The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence
Beyond NP: Technical Report WS-16-05

Constrained Sampling and Counting: Universal Hashing Meets SAT Solving*

Kuldeep S. Meel,[1] Moshe Y. Vardi,[1] Supratik Chakraborty,[2] Daniel J. Fremont,[3] Sanjit A. Seshia,[3] Dror Fried,[1] Alexander Ivrii,[4] Sharad Malik[5]

[1]Department of Computer Science, Rice University [2]Indian Institute of Technology, Bombay
[3]University of California, Berkeley [4]IBM Research, Haifa
[5]Princeton University

Abstract

Constrained sampling and counting are two fundamental problems in artificial intelligence with a diverse range of applications, spanning probabilistic reasoning and planning to constrained-random verification. While the theory of these problems was thoroughly investigated in the 1980s, prior work either did not scale to industrial size instances or gave up correctness guarantees to achieve scalability. Recently, we proposed a novel approach that combines universal hashing and SAT solving and scales to formulas with hundreds of thousands of variables without giving up correctness guarantees. This paper provides an overview of the key ingredients of the approach and discusses challenges that need to be overcome to handle larger real-world instances.

1 Introduction

Constrained sampling and counting are two fundamental problems in artificial intelligence. In constrained sampling, the task is to sample randomly from the set of solutions of input constraints while the problem of constrained counting is to count the number of solutions. Both problems have numerous applications, including in probabilistic reasoning, machine learning, planning, statistical physics, inexact computing, and constrained-random verification (Bacchus, Dalmao, and Pitassi 2003; Jerrum and Sinclair 1996; Naveh et al. 2006; Roth 1996). For example, probabilistic inference over graphical models can be reduced to constrained counting for propositional formulas (Cooper 1990; Roth 1996). In addition, approximate probabilistic reasoning relies heavily on sampling from high-dimensional probabilistic spaces encoded as sets of constraints (Ermon et al. 2013a; Jerrum and Sinclair 1996). Both constrained sampling and counting can be viewed as aspects of one of the most fundamental problems in artificial intelligence: exploring the structure of the solution space of a set of constraints (Russell and Norvig 2009).

Constrained sampling and counting are known to be computationally hard (Valiant 1979; Jerrum, Valiant, and Vazirani 1986; Toda 1989). To bypass these hardness results, approximate versions of the problems have been investigated. Despite strong theoretical and practical interest in approximation techniques over the years, there is still an immense gap between theory and practice in this area. Theoretical algorithms offer guarantees on the quality of approximation, but do not scale in practice, whereas practical tools achieve scalability at the cost of offering weaker or no guarantees.

Our recent work in constrained sampling and counting (Chakraborty et al. 2014; Chakraborty, Meel, and Vardi 2013a; 2013b; 2014; Chakraborty et al. 2014; 2015a; 2015b; Ivrii et al. 2015), has yielded significant progress in this area. By combining the ideas of using SAT solving as an oracle and the reduction of the solution space via universal hashing, we have developed *highly scalable* algorithms that offer *rigorous* approximation guarantees. Thus, we were able to take the first step in bridging the gap between theory and practice in approximate constrained sampling and counting. A key enabling factor has been the tremendous progress over the past two decades in propositional satisfiability (SAT) solving, which makes SAT solving usable as an algorithmic building block in practical algorithms.

In this paper, we provide an overview of hashing-based sampling and counting techniques and put them in the context of related work. We then highlight key insights that have allowed us to further push the scalability envelope of these algorithms. Finally, we discuss challenges that still need to be overcome before hashing-based sampling and counting algorithms can be applied to large scale real-world instances.

The rest of the paper is organized as follows. We introduce notation and preliminaries in Section 2 and discuss related work in Section 3. We discuss key enabling algorithmic techniques for the hashing-based approach in Section 4, followed by an overview of the sampling and counting algorithms themselves in Section 5. We describe recent advances in pushing forward the scalability of these algorithms in Section 6, and finally conclude in Section 7.

2 Preliminaries

Let F denote 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. We also use $\mathsf{Vars}(F)$ to denote 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 $\sigma|_S$ for the projection of σ onto S. A *satisfying assign-*

*The order of authors is based on the number of published works contributed to the project and ties have been broken alphabetically.

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ment 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 and the projection of R_F onto S by $R_{F|S}$. For notational convenience, whenever the formula F is clear from the context, we omit mentioning it.

Let $\mathcal{I} \subseteq X$ be a subset of the support such that if two satisfying assignments σ_1 and σ_2 agree on $\mathcal{I}$, then $\sigma_1 = \sigma_2$. In other words, in every satisfying assignment, the truth values of variables in $\mathcal{I}$ uniquely determine the truth value of every variable in $X \setminus \mathcal{I}$. The set $\mathcal{I}$ is called an *independent support* of F, and $\mathcal{D} = X \setminus \mathcal{I}$ is a *dependent support*. Note that there is a one-to-one correspondence between R_F and $R_{F|\mathcal{I}}$. There may be more than one independent support: $(a \vee \neg b) \wedge (\neg a \vee b)$ has three, namely $\{a\}$, $\{b\}$ and $\{a, b\}$. Clearly, if $\mathcal{I}$ is an independent support of F, so is every superset of $\mathcal{I}$.

The *constrained-sampling problem* is to sample randomly from R_F, given F. A *probabilistic generator* is a probabilistic algorithm that generates from F a random solution in R_F. Let $\mathsf{Pr}[E]$ denote the probability of an event E. A *uniform generator* $\mathcal{G}^u(\cdot)$ is a probabilistic generator that, given F, guarantees $\mathsf{Pr}[\mathcal{G}^u(F) = y] = 1/|R_F|$, for every $y \in R_F$. An *almost-uniform generator* $\mathcal{G}^{au}(\cdot, \cdot)$ guarantees that for every $y \in R_F$, we have $\frac{1}{(1+\varepsilon)|R_F|} \leq \mathsf{Pr}[\mathcal{G}^{au}(F, \varepsilon) = y] \leq \frac{1+\varepsilon}{|R_F|}$, where $\varepsilon > 0$ is a specified *tolerance*. Probabilistic generators are allowed to occasionally "fail" in the sense that no solution may be returned even if R_F is non-empty. The failure probability for such generators must be bounded by a constant strictly less than 1.

The *constrained-counting problem* is to compute the size of the set R_F for a given CNF formula F. An *approximate counter* is a probabilistic algorithm $\mathsf{ApproxCount}(\cdot, \cdot, \cdot)$ that, given a formula F, a tolerance $\varepsilon > 0$, and a confidence $1 - \delta \in (0, 1]$, guarantees that $\mathsf{Pr}\Big[|R_F|/(1+\varepsilon) \leq \mathsf{ApproxCount}(F, \varepsilon, 1-\delta) \leq (1+\varepsilon)|R_F|\Big] \geq 1 - \delta$.

The type of hash functions used in the hashing-based approach to sampling and counting are r-*universal* hash functions. For positive integers n, m, and r, we write $H(n, m, r)$ to denote a family of r-universal hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$. We use $h \xleftarrow{R} H(n, m, r)$ to denote the probability space obtained by choosing a hash function h uniformly at random from $H(n, m, r)$. The property of r-universality guarantees that for all $\alpha_1, \ldots, \alpha_r \in \{0,1\}^m$ and all distinct $y_1, \ldots, y_r \in \{0,1\}^n$, $\mathsf{Pr}\Big[\bigwedge_{i=1}^{r} h(y_i) = \alpha_i : h \xleftarrow{R} H(n, m, r)\Big] = 2^{-mr}$. We use a particular class of such hash functions, denoted by $H_{xor}(n, m)$, which is defined as follows. Let $h(y)[i]$ denote the i^{th} component of the vector $h(y)$. This family of hash functions is then defined as $\{h \mid h(y)[i] = a_{i,0} \oplus (\bigoplus_{k=1}^{n} a_{i,k} \cdot y[k]), a_{i,k} \in \{0,1\}, 1 \leq i \leq m,\ 0 \leq k \leq n\}$, where $\oplus$ denotes the XOR operation. By choosing values of $a_{i,k}$ randomly and independently, we can effectively choose a random hash function from $H_{xor}(n, m)$. It was shown in (Gomes, Sabharwal, and Selman 2007) that this family is 3-universal.

3 Related Work

Constrained counting, the problem of counting the number of solutions of a propositional formula, is known as #SAT. It is #P-complete (Valiant 1979), where #P is the set of counting problems associated with NP decision problems. Theoretical investigations of #P have led to the discovery of deep connections in complexity theory (Toda 1989; Valiant 1979), and there is strong evidence for its hardness (Arora and Barak 2009). It is also known that an efficient algorithm for constrained *sampling* would yield a fully polynomial randomized approximation scheme (FPRAS) for #P-complete inference problems (Jerrum and Sinclair 1996) – a possibility that lacks any evidence so far and is widely disbelieved.

In many applications of constrained counting, such as in probabilistic reasoning, exact counting may not be critically important, and approximate counts suffice. Even when exact counts are important, the inherent complexity of the problem may force one to work with approximate counters. Jerrum, Valiant, and Vazirani (Jerrum, Valiant, and Vazirani 1986) showed that approximate counting of solutions of CNF formulas, to within a given tolerance factor, can be done with high confidence in randomized polynomial time using an NP oracle. A key result of (Jerrum, Valiant, and Vazirani 1986) states that for many problems, generating solutions *almost uniformly* is inter-reducible with approximate counting; hence, they have similar complexity. Building on the Sipser's and Stockmeyer's early work (Sipser 1983; Stockmeyer 1983), Bellare, Goldreich, and Petrank (Bellare, Goldreich, and Petrank 2000) later showed that in fact, an NP-oracle suffices for generating solutions of CNF formulas *exactly uniformly* in randomized polynomial time. Unfortunately, these deep theoretical results have not been successfully reduced to practice. Our experience in implementing these techniques indicates that they do not scale in practice even to small problem instances involving few tens of variables (Meel 2014).

Industrial approaches to constrained sampling in the context of constrained-random verification (Naveh et al. 2006) either rely on Binary Decision Diagram (BDD)-based techniques (Yuan et al. 2004), which scale rather poorly, or use heuristics that offer no guarantee of performance or uniformity when applied to large problem instances (Kitchen and Kuehlmann 2007). In prior academic works (Ermon, Gomes, and Selman 2012b; Kirkpatrick, Gelatt, and Vecchi 1983; Gomes, Sabharwal, and Selman 2007; Wei, Erenrich, and Selman 2004), the focus is on heuristic techniques including Markov chain Monte Carlo (MCMC) methods and techniques based on random seeding of SAT solvers. These methods scale to large problem instances, but either offer very weak or no guarantees on the uniformity of sampling, or require the user to provide hard-to-estimate problem-specific parameters that crucially affect the performance and uniformity of sampling (Ermon et al. 2013b; 2013c; Gogate and Dechter 2008; Kitchen and Kuehlmann 2007).

The earliest approaches to #SAT were based on DPLL-style SAT solvers and computed exact counts. These approaches, e.g. CDP (Birnbaum and Lozinskii 1999), incrementally counted the number of solutions by introduc-

ing appropriate multiplication factors for each partial solution found, eventually covering the entire solution space. Subsequent counters such as Relsat (Jr. and Schrag 1997), Cachet (Sang et al. 2004), and sharpSAT (Thurley 2006) improved upon this by using several optimizations such as component caching, clause learning, and the like. Techniques based on BDDs and their variants (Minato 1993), or d-DNNF formulas (Darwiche 2004), have also been used to compute exact counts. Although exact counters have been successfully used in small- to medium-sized problems, scaling to larger problem instances has posed significant challenges. Consequently, a large class of practical applications has remained beyond the reach of exact counters (Meel 2014).

To overcome the scalability challenge, more efficient techniques for *approximate* counting have been proposed. The large majority of approximate counters used in practice are *bounding counters*, which provide lower or upper bounds but do not offer guarantees on the tightness of these bounds. Examples include SampleCount (Gomes et al. 2007a), BPCount (Kroc, Sabharwal, and Selman 2008), MBound and Hybrid-MBound (Gomes, Sabharwal, and Selman 2006), and MiniCount (Kroc, Sabharwal, and Selman 2008). Another category of counters is called *guarantee-less counters*. While these counters may be efficient, they provide no guarantees and the computed estimates may differ from the exact counts by several orders of magnitude (Gomes et al. 2007b). Examples of guarantee-less counters include ApproxCount (Wei and Selman 2005), SearchTreeSampler (Ermon, Gomes, and Selman 2012a), SE (Rubinstein 2012), and SampleSearch (Gogate and Dechter 2011).

4 Enabling Algorithmic Techniques

Hashing-based approaches to sampling and counting rely on three key algorithmic techniques – one classical, and two more recent: *universal hashing*, *satisfiability* (SAT) *solving*, and *satisfiability modulo theories* (SMT) *solving*.

Universal Hashing Universal hashing is an algorithmic technique that selects a hash function at random from a family of functions with a certain mathematical property (Carter and Wegman 1977). This technique *guarantees* a low expected number of collisions, even for an arbitrary distribution of the data being hashed. We have shown that universal hashing enables us to partition the set R_F of satisfying assignments of a formula F into roughly equally-sized "small" cells (Chakraborty, Meel, and Vardi 2013a). By choosing the definition of "small" carefully, we can sample almost uniformly by first choosing a random cell and then sampling uniformly inside the cell (Chakraborty, Meel, and Vardi 2014). Measuring the size of sufficiently many randomly chosen cells also gives us an approximation of the size of R_F (Chakraborty, Meel, and Vardi 2013b). To get a good approximation of uniformity and count, we employ a *3-universal* family of hash functions.

SAT Solving The paradigmatic NP-complete problem of boolean satisfiability (SAT) solving (Cook 1971), is a central problem in computer science. Efforts to develop practically successful SAT solvers go back to the 1950s. The past 20 years have witnessed a "SAT revolution" with the development of *conflict-driven clause-learning* (CDCL) solvers (Biere et al. 2009). Such solvers combine a classical backtracking search with a rich set of effective heuristics. While 20 years ago SAT solvers were able to solve instances with at most a few hundred variables, modern SAT solvers solve instances with up to *millions* of variables in a reasonable time (Malik and Zhang 2009). Furthermore, SAT solving is continuing to demonstrate impressive progress (Vardi 2014). Propositional sampling and counting are both extensions of SAT; thus, practically effective SAT solving is a key enabler in our work.

SMT Solving The Satisfiability Modulo Theories (SMT) problem is a decision problem for logical formulas in combinations of background theories. Examples of theories typically used are the theory of real numbers, the theory of integers, and the theories of various data structures such as lists, arrays, bit vectors, and others. SMT solvers have shown dramatic progress over the past couple of decades and are now routinely used in industrial software development (de Moura and Bjørner 2011). Even though we focus on sampling and counting for propositional formulas, we have to be able to combine propositional reasoning with reasoning about hash values, which requires the power of SMT solvers. In our approach, we express hashing by means of XOR constraints. These constraints can be reduced to CNF, but such a reduction typically leads to computational inefficiencies during SAT solving. Instead, we use CryptoMiniSAT, an SMT solver which combines the power of CDCL SAT solving with algebraic treatment of XOR constraints, to yield a highly effective solver for a combination of propositional CNF and XOR constraints (Soos, Nohl, and Castelluccia 2009).

5 Hashing-Based Sampling and Counting

In recent years, we have shown that the combination of universal hashing and SAT/SMT solving can yield a dramatic breakthrough in our ability to perform almost-uniform sampling and approximate counting for industrial-scale formulas with *hundreds of thousands* of variables (Chakraborty et al. 2014; Chakraborty, Meel, and Vardi 2013a; 2013b; 2014; Chakraborty et al. 2014; 2015a; 2015b; Ivrii et al. 2015). The algorithms and tools we developed provide the first scalable implementation with provable approximation guarantees of these fundamental algorithmic building blocks.

Approximate Counting In (Chakraborty, Meel, and Vardi 2013b), we introduced an approximate constrained counter, called ApproxMC. The algorithm employs XOR constraints to partition the solution space into "small" cells, where finding the right parameters to get the desired sizes of cells requires an iterative search. The algorithm then repeatedly invokes CryptoMiniSAT to exactly measure the size of sufficiently many random cells (to achieve the desired confidence) and returns an estimate given by multiplying the me-

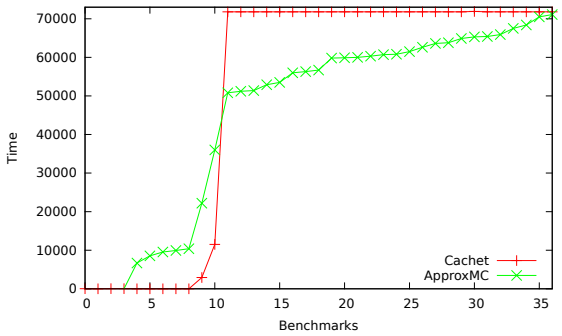

Figure 1: Performance comparison between ApproxMC and Cachet.

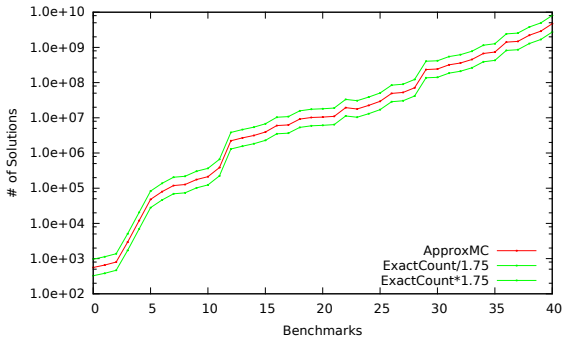

Figure 2: Quality of counts computed by ApproxMC. The benchmarks are arranged in increasing order of model counts.

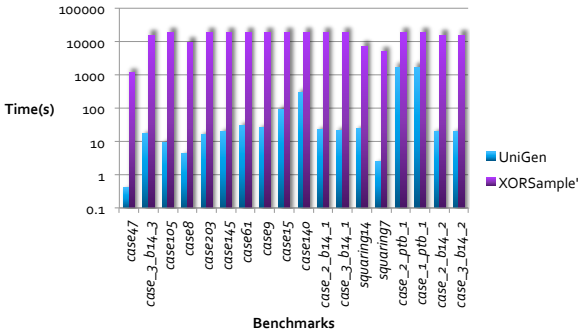

Figure 3: Performance comparison between UniGen and XORSample'.

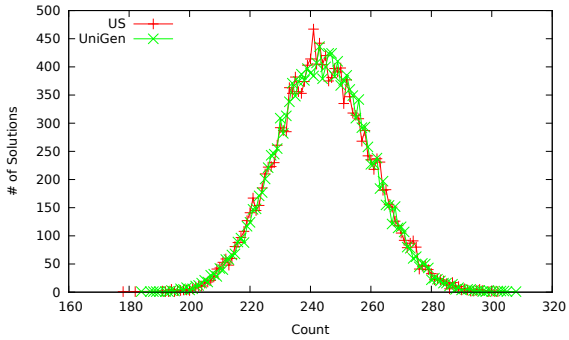

Figure 4: Histogram of solution frequencies comparing the distribution of UniGen to that of an ideal uniform sampler (US).

dian of the measured cell sizes by the number of cells covering the solution space.

We compared ApproxMC with Cachet, a well known exact counter (Sang et al. 2004), using a timeout of 20 hours. While Cachet can solve instances with up to 13K variables, ApproxMC can solve very large instances with over 400K variables (see Fig 1 for performance comparison over a subset of benchmarks). On instances where the exact count was available, the approximate count computed by ApproxMC was within 4% of the exact count, even when the tolerance requested was only 75% (See Figure 2).

Almost-uniform Sampling In (Chakraborty, Meel, and Vardi 2013a; 2014), we described an almost-uniform propositional sampler called UniGen. The algorithm first calls ApproxMC to get an approximate count of the size of the solution space. Using this count enables us to fine-tune the parameters for using XOR constraints to partition the solution space in such a way that a randomly chosen cell is expected to be small enough so the solutions in it can be enumerated and sampled. UniGen was able to handle formulas with approximately 0.5M variables. Its performance is several orders of magnitude better than that of a competing algorithm called XORSample' (Gomes, Sabharwal, and Selman 2007), which *does not* offer an approximation guarantee of almost uniformity (see Fig. 3). To evaluate the uniformity of sampling, we compared UniGen with a uniform sampler (called US, implemented by enumerating all solutions) on a benchmark with about 16K solutions. We generated 4M samples, each from US and UniGen; the resulting distributions were statistically indistinguishable. (see Fig. 4).

Weighted Counting and Sampling In (Chakraborty et al. 2014), we showed how to extend the above algorithms for approximate counting and almost-uniform sampling from the *unweighted* case, in which all assignments are given equal weight, to the *weighted* case, in which a weight function associates a weight with each assignment, and counting and sampling must be done with respect to these weights. Under some mild assumptions on the distribution of weights, we showed that the unweighted algorithms for approximate counting and almost-uniform sampling can be adapted to work in the weighted setting, using only a SAT solver (NP-oracle) and a black-box weight function $w(\cdot)$. For the algorithm to work well in practice, we require that the *tilt* of the weight function, which is the ratio of the maximum weight of a satisfying assignment to the minimum weight of a satisfying assignment, be small. This is a reasonable assumption for several important classes of problems (Chakraborty et al. 2014). We were able to handle formulas with over 60K variables, and our tools significantly outperformed SDD, a state-of-the-art exact weighted constrained counter (Dar-

wiche 2011).

Related recent work on hashing-based approaches The above work generated interest in the research community focused on sampling and counting problems. We introduced the approach of combining universal hashing with SAT solving for almost-uniform sampling and approximate counting in (Chakraborty, Meel, and Vardi 2013a; 2013b). Recently, (Belle, Van den Broeck, and Passerini 2015) proposed a probabilistic inference algorithm that is a "simple reworking" (quoting authors) of WeightMC.

Building on our underlying approach, Ermon et al. suggested further improvements and proposed an algorithm, called WISH, for constrained weighted counting (Ermon et al. 2013b; 2013c). The algorithm WISH falls short of WeightMC in terms of approximation guarantees and performance. Unlike WeightMC, WISH does not provide (ε, δ) guarantees and only provides constant-factor approximation guarantees. Furthermore, WISH requires access to a most-probable-explanation (MPE) oracle, which is an optimization oracle and more expensive in practice than the NP oracle (i.e. SAT solver) required by our approach (Park and Darwiche 2006). Recently, Zhu and Erom (Zhu and Ermon 2015) proposed an approximate algorithm, named RP-InfAlg, for approximate probabilistic inference. This algorithm provides very weak approximation guarantees, and requires the use of hard-to-estimate parameters. Furthermore, their experiments were done with specific values of parameters, which are not easy to guess or compute efficiently. The computational effort required in identifying the right values of the parameters is not addressed in their work. On the constrained sampling front, Ermon et al (Ermon et al. 2013a) proposed the PAWS algorithm, which provides weaker guarantees than UniGen. Like UniGen, PAWS requires a parameter estimation step. However, given a propositional formula with n variables, UniGen uses $O(n)$ calls to an NP oracle to estimate the parameters, while PAWS requires $O(n.\log n)$ calls.

6 Towards Scalable Sampling and Counting Algorithms

We now discuss four recent promising directions towards further improving the scalability of hashing-based sampling and counting algorithms.

Parallelism There has been a strong recent revival of interest in parallelizing a wide variety of algorithms to achieve improved performance (Larus 2009). One of the main goals in parallel-algorithm design is to achieve a speedup nearly linear in the number of processors, which requires the avoidance of dependencies among different parts of the algorithm (Eager, Zahorjan, and Lazowska 1989). Most of the sampling algorithms employed for sampling fail to meet this criterion, and are hence not easily parallelizable. For example, by using constraint solvers with randomized branching heuristics, samples with sequential dependencies are generated. Similarly, MCMC samplers often require a long sequential walk before converging to the stationary distribution. The lack of techniques for sampling solutions of constraints in parallel while preserving guarantees of effectiveness in finding bugs has been a major impediment to high-performance constrained-random verification (CRV).

The algorithm UniGen2 presented in (Chakraborty et al. 2015a) takes a step forward in addressing this problem. It has an initial preprocessing step that is sequential but low-overhead, followed by inherently parallelizable sampling steps. It generates samples (stimuli) that are provably nearly as effective as those generated by an almost-uniform sampler for purposes of detecting a bug. Furthermore, experimental evaluation over a diverse set of benchmarks demonstrates that the performance improvement for UniGen2 scales linearly with the number of processors: low communication overhead allows UniGen2 to achieve efficiency levels close to those of ideal distributed algorithms. Given that current practitioners are forced to trade guarantees of effectiveness in bug hunting for scalability, the above properties of UniGen2 are significant. Specifically, they enable a new paradigm for CRV wherein both stimulus generation and simulation are done in parallel, providing the required runtime performance without sacrificing theoretical guarantees.

Independence Another way to boost the performance of UniGen is by reducing "waste" of hashed solutions. Currently, after UniGen chooses a random cell of solutions, it selects a single sample from that cell and throws away all the other solutions. The reason for this "wasting" of solutions is to ensure that the samples obtained are not only almost-uniformly distributed, but also *independently* distributed. Independence is an important feature of sampling, which is required to ensure that probabilities are multiplicative. (For example, in a sequence of Bernoulli coin tosses, independence is required to ensure that an outcome of heads is almost-certain in the limit.) In MCMC sampling, ensuring independence requires very long random walks between samples. An important feature of UniGen is that its almost-uniform samples are independent. Such independence, however, comes at the cost of throwing away many solutions.

In some applications, such as constrained-random verification, full independence between successive samples is not needed: it is only required that the samples provide good coverage over the space of all solutions. In (Chakraborty et al. 2015a), our proposed algorithm, UniGen2, gains efficiency by relaxing independence, while still maintaining provable coverage guarantees. Specifically, a cell is "small" in the sense discussed earlier when the number of solutions it contains is between parameters $\mathrm{loThresh}$ and $\mathrm{hiThresh}$, both computed from the tolerance ε. Instead of picking only one solution from a small cell, UniGen2 randomly picks $\mathrm{loThresh}$ distinct solutions. This leads to a theoretical guarantee that UniGen2 requires significantly fewer SAT calls than UniGen to obtain a given level of bug-finding effectiveness in constrained-random verification.

Independent Support These hashing-based techniques crucially rely on the ability of combinatorial solvers to solve propositional formulas represented as a conjunction of the input formula and 3-universal hash functions. Due to our use

of H_{xor}, this translates to *hybrid* formulas which are conjunctions of CNF and XOR clauses. Therefore, a key challenge is to further speed up search for the solutions of such hybrid formulas. The XOR clauses invoked here are *high-density* clauses, consisting typically of $n/2$ variables, where n is the number of variables in the input formula. Experience has shown that the high density of the XOR clauses plays a major role in runtime performance of solving hybrid formulas.

Recently, Ermon et al. introduced a technique to reduce the density of XOR constraints. Their technique gives hash functions with properties that are weaker than 2–universality, but sufficient for approximate counting (Ermon et al. 2014). In practice, however, this approach does not appear to achieve significant reduction in the density of XOR constraints and, therefore, the practical impact on runtime is not significant. Furthermore, It is not yet clear whether the constraints resulting from Ermon et al's techniques (Ermon et al. 2014) can be used for constrained sampling, while providing the same theoretical guarantees that our work gives.

Motivated by the problem of high-density XOR constraints, we proposed the idea of restricting the hash functions to only the independent support $\mathcal{I}$ of a formula, and showed that even with this restriction, we obtain the required universality properties of hash functions needed for constrained sampling and counting techniques (Chakraborty, Meel, and Vardi 2014). Since many practical instances admit independent supports much smaller than the total number of variables (e.g. we may drop all variables introduced by Tseitin encoding), this often allows the use of substantially less dense XOR constraints. While it is possible in many cases to obtain an over-approximation of $\mathcal{I}$ by examining the domain from which the instance is derived, the work in (Chakraborty, Meel, and Vardi 2014) does not provide an algorithmic approach for determining $\mathcal{I}$, and experience has shown that the manual approach is error prone.

In (Ivrii et al. 2015), we proposed the first algorithm, called MIS, to find minimal independent supports. The key idea of this algorithmic procedure is the reduction of the problem of minimizing an independent support of a Boolean formula to Group Minimal Unsatisfiable Subset (GMUS). To illustrate the practical value of this approach, we used MIS to compute a minimal independent support for each of our UniGen2 and ApproxMC benchmarks, and ran both algorithms using hashing only over the computed supports. Over a wide suite of benchmarks, experiments demonstrated that this hashing scheme improves the runtime performance of UniGen2 and ApproxMC by two to three orders of magnitude. It is worth noting that these runtime improvements come at no cost of theoretical guarantees: both UniGen2 and ApproxMC still provide the same strong theoretical guarantees.

From Weighted to Unweighted Model Counting Recent approaches to weighted model counting (WMC) have focused on adapting unweighted model counting (UMC) techniques to work in the weighted setting (Sang, Bearne, and Kautz 2005; Xue, Choi, and Darwiche 2012; Chakraborty et al. 2014). Such adaptation requires intimate understanding of the implementation details of the UMC techniques, and on-going maintenance, since some of these techniques evolve over time. In (Chakraborty et al. 2015b), 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 make 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.

7 Conclusion

Constrained sampling and counting problems have a wide range of applications in artificial intelligence, verification, machine learning and the like. In contrast to prior work that failed to provide scalable algorithms with strong theoretical guarantees, hashing-based techniques offer scalable sampling and counting algorithms with rigorous guarantees on quality of approximation. Yet, many challenges remain in making this approach more viable for real-world problem instances. Promising directions of future research include designing efficient hash functions and developing SAT/SMT solvers specialized to handle constraints arising from these techniques.

References

Arora, S., and Barak, B. 2009. *Computational Complexity: A Modern Approach*. Cambridge Univ. Press.

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. 2000. Uniform generation of NP-witnesses using an NP-oracle. *Information and Computation* 163(2):510–526.

Belle, V.; Van den Broeck, G.; and Passerini, A. 2015. Hashing-based approximate probabilistic inference in hybrid domains. In *Proc. of UAI*.

Biere, A.; Heule, M.; Van Maaren, H.; and Walsh, T. 2009. *Handbook of Satisfiability*. IOS Press.

Birnbaum, E., and Lozinskii, E. L. 1999. The good old

Davis-Putnam procedure helps counting models. *Journal of Artificial Intelligence Research* 10(1):457–477.

Carter, J. L., and Wegman, M. N. 1977. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, 106–112. ACM.

Chakraborty, S.; Fremont, D. J.; Meel, K. S.; Seshia, S. A.; and Vardi, M. Y. 2014. Distribution-aware sampling and weighted model counting for SAT. In *Proc. 28th Conference on Artificial Intelligence*, 1722–1730.

Chakraborty, S.; Fremont, D.; Meel, K.; Seshia, S.; and Vardi, M. 2015a. On parallel scalable uniform sat witness generation. In *Tools and Algorithms for the Construction and Analysis of Systems*, 304–319.

Chakraborty, S.; Fried, D.; Meel, K. S.; and Vardi, M. Y. 2015b. From weighted to unweighted model counting. In *Proceedings of the 24th International Conference on Artificial Intelligence*, 689–695. AAAI Press.

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. 51st Annual Design Automation Conf.*, 1–6.

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

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC, 151–158. New York, NY, USA: ACM.

Cooper, G. F. 1990. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence* 42(2):393–405.

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

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proc. 22nd Int'l Joint Conf. on Artificial Intelligence*, 819–826.

de Moura, L., and Bjørner, N. 2011. Satisfiability Modulo Theories: introduction and applications. *Commun. ACM* 54(9):69–77.

Eager, D. L.; Zahorjan, J.; and Lazowska, E. D. 1989. Speedup versus efficiency in parallel systems. *IEEE Trans. on Computers* 38(3):408–423.

Ermon, S.; Gomes, C.; 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*.

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.

Ermon, S.; Gomes, C. P.; and Selman, B. 2012a. Uniform solution sampling using a constraint solver as an oracle. In *Proc. of UAI*.

Ermon, S.; Gomes, C. P.; and Selman, B. 2012b. Uniform solution sampling using a constraint solver as an oracle. In *Proc. of UAI*, 255–264.

Gogate, V., and Dechter, R. 2008. Approximate Solution Sampling (and Counting) on AND/OR Spaces. In *Proc. of CP*, 534–538.

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

Gomes, C. P.; Hoffmann, J.; Sabharwal, A.; and Selman, B. 2007a. From sampling to model counting. In *Proc. of IJCAI*, 2293–2299.

Gomes, C. P.; Hoffmann, J.; Sabharwal, A.; and Selman, B. 2007b. From sampling to model counting. In *Proc. of IJCAI*, 2293–2299.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Model counting: A new strategy for obtaining good bounds. In *Proc. of AAAI*, volume 21, 54–61.

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

Ivrii, A.; Malik, S.; Meel, K.; and Vardi, M. 2015. On computing minimal independent support and its applications to sampling and counting. *Constraints* 1–18.

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. *Theoretical Computer Science* 43(2-3):169–188.

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

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

Kitchen, N., and Kuehlmann, A. 2007. Stimulus generation for constrained random simulation. In *Proc. of ICCAD*, 258–265.

Kroc, L.; Sabharwal, A.; and Selman, B. 2008. Leveraging belief propagation, backtrack search, and statistics for model counting. In *Proc. of CPAIOR*, 127–141.

Larus, J. 2009. Spending moore's dividend. *Commun. ACM* 52(5):62–69.

Malik, S., and Zhang, L. 2009. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM* 52(8):76–82.

Meel, K. S. 2014. *Sampling Techniques for Boolean Satisfiability*. M.S. Thesis, Rice University.

Minato, S. 1993. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proc. of Design Automation Conference*, 272–277.

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. 2006. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research* 101–133.

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

Rubinstein, R. 2012. Stochastic Enumeration Method for Counting NP-Hard Problems. *Methodology and Computing in Applied Probability* 1–43.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach (3dd Ed.)*. Prentice Hall.

Sang, T.; Bearne, P.; and Kautz, H. 2005. Performing bayesian inference by weighted model counting. In *Prof. 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*.

Sipser, M. 1983. A complexity theoretic approach to randomness. In *Proc. of STOC*, 330–335.

Soos, M.; Nohl, K.; and Castelluccia, C. 2009. Extending SAT Solvers to Cryptographic Problems. In *Proc. of SAT*. Springer-Verlag.

Stockmeyer, L. 1983. The complexity of approximate counting. In *Proc. of STOC*, 118–126.

Thurley, M. 2006. SharpSAT: counting models with advanced component caching and implicit BCP. In *Proc. of SAT*, 424–429.

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

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

Vardi, M. 2014. Boolean satisfiability: Theory and engineering. *Commun. ACM* 57(3):5.

Wei, W., and Selman, B. 2005. A new approach to model counting. In *Proc. of SAT*, 2293–2299. Springer.

Wei, W.; Erenrich, J.; and Selman, B. 2004. Towards efficient sampling: Exploiting random walk strategies. In *Proc. of AAAI*, 670–676.

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

Yuan, J.; Aziz, A.; Pixley, C.; and Albin, K. 2004. Simplifying boolean constraint solving for random simulation-vector generation. *IEEE Trans. on CAD of Integrated Circuits and Systems* 23(3):412–420.

Zhu, M., and Ermon, S. 2015. A hybrid approach for probabilistic inference using random projections. In *Proc. of ICML*, 2039–2047.