generalization in detail in the full version of our paper (Chakraborty et al. 2014).

5 Experimental Results

To evaluate the performance of WeightGen and WeightMC, we built prototype implementations and conducted an extensive set of experiments. The suite of benchmarks was made up of problems arising from various practical domains as well as problems of theoretical interest. Specifically, we used bit-level unweighted versions of constraints arising from grid networks, plan recognition, DQMR networks, bounded model checking of circuits, bit-blasted versions of SMT-LIB (SMT) benchmarks, and ISCAS89 (Brglez, Bryan, and Kozminski 1989) circuits with parity conditions on randomly chosen subsets of outputs and next-state variables (Sang, Beame, and Kautz 2005; John and Chakraborty 2011). While our algorithms are agnostic to the weight oracle, other tools that we used for comparison require the weight of an assignment to be the product of the weights of its literals. Consequently, to create weighted problems with tilt bounded by some number r, we randomly selected $m = \max(15, n/100)$ of the n variables in a problem instance and assigned them the weight w, where $(w/(1-w))^m = r$, and assigned their negations the weight $1-w$. All other literals were assigned the weight 1. To demonstrate the agnostic nature of our algorithms with respect to the weight oracle, we also evaluated WeightMC and WeightGen with a non-factored weight representation. However, we do not present these results here due to lack of space and also due to the unavailability of a competing tool with which to compare our results on benchmarks with non-factored weights. Details of these experiments are presented in the full version of our paper. Unless mentioned otherwise, our experiments for WeightMC reported in this paper used $r = 5$, $\varepsilon = 0.8$, and $\delta = 0.2$, while our experiments for WeightGen used $r = 5$ and $\varepsilon = 16$.

To facilitate performing multiple experiments in parallel, we used a high performance cluster, with each experiment running on its own core. Each node of the cluster had two quad-core Intel Xeon processors with 4GB of main memory. We used 2500 seconds as the timeout of each invocation of BoundedWeightSAT and 20 hours as the overall timeout for WeightGen and WeightMC. If an

Table 1: WeightMC, SDD, and WeightGen runtimes in seconds.

Benchmark	vars	#clas	Weight-MC	SDD	Weight-Gen
or-50	100	266	17	0.39	0.17
or-60	120	323	115	0.51	1.5
s526a_3_2	366	944	115	13	1.27
s526_15_7	452	1303	119	41	2.74
s953a_3_2	515	1297	12994	357	33
s1196a_15_7	777	2165	2559	1809	30
s1238a_7_4	704	1926	2885	mem	45
Squaring1	891	2839	18276	mem	164
Squaring7	1628	5837	21982	mem	175
LoginService2	11511	41411	207	mem	4.81
Sort	12125	49611	39641	T	258
Karatsuba	19594	82417	4352	T	326
EnqueueSeq	16466	58515	5763	mem	96
TreeMax	24859	103762	41	mem	4.2
LLReverse	63797	257657	1465	mem	298

invocation of BoundedWeightSAT timed out in line 11 of WeightMCCore or line 17 of WeightGen, we repeated the execution of the corresponding loops without incrementing the variable i in both algorithms. With this setup, WeightMC and WeightGen were able to successfully return weighted counts and generate weighted random solutions for formulas with almost $64,000$ variables.

We compared the performance of WeightMC with the SDD Package (sdd), a state-of-the-art tool which can perform exact weighted model counting by compiling CNF formulae into Sentential Decision Diagrams (Choi and Darwiche 2013). We also tried to compare our tools against Cachet, WISH, and PAWS, but the current versions of the tools made available to us were broken and we are yet, at the time of submission, to receive working tools. If we are granted access to working tools in future, we will update the full version of our paper with the corresponding comparisons. Our results are shown in Table 1, where column 1 lists the benchmarks and columns 2 and 3 give the number of variables and clauses for each benchmark. Column 4 lists the time taken by WeightMC, while column 5 lists the time taken by SDD. We also measured the time taken by WeightGen to generate samples, which we discuss later in this section, and list it in column 6. "T" and "mem" indicate that an experiment exceeded our imposed 20-hour and 4GB-memory limits, respectively. While SDD was generally superior for small problems, WeightMC was significantly faster for all benchmarks with more than 1,000 variables.

To evaluate the quality of the approximate counts returned by WeightMC, we computed exact weighted model counts using the SDD tool for a subset of our benchmarks. Figure 1 shows the counts returned by WeightMC, and the exact counts from SDD scaled up and down by $1 + \varepsilon$. The weighted model counts are represented on the y-axis, while the x-axis represents benchmarks arranged in increasing order of counts. We observe that the weighted counts returned by WeightMC always lie within the tolerance of the exact counts. Over all of the benchmarks, the L_2 norm of the relative error was 0.014, demonstrating that in practice WeightMC is substantially more accurate than the theoretical guarantees provided by Theorem 3.

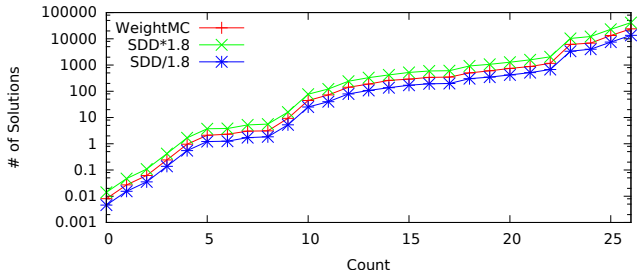

Figure 1: Quality of counts computed by WeightMC. The benchmarks are arranged in increasing order of weighted model counts.

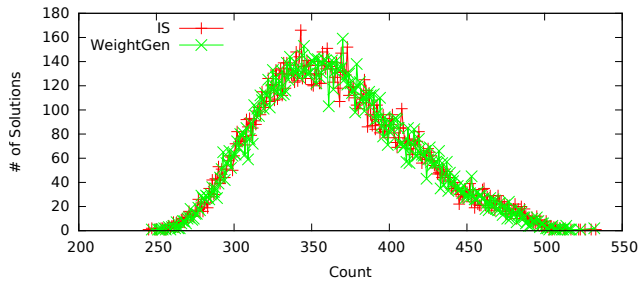

Figure 2: Uniformity comparison for case110

In another experiment, we studied the effect of different values of the tilt bound r on the runtime of WeightMC. Runtime as a function of r is shown for several benchmarks in Figure 3, where times have been normalized so that at the lowest tilt ($r = 1$) each benchmark took one (normalized) time unit. Each runtime is an average over 15 runs on the same benchmark. The theoretical linear dependence on the tilt shown in Theorem 2 can be seen to roughly occur in practice.

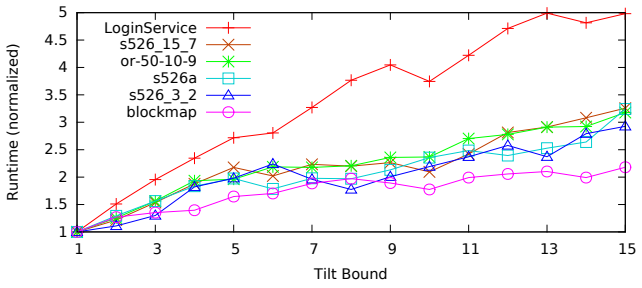

Figure 3: Runtime of WeightMC as a function of tilt bound.

Since a probabilistic generator is likely to be invoked many times with the same formula and weights, it is useful to perform the counting on line 10 of WeightGen only once, and reuse the result for every sample. Reflecting this,

column 6 in Table 1 lists the time, averaged over a large number of runs, taken by WeightGen to generate one sample given that the weighted model count on line 10 has already been found. It is clear from Table 1 that WeightGen scales to formulas with thousands of variables.

To measure the accuracy of WeightGen, we implemented an *Ideal Sampler*, henceforth called IS, and compared the distributions generated by WeightGen and IS for a representative benchmark. Given a CNF formula F, IS first generates all the satisfying assignments, then computes their weights and uses these to sample the ideal distribution. We generated a large number N ($= 6 \times 10^6$) of sample witnesses using both IS and WeightGen. In each case, the number of times various witnesses were generated was recorded, yielding a distribution of the counts. Figure 2 shows the distributions generated by WeightGen and IS for one of our benchmarks (case110) with 16,384 solutions. The almost perfect match between the distribution generated by IS and WeightGen held also for other benchmarks. Thus, as was the case for WeightMC, the accuracy of WeightGen is better in practice than that established by Theorem 3.

6 White-Box Weight Functions

As noted above, the runtime of WeightMC is proportional to the tilt of the weight function, which means that the algorithm becomes impractical when the tilt is large. If the assignment weights are given by a known polynomial-time-computable function instead of an oracle, we can do better. We abuse notation slightly and denote this weight function by $w(X)$, where X is the set of support variables of the Boolean formula F. The essential idea is to partition the set of satisfying assignments into regions within which the tilt is small. Defining $R_F(a, b) = \{\sigma \in R_F | a < w(\sigma) \leq b\}$, we have $w(R_F) = w(R_F(w_{min}, w_{max}))$. If we use a partition of the form $R_F(w_{min}, w_{max}) = R_F(w_{max}/2, w_{max}) \cup R_F(w_{max}/4, w_{max}/2) \cup \cdots \cup R_F(w_{max}/2^N, w_{max}/2^{N-1})$, where $w_{max}/2^N \leq w_{min}$, then in each partition region the tilt is at most 2. Note that we do not need to know the actual values of w_{min} and w_{max}: any bounds L and H such that $0 < L \leq w_{min}$ and $w_{max} \leq H$ will do (although if the bounds are too loose, we may partition R_F into more regions than necessary). If assignment weights are poly-time computable, we can add to F a constraint that eliminates all assignments not in a particular region. So we can run WeightMC on each region in turn, passing 2 as the upper bound on the tilt, and sum the results to get $w(R_F)$. This idea is implemented in PartitionedWeightMC (Algorithm 6).

Algorithm 6 PartitionedWeightMC$(F, \varepsilon, \delta, S, L, H)$

1: $N \leftarrow \lceil \log_2 H/L \rceil + 1$; $\delta' \leftarrow \delta/N$; $c \leftarrow 0$
2: **for all** $1 \leq m \leq N$ **do**
3: $\quad G \leftarrow F \wedge (H/2^m < w(X) \leq H/2^{m-1})$
4: $\quad d \leftarrow$ WeightMC$(G, \varepsilon, \delta', S, 2)$
5: $\quad$ **if** $(d = \perp)$ **then return** $\perp$
6: $\quad c \leftarrow c + d$
7: **return** c

The correctness and runtime of PartitionedWeightMC are established by the following theorems, whose proofs are presented in (Chakraborty et al. 2014).

Theorem 5. *If* PartitionedWeightMC$(F, \varepsilon, \delta, S, L, H)$ *returns* c *(and all arguments are in the required ranges), then* $\Pr\left[(1+\varepsilon)^{-1} w(R_F) \leq c \leq (1+\varepsilon) w(R_F))\right] \geq 1 - \delta$.

Theorem 6. *With access to an* NP *oracle, the runtime of* PartitionedWeightMC$(F, \varepsilon, \delta, S, L, H)$ *is polynomial in* $|F|$, $1/\varepsilon$, $\log(1/\delta)$, *and* $\log r = \log(H/L)$.

The reduction of the runtime's dependence on the tilt bound r from linear to logarithmic can be a substantial saving. If the assignment weights are products of literal weights, as is the case in many applications, the best *a priori* bound on the tilt ρ given only the literal weights is exponential in n. Thus, unless the structure of the problem allows a better bound on ρ to be used, WeightMC will not be practical. In this situation PartitionedWeightMC can be used to maintain polynomial runtime.

When implementing PartitionedWeightMC in practice the handling of the weight constraint $H/2^m < w(X) \leq H/2^{m-1}$ is critical to efficiency. If assignment weights are sums of literal weights, or equivalently products of literal weights (we just take logarithms), then the weight constraint is a pseudo-Boolean constraint. In this case we may replace the SAT-solver used by WeightMC with a pseudo-Poolean satisfiability (PBS) solver. While a number of PBS-solvers exist (Manquinho and Roussel 2012), none have the specialized handling of XOR clauses that is critical in making WeightMC practical. The design of such solvers is a clear direction for future work. We also note that the choice of 2 as the tilt bound for each region is arbitrary, and the value may be adjusted depending on the application: larger values will decrease the number of regions, but increase the difficulty of counting within each region. Finally, note that the same partitioning idea can be used to reduce WeightGen's dependence on r to be logarithmic.

7 Conclusion

In this paper, we considered approximate approaches to the twin problems of distribution-aware sampling and weighted model counting for SAT. For approximation techniques that provide strong theoretical two-way bounds, a major limitation is the reliance on potentially-expensive most probable explanation (MPE) queries. We identified a novel parameter, *tilt*, to categorize weighted counting and sampling problems for SAT. We showed how to remove the reliance on MPE-queries, while retaining strong theoretical guarantees. First, we provided model counting and sampling algorithms that work with a black-box model of giving weights to assignments, requiring access only to an NP-oracle, which is efficient for small tilt values. Experimental results demonstrate the effectiveness of this approach in practice. Second, we provided an alternative approach that promises to be efficient for large tilt values, requiring, however, a white-box weight model and access to a pseudo-Boolean solver. As a next step, we plan to empirically evaluate this latter approach using pseudo-Boolean solvers designed to handle parity constraints efficiently.

Acknowledgments: The authors would like to thank Armando Solar-Lezama for generously providing a large set of benchmarks. We thank the anonymous reviewers for their detailed, insightful comments and suggestions that helped improve this manuscript significantly.

References

Bacchus, F.; Dalmao, S.; and Pitassi, T. 2003. Algorithms and complexity results for #SAT and Bayesian inference. In *Proc. of FOCS*, 340–351.

Bellare, M.; Goldreich, O.; and Petrank, E. 1998. Uniform generation of NP-witnesses using an NP-oracle. *Information and Computation* 163(2):510–526.

Brglez, F.; Bryan, D.; and Kozminski, K. 1989. Combinational profiles of sequential benchmark circuits. In *Proc. of ISCAS*, 1929–1934.

Chakraborty, S.; Fremont, D. J.; Meel, K. S.; Seshia, S.; and Vardi, M. Y. 2014. Distribution-aware sampling and weighted model counting for SAT (Technical Report). `http://arxiv.org/abs/1403.6246`.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013a. A scalable and nearly uniform generator of SAT witnesses. In *Proc. of CAV*, 608–623.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013b. A scalable approximate model counter. In *Proc. of CP*, 200–216.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2014. Balancing scalability and uniformity in SAT-witness generator. In *Proc. of DAC*.

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

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

CryptoMiniSAT. `http://www.msoos.org/cryptominisat2/`.

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

Dıez, F. J., and Druzdzel, M. J. 2006. Canonical probabilistic models for knowledge engineering. Technical report, CISIAD-06-01, UNED, Madrid, Spain.

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

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013a. Embed and project: Discrete sampling with universal hashing. In *Proc of NIPS*, 2085–2093.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013b. Optimization with parity constraints: From binary codes to discrete integration. In *Proc. of UAI*, 202–211.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013c. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proc. of ICML*, 334–342.

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

Gogate, V., and Dechter, R. 2011. Samplesearch: Importance sampling in presence of determinism. *Artificial Intelligence* 175(2):694–729.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Model counting: A new strategy for obtaining good bounds. In *Proc. of AAAI*, 54–61.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2007. Near uniform sampling of combinatorial spaces using XOR constraints. In *Proc. of NIPS*, 670–676.

Jerrum, M., and Sinclair, A. 1996. The Markov chain Monte Carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems* 482–520.

Jerrum, M.; Valiant, L.; and Vazirani, V. 1986. Random generation of combinatorial structures from a uniform distribution. *TCS* 43(2-3):169–188.

John, A., and Chakraborty, S. 2011. A quantifier elimination algorithm for linear modular equations and disequations. In *Proc. of CAV*, 486–503.

Karp, R.; Luby, M.; and Madras, N. 1989. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms* 10(3):429–448.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220(4598):671–680.

Madras, N., and Piccioni, M. 1999. Importance sampling for families of distributions. *Annals of applied probability* 9(4):1202–1225.

Madras, N. 2002. *Lectures on Monte Carlo Methods*, volume 16 of *Fields Institute Monographs*. AMS.

Manquinho, V., and Roussel, O. 2012. Seventh pseudo-Boolean competition. `http://www.cril.univ-artois.fr/PB12/`.

Naveh, Y.; Rimon, M.; Jaeger, I.; Katz, Y.; Vinov, M.; Marcus, E.; and Shurek, G. 2006. Constraint-based random stimuli generation for hardware verification. In *Proc. of IAAI*, 1720–1727.

Park, J. D., and Darwiche, A. 2004. Complexity results and approximation strategies for map explanations. *J. Artif. Intell. Res.* 21:101–133.

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

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

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

The SDD package. `http://reasoning.cs.ucla.edu/sdd/`.

SMTLib. `http://goedel.cs.uiowa.edu/smtlib/`.

Toda, S. 1989. On the computational power of PP and (+)P. In *Proc. of FOCS*, 514–519.

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

Wainwright, M. J., and Jordan, M. I. 2008. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning* 1(1-2):1–305.

Xue, Y.; Choi, A.; and Darwiche, A. 2012. Basing decisions on sentences in decision diagrams. In *Proc. of AAAI*, 842–849.