On the Sparsity of XORs in Approximate Model Counting[⋆]

Durgesh Agrawal[1], Bhavishya[1], and Kuldeep S. Meel[2]

[1] Indian Institute of Technology, Kanpur
[2] School of Computing, National University of Singapore

Abstract. Given a Boolean formula φ, the problem of model counting, also referred to as #SAT, is to compute the number of solutions of φ. The hashing-based techniques for approximate counting have emerged as a dominant approach, promising achievement of both scalability and rigorous theoretical guarantees. The standard construction of strongly 2-universal hash functions employs *dense* XORs (i.e., involving half of the variables in expectation), which is widely known to cause degradation in the runtime performance of state of the art SAT solvers. Consequently, the past few years have witnessed an intense activity in the design of *sparse* XORs as hash functions. Such constructions have been proposed with beliefs to provide runtime performance improvement along with theoretical guarantees similar to that of dense XORs.
The primary contribution of this paper is a rigorous theoretical and empirical analysis to understand the effect of the sparsity of XORs. In contradiction to prior beliefs of applicability of analysis for sparse hash functions to all the hashing-based techniques, we prove a contradictory result. We show that the best-known bounds obtained for sparse XORs are still too weak to yield theoretical guarantees for a large class of hashing-based techniques, including the state of the art approach ApproxMC3. We then turn to a rigorous empirical analysis of the performance benefits of sparse hash functions. To this end, we first design, to the best of our knowledge, the most efficient algorithm called SparseCount2 using sparse hash functions, which achieves at least up to two orders of magnitude performance improvement over its predecessor. Contradicting the current beliefs, we observe that SparseCount2 still falls short of ApproxMC3 in runtime performance despite the usage of dense XORs in ApproxMC3. In conclusion, our work showcases that the question of whether it is possible to use short XORs to achieve scalability while providing strong theoretical guarantees is still wide open.

1 Background and Introduction

Given a Boolean formula φ, the problem of model counting, also referred to as #SAT, is to compute the number of solutions of φ. Model counting is a

[⋆] The author list has been sorted alphabetically by last name; this order should not be used to determine the extent of authors' contributions. Part of the work was carried out during the first two authors' internships at National University of Singapore.

fundamental problem in computer science with a wide range of applications ranging from quantified information flow, reliability of networks, probabilistic programming, Bayesian networks, and others [23,24,17,22,10,5,4].

Given the computational intractability of #SAT, attention has been focused on the approximation of #SAT [31,29]. In a breakthrough result, Stockmeyer provided a hashing-based randomized approximation scheme for counting that makes polynomially many invocations of an NP oracle [28]. The procedure, however, was computationally prohibitive in practice at that time, and no practical tools existed based on Stockmeyer's proposed algorithmic framework until the early 2000s [17]. Motivated by the success of SAT solvers, there has been a surge of interest in the design of hashing-based techniques for approximate model counting in the past decade [16,8,13,9,26,25].

The core idea of the hashing-based framework is to employ pairwise independent hash functions[3] to partition the solution space into *roughly equal-sized small* cells, wherein a cell is called *small* if it has solutions less than or equal to a pre-computed threshold, denoted by *thresh*. A SAT solver is employed to check if a cell is small by enumerating solutions one-by-one until either there are no more solutions or we have already enumerated $thresh + 1$ solutions. The current state of the art techniques can be broadly classified into two categories:

- The first category of techniques, henceforth called Cat1, consists of techniques that compute a constant factor approximation by setting *thresh* to a constant and use Stockmeyer's technique of constructing multiple copies of the input formula. [30,12,2,1,32]
- The second class of techniques, henceforth called Cat2, consists of techniques that directly compute an (ε, δ)-estimate by setting *thresh* to $\mathcal{O}(\frac{1}{\varepsilon^2})$, and hence invoking the underlying NP oracle $\mathcal{O}(\frac{1}{\varepsilon^2})$ times [8,9,22,26,25,21,7].

The current state of the art technique, measured by runtime performance, is ApproxMC3, which falls into the class of Cat2 techniques [26]. The proofs of correctness for all the hashing-based techniques involve the use of concentration bounds due to pairwise independent hash functions.

The standard construction of pairwise independent hash functions employed in these techniques can be expressed as a conjunction of XOR constraints such that every variable is chosen with probability $f = 1/2$ for each XORs. As such, each XOR contains, on an average, $n/2$ variables. A SAT solver is invoked to enumerate solutions of the formula φ in conjunction with these XOR constraints. The performance of SAT solvers, however, degrades with an increase in the size of XORs [16]. Therefore recent efforts have focused on the design of hash functions where each variable is chosen with probability $f < 1/2$ [14,11,18,2,1]. We refer to the XOR constructed with $f = 1/2$ as dense XORs while those constructed with $f < 1/2$ as sparse XORs. In particular, given a hash function, h and cell α, the random variable of interest, denoted by $\mathsf{Cnt}_{\langle \varphi, h, \alpha \rangle}$ is the number of solutions

[3] Pairwise independent hash functions were initially referred to as strongly 2-universal hash functions in [6]. The prior work on approximate counting often uses the term *2-universal hashing* to refer to strongly 2-universal hash functions.

of φ that h maps to cell α. The pairwise independence of dense XORs is known to bound the variance of $\mathsf{Cnt}_{\langle\varphi,h,\alpha\rangle}$ by the expectation of $\mathsf{Cnt}_{\langle\varphi,h,\alpha\rangle}$, which is sufficient for their usage for both Cat1 and Cat2 techniques.

In a significant result, Asteris and Dimakis [3], and Zhao et al. [32] showed that $f = \mathcal{O}(\log n/n)$ asymptotically suffices for Cat1 techniques. It is worth pointing that $f = \mathcal{O}(\log n/n)$ provides weaker guarantees on the variance of $\mathsf{Cnt}_{\langle\varphi,h,\alpha\rangle}$ as compared to the case when $f = 1/2$. However, Zhao et al. showed that the weaker guarantees are sufficient for Cat1 techniques with only polynomial overhead on the time complexity. Furthermore, Zhao et al. provided necessary and sufficient conditions on the required asymptotic value of f and proposed a new algorithm SparseCount that uses the proposed family of hash functions. One would expect that the result of Zhao et al. would settle the quest for efficient hash functions. However, upon closer examination, few questions have been left unanswered in Zhao et al.'s work and subsequent follow-up studies [1,22,9].

1. Can the hash function constructed by Zhao et al. be used for Cat2 techniques, in particular for state of the art hashing-based techniques like ApproxMC3?
2. In practice, can the overhead due to the weakness of theoretical guarantees of sparse XORs proposed by Zhao et al. be compensated by the gain of performance due to sparse XORs in the runtime of SparseCount?
3. Is the runtime performance of SparseCount competitive to that of ApproxMC3? The reader may observe that Zhao et al.'s paper does not compare their proposed algorithm for (ε, δ)-guarantees, called SparseCount, with state of the art algorithms at that time such as ApproxMC2, which is now in its third version, ApproxMC3 [26]. Therefore the question of whether the proposed sparse XORs are efficient in runtime was not settled. It is perhaps worth remarking that another line of work based on the construction of sparse XORs using low-density parity codes is known to introduce significant slowdown [2,1] (See Section 9 of [1]).

The primary contribution of this paper is a rigorous theoretical and empirical analysis to understand the effect of sparse XORs for approximate model counters. In particular, we make the following key contributions:

1. We prove that the bounds obtained by Zhao et al., which are the strongest known bounds at this point, for the variance of $\mathsf{Cnt}_{\langle\varphi,h,\alpha\rangle}$, are still too weak for the analysis of ApproxMC3. To the best of our knowledge, this is the first time the need for stronger bounds in the context of Cat2 techniques has been identified.
2. Since the weakness of bounds prevents usage of sparse hash functions in ApproxMC3, we design the most efficient algorithm, to the best of our knowledge, using sparse hash functions. To this end, we propose an improvement of SparseCount, called SparseCount2, that reduces the number of SAT calls from linear to logarithmic and significantly improves the runtime performance of SparseCount. The improvement from linear to logarithmic uses the idea of prefix-slicing introduced by Chakraborty, Meel, and Vardi [9] for ApproxMC2.

3. We next present a rigorous empirical study involving a benchmark suite totaling over 1800 instances of runtime performance of SparseCount2 vis-a-vis the state of the art approximate counting technique, ApproxMC3. Surprisingly and contrary to current beliefs, we discover that ApproxMC3, which uses dense XORs significantly outperforms SparseCount2 for every benchmark. It is worth remarking that both ApproxMC3 and SparseCount2 use identical SAT solver for underlying SAT calls and similar to other hashing-based techniques, over 99% for each of the algorithms is indeed consumed by the underlying SAT solver.

Given the surprising nature of our results, few words are in order. First of all, our work identifies the tradeoffs involved in the usage of sparse hash functions and demonstrates that the variance bounds offered by sparse hash functions are too weak to be employed in the state of the art techniques. Secondly, our work demonstrates that the weakness of variance bounds leads to such a large overhead that the algorithms using sparse hash functions scale much worse compared to the algorithms without sparse XORs. Thirdly and finally, we believe the negative results showcase that the question of the usage of sparse XORs to achieve scalability while providing strong theoretical guarantees is still wide open. In an upcoming work, Meel ⓡ Akshay[4] [21] define a new family of hash functions, called concentrated hashing, and provide a new construction of sparse hash functions belonging to concentrated hashing, and design a new algorithmic framework on top of ApproxMC, which is shown to achieve runtime improvements.

The rest of the paper is organized as follows. We discuss notations and preliminaries in Section 2. We then discuss the weakness of guarantees offered by sparse XORs in Section 3. In Section 4, we seek to design an efficient algorithm that utilizes all the advancements, to the best of our knowledge, in approximate model counting community. We present a rigorous empirical study comparing performance of SparseCount, SparseCount2, and ApproxMC3 in Section 5 and conclude in Section 6.

2 Preliminaries and Notations

Let φ be a Boolean formula in conjunctive normal form (CNF), and let $\mathsf{Vars}(\varphi)$ be the set of variables appearing in φ. The set $\mathsf{Vars}(\varphi)$ is also called the *support* of φ. Unless otherwise stated, we will use n to denote the number of variables in φ i.e., $|\mathsf{Vars}(\varphi)|$. An assignment of truth values to the variables in $\mathsf{Vars}(\varphi)$ is called a *satisfying assignment* or *witness* of φ if it makes φ evaluate to true. We denote the set of all witnesses of φ by R_φ.

We write $\mathsf{Pr}\left[\mathcal{Z}\right]$ to denote the probability of outcome $\mathcal{Z}$. The expected value of $\mathcal{Z}$ is denoted $\mathsf{E}\left[\mathcal{Z}\right]$ and its variance is denoted $\sigma^2\left[\mathcal{Z}\right]$.

The *propositional model counting problem* is to compute $|R_\varphi|$ for a given CNF formula φ. A *probably approximately correct* (PAC) counter is a probabilistic algorithm $\mathsf{ApproxCount}(\cdot,\cdot,\cdot)$ that takes as inputs a formula F, a tolerance $\varepsilon > 0$,

[4] ⓡ is used to denote random author ordering, as suggested by the authors.

and a confidence parameter $\delta \in (0,1]$, and returns a count c with (ε, δ)-guarantees, i.e., $\mathsf{Pr}\Big[|R_\varphi|/(1+\varepsilon) \le c \le (1+\varepsilon)|R_\varphi|\Big] \ge 1-\delta$.

In this work, we employ a family of universal hash functions. Let $H(n,m) \triangleq \{h : \{0,1\}^n \to \{0,1\}^m\}$ be a family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$. We use $h \xleftarrow{R} H$ to denote the probability space obtained by choosing a function h uniformly at random from H.

In this work, we will use the concept of *prefix-slicing* introduced by Chakraborty et al. [9]. For $h \in H(n,m)$, formally, for every $j \in \{1,\ldots,m\}$, the j^{th} prefix-slice of h, denoted $h^{(j)}$, is a map from $\{0,1\}^n$ to $\{0,1\}^j$, such that $h^{(j)}(y)[i] = h(y)[i]$, for all $y \in \{0,1\}^n$ and for all $i \in \{1,\ldots j\}$. Similarly, the j^{th} prefix-slice of α, denoted $\alpha^{(j)}$, is an element of $\{0,1\}^m$ such that $\alpha^{(j)}[i] = \alpha[i]$ for all $i \in \{1,\ldots j\}$. The randomness in the choices of h and α induce randomness in the choices of $h^{(m)}$ and $\alpha^{(m)}$. However, the $(h^{(j)}, \alpha^{(j)})$ pairs chosen for different values of j are no longer independent. Specifically, $h^{(k)}(y)[i] = h^{(\ell)}(y)[i]$ and $\alpha^{(k)}[i] = \alpha^{(l)}[i]$ for $1 \le k \le \ell \le m$ and for all $i \in \{1,\ldots k\}$.

For a formula φ, $h \in H(n,m)$, and $\alpha \in \{0,1\}^m$, we define $\mathsf{Cnt}_{\langle F,h^{(i)},\alpha^{(i)}\rangle} := |\{y \in R_\varphi \mid h^{(i)}(y) = \alpha^{(i)}\}|$, i.e. the number of solutions of φ mapped to $\alpha^{(i)}$ by $h^{(i)}$. For the sake of notational clarity, whenever $h^{(i)}$ and $\alpha^{(i)}$ are clear from the context, we will use $\mathsf{Cnt}_{\langle i\rangle}$ as a shorthand for $\mathsf{Cnt}_{\langle F,h^{(i)},\alpha^{(i)}\rangle}$.

Definition 1. *[6] A family of hash functions $H(n,m)$ is* pairwise independent *(also known as strongly 2-universal) if $\forall\ \alpha_1, \alpha_2 \in \{0,1\}^m$, $\forall$ distinct $y_1, y_2 \in \{0,1\}^n$, $h \xleftarrow{R} H$, we have $\mathsf{Pr}[h(y_1) = \alpha_1 \wedge h(y_2) = \alpha_2] = \frac{1}{2^{2m}}$.*

Definition 2. *Let $A \in \{0,1\}^{m\times n}$ be a random matrix whose entries are Bernoulli i.i.d. random variables such that $f_i = \mathsf{Pr}[A[i,j]=1]$ for all $j \in [n]$. Let $b \in \{0,1\}^m$ be chosen uniformly at random, independently from A. Let $h_{A,b}(y) = Ay+b$ and $H^{\{f_i\}}(n,m) = \{h_{A,b} : \{0,1\}^n \to \{0,1\}^m\}$, where $h_{A,b} \xleftarrow{R} H^{\{f_i\}}(n,m)$ is chosen randomly according to this process. Then, $H^{\{f_i\}}(n,m)$ is defined as hash family with $\{f_i\}$-sparsity.*

Since we can represent hash functions in $H^{\{f_i\}}(n,m)$ using a set of XORs; we will use *dense XORs* to refer to hash functions with $f_i = \frac{1}{2}$ for all i while we use *sparse XORs* to refer to hash functions with $f_i < \frac{1}{2}$ for some i. Note that $H^{\{f_i = \frac{1}{2}\}}(n,m)$ is the standard pairwise independent hash family, also denoted as $H_{xor}(n,m)$ in earlier works [22].

Definition 3. *[11] Let $k \ge 0$ and $\delta > 2$. Let Z be a random variable with $\mu = E[Z]$. Then Z is strongly (k,δ)-concentrated if $Pr[|Z-\mu| \ge \sqrt{k}] \le 1/\delta$ and weakly (k,δ)-concentrated if both $Pr[Z \le \mu - \sqrt{k}] \le 1/\delta$ and $Pr[Z \ge \mu + \sqrt{k}]$ $\le 1/\delta$.*

2.1 Related Work

Gomes et al. [15] first identified the improvements in solving time due to the usage of sparse XORs in approximate model counting algorithms. The question of whether sparse XORs can provide the required theoretical guarantees was left open. A significant progress in this direction was achieved by Ermon et al. [11], who provided the first rigorous analysis of the usage of sparse XOR constraints. Building on Ermon et al., Zhao et al. [32] and Asteris and Dimakis [3] independently provided further improved analysis of Ermon et al. and showed that probability $f = \mathcal{O}(\frac{\log n}{n})$ suffices to provide constant factor approximation, which can be amplified to $(1+\varepsilon)$ approximation.

While the above mentioned efforts focused on each entry of A to be i.i.d., Achlioptas and Theodorpoulos [2], Achlioptas, Hammoudeh, and Theodorpoulos [1] investigated the design of hash functions where A is a structured matrix by drawing on connections to the error correcting codes. While their techniques provide a construction of sparse constraints, the constants involved in asymptotics lead to impractical algorithms for (ε, δ) guarantees (See Section 9 of [1]). The work of Achlioptas et al. demonstrates the promise and limitations of structured random matrices in the design of hashing-based algorithms; however, there is no such study in the case when all the entries are i.i.d. In this paper, we theoretically improve the construction proposed by Asteris and Dimakis [3], and Zhao et al. [32] and perform a rigorous empirical study to understand the tradeoffs of sparsity.

3 Weakness of Guarantees offered by Sparse XORs

In this section, we present the first contribution of this paper: demonstration of the weakness of theoretical guarantees obtained in prior work [11,3,32] for sparse XORs. To this end, we investigate whether the bounds offered by Zhao et al. on the variance of $\mathsf{Cnt}_{\langle i \rangle}$, which are the strongest bounds known on sparse XORs, can be employed in the analysis of Cat2 techniques. For clarity of exposition, we focus on the usage of sparse XOR bounds in ApproxMC3, but our conclusions extend to other Cat2 techniques, as pointed out below.

The analysis of ApproxMC3 employs the bounds on the variance of $\mathsf{Cnt}_{\langle i \rangle}$ using the following standard concentration bounds.

Lemma 1. *For every $\beta > 0, 0 < \varepsilon \leq 1$, $0 \leq i \leq n$, we have:*

1. ***Chebyshev Inequality***

$$\mathsf{Pr}\left[\left|\mathsf{Cnt}_{\langle i \rangle} - \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]\right| \geq \frac{\varepsilon}{1+\varepsilon}\mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]\right] \leq \frac{(1+\varepsilon)^2\sigma^2[\mathsf{Cnt}_{\langle i \rangle}]}{\varepsilon^2\mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^2}$$

2. ***Paley-Zygmund Inequality***

$$\mathsf{Pr}[\mathsf{Cnt}_{\langle i \rangle} \leq \beta\mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]] \leq \frac{1}{1 + \dfrac{(1-\beta)^2\mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^2}{\sigma^2[\mathsf{Cnt}_{\langle i \rangle}]}}$$

The analysis of Cat2 techniques (and ApproxMC3 in particular) bounds the failure probablity of the underlying algorithm by upper bounding the above expressions for appropriately chosen values of i. To obtain meaningful upper bounds, these techniques employ the inequality $\sigma^2[\mathsf{Cnt}_{\langle i\rangle}] \leq \mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}]$ obtained via the usage of 2-universal hash functions[5].

Recall, that the core idea of the hashing-based framework is to employ 2-universal hash functions to partition the solution space into *roughly equal sized small* cells, wherein a cell is called *small* if it has solutions less than or equal to a pre-computed threshold, denoted by *thresh*, which is chosen as $\mathcal{O}(1/\varepsilon^2)$. To this end, the analysis lower bounds $\mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}]$ by $\frac{thresh}{2}$, which allows the denominator to be lower bounded by a constant. Given that *thresh* can be set to $\mathcal{O}(\frac{1}{\varepsilon^2})^{1/c}$ for some $c > 0$, we can relax the requirement on the chosen hash family to ensuring $\sigma^2[\mathsf{Cnt}_{\langle i\rangle}] \leq \mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}]^{2-c}$ for some $c > 0$. Note that pairwise independent hash functions based on dense XORs ensure $\sigma^2[\mathsf{Cnt}_{\langle i\rangle}] \leq \mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}]$ (i.e., $c = 1$).

We now investigate the guarantees provided by sparse XORs. To this end, we first recall the following result, which follows from combining Theorem 1 and Theorem 3 of [11].

Lemma 2. *[11][6] For $2 \leq |R_F| \leq 2^n$, let*

$$w^* = max\left\{ w \mid \sum_{j=1}^{w} \binom{n}{j} \leq |R_F| - 1 \right\}$$

$$q^* = |R_F| - 1 - \sum_{w=1}^{w^*} \binom{n}{w}$$

$$\eta = \frac{1}{|R_F|-1}\left(q^*\left(\frac{1}{2} + \frac{1}{2}(1-2f)^{w^*+1}\right)^m + \sum_{w=1}^{w^*}\binom{n}{w}\left(\frac{1}{2}+\frac{1}{2}(1-2f)^w\right)^m \right)$$

For $h \xleftarrow{R} H^{\{f_j\}}(n,m)$, we have:

$$\sigma^2[\mathsf{Cnt}_{\langle i\rangle}] \leq \mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}] + \eta \mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}](|R_F| - 1) - \mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}]^2.$$

Zhao et al. [32], building on Ermon et al. [11], obtain the following bound (see, Lemma 8 and Lemma 10 of [32]).

Lemma 3. *[32] Define $k = 2^m \eta(1 - \frac{1}{|R_F|})$. Then $k \leq \gamma$ for $\gamma > 1$.*

The bound on $\sigma^2[\mathsf{Cnt}_{\langle i\rangle}]$ from Zhao et al. can be stated as:

Theorem 1. $\sigma^2[\mathsf{Cnt}_{\langle i\rangle}] \leq \zeta$ *where* $\zeta \in \Omega(\mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}]^2)$.

[5] While we are focusing on ApproxMC3, the requirement of $\sigma^2[\mathsf{Cnt}_{\langle i\rangle}] \leq \mathsf{E}[\mathsf{Cnt}_{\langle i\rangle}]$ holds for other Cat2 techniques.

[6] The expression stated in the Theorem can be found in the revised version at `https://cs.stanford.edu/~ermon/papers/SparseHashing-revised.pdf` (Accessed: May 10, 2020).

Proof.

$$\sigma^2[\mathsf{Cnt}_{\langle i \rangle}] \leq \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}] + \eta \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}](|R_F| - 1) - \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^2.$$

(Substituting $|R_F| = \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}] \times 2^m$, we have)

$$\sigma^2[\mathsf{Cnt}_{\langle i \rangle}] \leq \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}] + 2^m \eta \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^2(1 - 1/|R_F|) - \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^2$$

Substituting $k = 2^m \eta(1 - \frac{1}{|R_F|})$, we have:

$$\sigma^2[\mathsf{Cnt}_{\langle i \rangle}] \leq \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}] + (k-1)\mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^2 = \zeta.$$

Using Corollary 3, we have $\zeta \in \Omega(\mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^2)$.

Recall, the analysis of ApproxMC3 requires us to upper bound $\sigma^2[\mathsf{Cnt}_{\langle i \rangle}]$ by $\mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^{2-c}$ for $c > 0$. Since the best-known bounds on $\sigma^2[\mathsf{Cnt}_{\langle i \rangle}]$ lower bound $\sigma^2[\mathsf{Cnt}_{\langle i \rangle}]$ by $\mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]^2$, these bounds are not sufficient to be used by ApproxMC3. At this point, one may wonder as to what is the key algorithmic difference between Cat1 and Cat2 that necessitates the use of stronger bounds: Cat1 techniques compute a constant factor approximation and then make use of Stockmeyer's argument to lift a constant factor approximation to $(1+\varepsilon)$-approximation, whereas, Cat2 techniques directly compute a $(1+\varepsilon)$-approximation, which necessitates the usage of stronger concentration bounds.

4 SparseCount2: An Efficient Algorithm for Sparse XORs

The inability of sparse XORs to provide good enough bounds on variance for usage in Cat2 techniques, in particular ApproxMC3, leads us to ask: how do we design the *most efficient* algorithm for approximate model counting making use of the existing gadgets in the model counting literature. We recall that Zhao et al. [32] provided matching necessary and sufficient conditions on the required asymptotic density of matrix A. Furthermore, they proposed a hashing-based algorithm, SparseCount, that utilizes sparser constraints.

As mentioned earlier, Chakraborty et al. [9] proposed the technique of using *prefix-slicing* of hash functions in the context of hashing-based techniques and their empirical evaluation demonstrated significant theoretical and empirical improvements owing to the usage of prefix hashing. In this work, we first show a dramatic reduction in the complexity of SparseCount by utilizing the concept of *prefix-slicing* and thereby improving the number of SAT calls from $\mathcal{O}(n \log n)$ to $\mathcal{O}((\log n)^2)$ for fixed ε and δ The modified algorithm, called SparseCount2, significantly outperforms SparseCount, as demonstrated in Section 5.

Algorithm 1 shows the pseudo-code for SparseCount2. SparseCount2 assumes access to SAT oracle that takes in a formula φ and returns YES if φ is satisfiable, otherwise it returns NO. Furthermore, SparseCount2 assumes access to the subroutine MakeCopies that creates multiple copies of a given formula, a standard technique first proposed by Stockmeyer [28] to lift a constant factor approximation to that of $(1+\varepsilon)$-factor approximation for arbitrary ε. Similar to Algorithm

Algorithm 1 SparseCount2 $(\varphi, \varepsilon, \delta)$ ▷ Assume φ is satisfiable

```
 1: $\Delta \leftarrow 0.0042$
 2: $\psi \leftarrow \mathsf{MakeCopies}(\varphi, \lceil \frac{1}{\log_4(1+\varepsilon)} \rceil)$
 3: $m \leftarrow 0$; $iter \leftarrow 0$; $\mathcal{C} \leftarrow \mathsf{EmptyList}$; $\hat{n} \leftarrow |\mathsf{Vars}(\psi)|$
 4: $T \leftarrow \left\lceil \frac{\log(\hat{n}/\delta)}{\Delta} \right\rceil$
 5: $\{f_j\} \leftarrow \mathsf{ComputeSparseDensities}(\hat{n})$
 6: repeat
 7:     $iter \leftarrow iter + 1$
 8:     $m \leftarrow \mathsf{CoreSearch}(\psi,\ m,\ \{f_j\})$
 9:     $\mathsf{AddToList}(\mathcal{C},\ 2^m)$
10: until $iter < T$
11: $\hat{c} \leftarrow \mathsf{Median}(\mathcal{C})$
12: return $\hat{c}^{\lceil \log_4(1+\varepsilon) \rceil}$
```

Algorithm 2 CoreSearch(ψ, mPrev, $\{f_j\}$)

```
1: Choose $h$ uniformly at random from $H^{\{f_j\}}(\hat{n}, \hat{n})$
2: Choose $\alpha$ uniformly at random from $\{0,1\}^{\hat{n}}$
3: $Y \leftarrow \mathsf{SAT}(\psi \wedge h^{(\hat{n})}(\mathsf{Vars}(\psi)) = \alpha^{\hat{n}})$
4: if $Y$ is YES then
5:     return $\hat{n}$
6: $m \leftarrow \mathsf{LogSATSearch}(\psi, h, \alpha,$ mPrev$)$
7: return $m$
```

1 of [11], we choose $\{f_j\}$ in line 5, such that the resulting hash functions guarantee weak $(\mu_i^2, 9/4)$-concentration for the random variable $\mathsf{Cnt}_{\langle i \rangle}$ for all i, where $\mu_i = \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}]$. SparseCount2 shares similarity with SparseCount with the core difference in the replacement of linear search in SparseCount with the procedure CoreSearch. CoreSearch shares similarity with the procedure ApproxMC2Core of Chakraborty et al. [9]. The subroutine CoreSearch employs prefix search, which ensures that for all i, $\mathsf{Cnt}_{\langle i \rangle} \geq \mathsf{Cnt}_{\langle i+1 \rangle}$. The monotonicity of $\mathsf{Cnt}_{\langle i \rangle}$ allows us to perform a binary search to find the value of i for which $\mathsf{Cnt}_{\langle i \rangle} \geq 1$ and $\mathsf{Cnt}_{\langle i+1 \rangle} = 0$. Consequently, we make $\mathcal{O}(\log n)$ calls to the underlying NP oracle during each invocation of CoreSearch instead of $\mathcal{O}(n)$ calls in case of SparseCount. Note that CoreSearch is invoked T times, where $T = \left\lceil \frac{log(\hat{n}/\delta)}{\Delta} \right\rceil$ ($\hat{n}, \delta, \Delta$ as defined in the algorithm) and the returned value is added to the list $\mathcal{C}$. We then return the median of $\mathcal{C}$.

It is worth noting that SparseCount2 and ApproxMC3 differ only in the usage of *thresh*, which is set to 1 for SparseCount2 and a function of ε for ApproxMC3, as observed in the discussion following Lemma 1. The usage of *thresh* dependent on ε requires stronger bounds on variance, which can not be provided by sparse XORs as discussed in the previous section.

Algorithm 3 LogSATSearch(ψ, h, α, mPrev)

```
1: loIndex ← 0; hiIndex ← $\hat{n} - 1$; $m$ ← mPrev
2: BigCell[0] ← 1; BigCell[$\hat{n}$] ← 0
3: BigCell[$i$] ←⊥ ∀ $i \in [1, \hat{n} - 1]$
4: while true do
5:     $Y \leftarrow \mathsf{SAT}(\psi \wedge h^{(m)}(\mathsf{Vars}(\psi)) = \alpha^{(m)})$
6:     if $Y$ is YES then
7:         if BigCell[$m + 1$] = 0 then
8:             return $m + 1$
9:         BigCell[$i$] ← 1 ∀$i \in \{1, ...m\}$
10:        loIndex ← $m$
11:        if $|m - \text{mPrev}| < 3$ then
12:            $m \leftarrow m + 1$
13:        else if $2m < |\hat{n}|$ then
14:            $m \leftarrow 2m$
15:        else $m$ ← (hiIndex+$m$)/2
16:    else
17:        if BigCell[$m - 1$] = 1 then
18:            return $m$
19:        BigCell[$i$] ← 0 ∀ $i \in \{m, ...\hat{n}\}$
20:        hiIndex ← $m$
21:        if $|m - \text{mPrev}| < 3$ then $m \leftarrow m - 1$
22:        else $m$ ← (loIndex+$m$)/2
```

4.1 Analysis of Correctness of SparseCount2

We now present the theoretical analysis of SparseCount2. It is worth asserting that the proof structure and technique for SparseCount2 and ApproxMC3 are significantly different, as is evident from the inability of ApproxMC3 to use sparse XORs. Therefore, while the algorithmic change might look minor, the proof of correctness requires a different analysis.

Theorem 2. *Let* SparseCount2 *employ* $H^{\{f_j\}_{j=0}^n}$ *hash families, where* $\{f_j\}_{j=0}^n$ *is chosen such that it guarantees weak* $(\mu_i^2, 9/4)$*-concentration for the random variable* $\mathsf{Cnt}_{\langle i \rangle}$ *for all i, then* SparseCount2 *returns count c such that*

$$\mathsf{Pr}\left[\frac{|R_\varphi|}{1+\varepsilon} \leq c \leq (1+\varepsilon) \times |R_\varphi|\right] \geq 1 - \delta$$

Proof. Similar to [32], we assume that $|R_\varphi|$ is a power of 2; a relaxation of the assumption simply introduces a constant factor in the approximation. Let $|R_\psi| = 2^{i^*}$ and for we define the variable $\mathsf{Cnt}_{\langle i \rangle}^t$ to denote the value of $\mathsf{Cnt}_{\langle i \rangle}$ when iter $= t$. Let $\mu_i^t = \mathsf{E}[\mathsf{Cnt}_{\langle i \rangle}^t] = \frac{2^{i^*}}{2^i}$. Note that the choice of f_i ensures that $\mathsf{Cnt}_{\langle i \rangle}^t$ is weakly $((\mu_i^t)^2, 9/4)$ concentrated.

Let $\mathcal{E}$ denote the event that $\hat{c} > 4 \times |R_\psi|$ or $\hat{c} < \frac{|R_\psi|}{4}$. We denote the event $\hat{c} > 4 \times |R_\psi|$ as $\mathcal{E}_H$ and the event $\hat{c} < \frac{|R_\psi|}{4}$ as $\mathcal{E}_L$. Note that $\mathsf{Pr}[\mathcal{E}] = \mathsf{Pr}[\mathcal{E}_L] + \mathsf{Pr}[\mathcal{E}_H]$. We now compute $\mathsf{Pr}[\mathcal{E}_L]$ and $\mathsf{Pr}[\mathcal{E}_H]$ as follows:

1. From Algorithm 1, we have $\hat{c} = \text{Median}(\mathcal{C})$. For $\hat{c} < \frac{|R_\psi|}{4}$, we have that for at least $\frac{T}{2}$ iterations of CoreSearch returns $m < i^* - 2$. For t-th invocation of CoreSearch (i.e., $iter = t$) to return $m-1$, then it is necessarily the case that $\mathsf{Cnt}^t_{\langle m \rangle} = 0$. Since $\{f_j\}_{j=0}^n$ is chosen such that the resulting hash function guarantees $((\mu^t_m)^2, 9/4)$-concentration for the random variable $\mathsf{Cnt}^t_{\langle m \rangle}$, we have $\mathsf{Pr}[\mathsf{Cnt}^t_{\langle m \rangle} \geq 1] \geq 5/9$ for $m \leq i^* - 2$.
   Let us denote, by $\mathcal{E}^i_L$, the event that at least for $\frac{T}{2}$ of $\{\mathsf{Cnt}^t_{\langle i \rangle}\}_{t=0}^T$ we have $\mathsf{Cnt}^t_{\langle i \rangle} = 0$. Therefore, by Chernoff bound we have $\mathsf{Pr}[\mathcal{E}^i_L] \leq e^{-\nu^{(1)}T}$ where $\nu^{(1)} = 2(4/9 - 1/2)^2$. By applying union bound, we have $\mathsf{Pr}[\mathcal{E}_L] \leq ne^{-\nu^{(1)}T}$
2. Again, from the Algorithm 1, we have $\hat{c} = \text{Median}(\mathcal{C})$. Therefore, for $\hat{c} > 4 \times |R_\psi|$, we have at least $\frac{T}{2}$ invocations of CoreSearch return $m > i^* + 2$. For t-th invocation of CoreSearch (i.e., $iter = t$) to return m, then it is necessarily the case that $\mathsf{Cnt}^t_{\langle m-1 \rangle} \geq 1$.
   Noting, $\mathsf{E}[\mathsf{Cnt}^t_{\langle m \rangle}] = 2^{i^* - m}$. For $m \geq i^* + 2$, we have for $m \geq i^* + 2$
   $$\mathsf{Pr}[\mathsf{Cnt}^t_{\langle m \rangle} \geq 1] \leq 1/4.$$
   Let us denote by $\mathcal{E}^i_H$, the event that for at least $\frac{T}{2}$ of $\{\mathsf{Cnt}^t_{\langle i \rangle}\}_{t=0}^T$ values, we have $\mathsf{Cnt}^t_{\langle i \rangle} \geq 1$. By Chernoff bound for $m \geq i^* + 2$, we have $\mathsf{Pr}[\mathcal{E}^i_H] \leq e^{-\nu^{(2)}T}$ where $\nu^{(2)} = 2(1/4 - 1/2)^2$. By applying union bound, we have $\mathsf{Pr}[\mathcal{E}_H] \leq ne^{-\nu^{(2)}T}$.

Therefore, we have $\mathsf{Pr}[\mathcal{E}] = \mathsf{Pr}[\mathcal{E}_L] + \mathsf{Pr}[\mathcal{E}_H] \leq ne^{-\nu^{(1)}T} + ne^{-\nu^{(2)}T}$. Substituting T, we have

$$\mathsf{Pr}\left[\frac{|R_\psi|}{4} \leq \hat{c} \leq 4 \times |R_\psi|\right] \geq 1 - \delta.$$

Now notice that $|R_\psi| = |R_\varphi|^{\frac{1}{\log_4(1+\varepsilon)}}$; Therefore, $\frac{|R_\psi|}{4} \leq \hat{c} \leq 4 \times |R_\psi|$ ensures that we have $\frac{|R_\varphi|}{1+\varepsilon} \leq c \leq (1+\varepsilon) \times |R_\varphi|$. Therefore,

$$\mathsf{Pr}\left[\frac{|R_\varphi|}{1+\varepsilon} \leq c \leq (1+\varepsilon) \times |R_\varphi|\right] \geq 1 - \delta.$$

5 Empirical Studies

We focus on empirical study for comparison of runtime performance of SparseCount, SparseCount2, and ApproxMC3. All the three algorithms, SparseCount, SparseCount2, and ApproxMC3, are implemented in C++ and use the same underlying SAT solver, CryptoMiniSat [27] augmented with the BIRD framework introduced in [26,25]. CryptoMiniSat augmented with BIRD is state of the art SAT solver equipped to handle XOR constraints natively. *It is worth noting that for hashing-based techniques, over 99% of the runtime is consumed by the underlying* SAT

Benchmark (.cnf)	**Vars**	**Clauses**	**Time** (s) SparseCount	SparseCount2	ApproxMC3
blasted_case200	14	42	18.13	8.49	0.01
blasted_case60	15	35	350.48	23.43	0.01
s27_3_2	20	43	1581.63	30.28	0.01
SetTest.sk_9_21	33744	148948	1679.62	171.02	0.81
lss.sk_6_7	82362	259552	1959.39	405.61	1.63
registerlesSwap.sk_3_10	372	1493	2498.02	60.23	0.03
polynomial.sk_7_25	313	1027	2896.49	99.94	0.02
02A-3	5488	21477	3576.82	467.26	0.06
blasted_case24	65	190	TO	125.25	0.05
ConcreteActivityService.sk_13_28	2481	9011	TO	467.97	0.84
GuidanceService2.sk_2_27	715	2181	TO	498.14	0.29
ActivityService2.sk_10_27	1952	6867	TO	505.23	0.5
UserServiceImpl.sk_8_32	1509	5009	TO	511.09	0.33
or-100-10-4-UC-60	200	500	TO	608.86	0.05
02A-2	3857	15028	TO	1063.67	0.05
LoginService2.sk_23_36	11511	41411	TO	1127.36	2.96
17.sk_3_45	10090	27056	TO	1299.15	1.69
diagStencil.sk_35_36	319730	1774184	TO	2188.19	112.52
tableBasedAddition.sk_240_1024	1026	961	TO	TO	2.17
blasted_squaring9	1434	5028	TO	TO	5.04
blasted_TR_b12_1_linear	1914	6619	TO	TO	259.3

Table 1. Table of Comparison between SparseCount, SparseCount2, and ApproxMC3

solver [26]. Therefore, the difference in the performance of the algorithms is primarily due to the number of SAT calls and the formulas over which the SAT solver is invoked. Furthermore, our empirical conclusions do not change even using the older versions of CryptoMiniSat.

We conducted experiments on a wide variety of publicly available benchmarks. Our benchmark suite consists of 1896 formulas arising from probabilistic inference in grid networks, synthetic grid structured random interaction Ising models, plan recognition, DQMR networks, bit-blasted versions of SMTLIB benchmarks, ISCAS89 combinational circuits, and program synthesis examples. Every experiment consisted of running a counting algorithm on a particular instance with a timeout of 4500 seconds. The experiments were conducted on a high-performance cluster, where each node consists of E5-2690 v3 CPU with 24 cores and 96GB of RAM. We set $\varepsilon = 0.8$ and $\delta = 0.2$ for all the tools.

The objective of our empirical study was to seek answers to the following questions:

1. How does SparseCount compare against SparseCount2 in terms of runtime performance?
2. How does SparseCount2 perform against ApproxMC3 in terms of runtime?

Overall, we observe that SparseCount2 significantly outperforms SparseCount. On the other hand, ApproxMC3 outperforms SparseCount2 with a mean speedup of 568.53×.

Our conclusions are surprising and stand in stark contrast to the widely held belief that the current construction of sparse XORs by Zhao et al. [32] and Ermon et al. [11] lead to runtime improvement [1,20,19].

Figure 1 shows the cactus plot for SparseCount, SparseCount2, and ApproxMC3. We present the number of benchmarks on $x-$axis and the time taken on $y-$axis. A point (x, y) implies that x benchmarks took less than or equal to y seconds for the corresponding tool to execute. We present a runtime comparison of SparseCount2 vis-a-vis SparseCount and ApproxMC3 in Table 1. Column 1 of this table gives the benchmark name while column 2 and 3 list the number of variables and clauses, respectively. Column 4, 5, and 6 list the runtime (in seconds) of SparseCount, SparseCount2 and ApproxMC3, respectively. Note that "TO" stands for timeout. For lack of space, we present results only on a subset of benchmarks. The detailed logs along with list of benchmarks and the binaries employed to run the experiments are available at `http://doi.org/10.5281/zenodo.3792748`

We present relative comparisons separately for ease of exposition and clarity.

5.1 SparseCount vis-a-vis SparseCount2

As shown in Figure 1, with a timeout of 4500 seconds, SparseCount could only finish execution on 90 benchmarks while SparseCount2 completed on 379 benchmarks. Note that SparseCount2 retains the same theoretical guarantees of SparseCount.

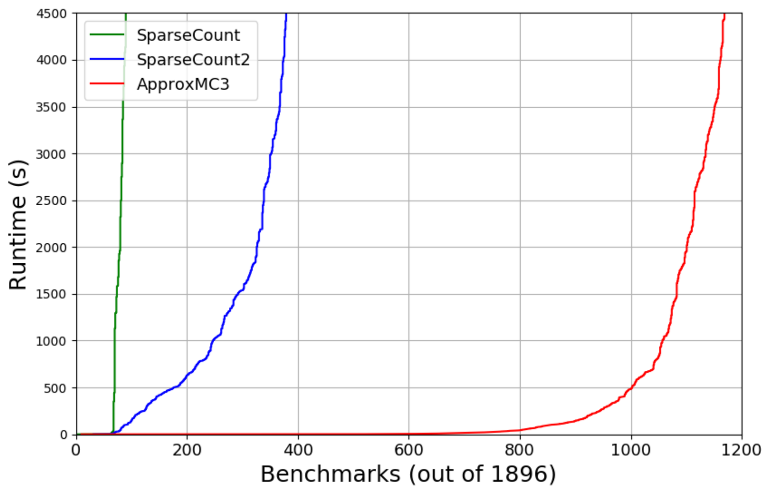

Fig. 1. Cactus plot of runtime performance (best viewed in color)

For a clear picture of performance gain achieved by SparseCount2 over SparseCount, we turn to Table 1. Table 1 clearly demonstrates that SparseCount2 outperforms SparseCount significantly. In particular, for all the benchmarks where both SparseCount and SparseCount2 did not timeout, the mean speedup is $10.94\times$.

Explanation The stark difference in the performance of SparseCount and SparseCount2 is primarily due to a significant reduction in the number of SAT calls in SparseCount2. Recall, SparseCount invokes the underlying SAT solver $\mathcal{O}(n \log n)$ times while SparseCount invokes the underlying SAT solver only $\mathcal{O}(\log^2 n)$ times. As discussed above, such a difference was achieved due to the usage of *prefix-slices*.

5.2 ApproxMC3 vis-a-vis SparseCount2

With a timeout of 4500 seconds, SparseCount2 could only finish execution on 379 benchmarks while ApproxMC3 finishes execution on 1169 benchmarks. Furthermore, Table 1 clearly demonstrates that ApproxMC3 significantly outperforms SparseCount2. In particular, for all the formulas where both SparseCount2 and ApproxMC3 did not timeout, the mean speedup is $568.53\times$. In light of recent improvements in CryptoMiniSat, one may wonder if the observations reported in this paper are mere artifacts of how the SAT solvers have changed in the past few years and perhaps such a study on an earlier version of CryptoMiniSat may have led to a different conclusion. To account for this threat of validity, we conducted preliminary experiments using the old versions of CryptoMiniSat and again observed that similar observations hold. In particular, the latest improvements in CryptoMiniSat such as BIRD framework [26,25] favor SparseCount and SparseCount2 relatively in comparison to ApproxMC3.

Explanation The primary contributing factor for the difference in the runtime performance of SparseCount2 and ApproxMC3 is the fact that weaker guarantees for the variance of $\mathsf{Cnt}_{\langle i \rangle}$ necessitates the usage of Stockmeyer's trick of usage of the *amplification technique* wherein the underlying routines are invoked over ψ instead of φ. Furthermore, the weak theoretical guarantees also lead to a larger value of T as compared to its analogous parameter in ApproxMC3. It is worth noticing that prior work on the design of sparse hash function has claimed that the usage of sparse hash functions leads to runtime performance improvement of the underlying techniques. Such inference may perhaps be drawn based only on observing the time taken by a SAT solver on CNF formula with a fixed number of XORs and only varying the density of XORs. While such an observation does indeed highlight that sparse XORs are easy for SAT solvers, but it fails, as has been the case in prior work, to take into account the tradeoffs due to the weakness of theoretical guarantees of sparse hash functions. To emphasize this further, the best known theoretical guarantees offered by sparse XORs are so weak that one can not merely replace the dense XORs with sparse XORs. The state of the art counters such as ApproxMC3 require stronger guarantees than those known today.

6 Conclusion

Hashing-based techniques have emerged as a promising paradigm to attain scalability and rigorous guarantees in the context of approximate model counting. Since the performance of SAT solvers was observed to degrade with an increase in the size of XORs, efforts have focused on the design of sparse XORs. In this paper, we performed the first rigorous analysis to understand the theoretical and empirical effect of sparse XORs. Our conclusions are surprising and stand in stark contrast to the widely held belief that the current construction of sparse XORs by Zhao et al. [32] and Ermon et al. [11] lead to runtime improvement. We demonstrate that the theoretical guarantees offered by the construction as mentioned earlier are still too weak to be employed in the state of the art approximate counters such as ApproxMC3. Furthermore, the most efficient algorithm using sparse XORs, to the best of our knowledge, still falls significantly short of ApproxMC3 in runtime performance. While our analysis leads to negative results for the current state of the art sparse construction of hash functions, we hope our work would motivate other researchers in the community to investigate the construction of efficient hash functions rigorously. In this spirit, concurrent work of Meel ⓡ Akshay [21] proposes a new family of hash functions called concentrated hash functions, and design a new family of sparse hash functions of the form $Ay+b$ wherein every entry of $A[i]$ is chosen with probability $p_i \in \mathcal{O}(\frac{\log n}{n})$. Meel ⓡ Akshay propose an adaption of ApproxMC3 that can make use of the newly designed sparse hash functions, and in turn, obtain promising speedups on a subset of benchmarks.

Acknowledgments The authors would like to sincerely thank the anonymous reviewers of AAAI-20 and SAT-20 (and in particular, Meta-Reviewer of AAAI-20) for providing detailed, constructive criticism that has significantly improved the quality of the paper. We are grateful to Mate Soos for many insightful discussions.

This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004] and AI Singapore Programme [AISG-RP-2018-005], and NUS ODPRT Grant [R-252-000-685-13]. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore https://www.nscc.sg.

References

1. Achlioptas, D., Hammoudeh, Z., Theodoropoulos, P.: Fast and flexible probabilistic model counting. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 148–164. Springer (2018)
2. Achlioptas, D., Theodoropoulos, P.: Probabilistic model counting with short xors. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 3–19. Springer (2017)
3. Asteris, M., Dimakis, A.G.: Ldpc codes for discrete integration. Tech. rep., Technical report, UT Austin (2016)

4. Baluta, T., Shen, S., Shinde, S., Meel, K.S., Saxena, P.: Quantitative verification of neural networks and its security applications. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1249–1264 (2019)
5. Biondi, F., Enescu, M., Heuser, A., Legay, A., Meel, K.S., Quilbeuf, J.: Scalable approximation of quantitative information flow in programs. In: Proc. of VMCAI (1 2018)
6. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. In: Proceedings of the ninth annual ACM symposium on Theory of computing. pp. 106–112. ACM (1977)
7. Chakraborty, S., Meel, K.S., Mistry, R., Vardi, M.Y.: Approximate probabilistic inference via word-level counting. In: Proc. of AAAI (2016)
8. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A scalable approximate model counter. In: Proc. of CP. pp. 200–216 (2013)
9. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In: Proc. of IJCAI (2016)
10. Duenas-Osorio, L., Meel, K.S., Paredes, R., Vardi, M.Y.: Counting-based reliability estimation for power-transmission grids. In: Proc. of AAAI (2 2017)
11. Ermon, S., Gomes, C.P., Sabharwal, A., Selman, B.: Low-density parity constraints for hashing-based discrete integration. In: Proc. of ICML. pp. 271–279 (2014)
12. Ermon, S., Gomes, C.P., Sabharwal, A., Selman, B.: Optimization with parity constraints: From binary codes to discrete integration. In: Proc. of UAI (2013)
13. Ermon, S., Gomes, C.P., Sabharwal, A., Selman, B.: Taming the curse of dimensionality: Discrete integration by hashing and optimization. In: Proc. of ICML. pp. 334–342 (2013)
14. Gomes, C.P., Hoffmann, J., Sabharwal, A., Selman, B.: Short xors for model counting: from theory to practice. In: Proc. of SAT. pp. 100–106 (2007), http://dl.acm.org/citation.cfm?id=1768142.1768155
15. Gomes, C.P., Hoffmann, J., Sabharwal, A., Selman, B.: Short XORs for Model Counting; From Theory to Practice. In: SAT. pp. 100–106 (2007)
16. Gomes, C.P., Sabharwal, A., Selman, B.: Model counting: A new strategy for obtaining good bounds. In: Proc. of AAAI. vol. 21, pp. 54–61 (2006)
17. Gomes, C.P., Sabharwal, A., Selman, B.: Model counting. In: Biere, A., Heule, M., Maaren, H.V., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 633–654. IOS Press (2009)
18. Ivrii, A., Malik, S., Meel, K.S., Vardi, M.Y.: On computing minimal independent support and its applications to sampling and counting. Constraints pp. 1–18 (2015). https://doi.org/10.1007/s10601-015-9204-z, http://dx.doi.org/10.1007/s10601-015-9204-z
19. Kuck, J., Dao, T., Zhao, S., Bartan, B., Sabharwal, A., Ermon, S.: Adaptive hashing for model counting. In: Conference on Uncertainty in Artificial Intelligence (2019)
20. Kuck, J., Sabharwal, A., Ermon, S.: Approximate inference via weighted rademacher complexity. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
21. Meel, K.S., Akshay, S.: Sparse hashing for scalable approximate model counting: Theory and practice. In: Proc. of LICS (2020)
22. Meel, K.S., Vardi, M., Chakraborty, S., Fremont, D.J., Seshia, S.A., Fried, D., Ivrii, A., Malik, S.: Constrained sampling and counting: Universal hashing meets sat solving. In: Proc. of Beyond NP Workshop (2016)

23. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence **82**(1), 273–302 (1996). https://doi.org/10.1016/0004-3702(94)00092-1, http://dx.doi.org/10.1016/0004-3702(94)00092-1
24. Sang, T., Beame, P., Kautz, H.: Performing bayesian inference by weighted model counting. In: Prof. of AAAI. pp. 475–481 (2005)
25. Soos, M., Gocht, S., Meel, K.S.: Accelerating approximate techniques for counting and sampling models through refined cnf-xor solving. In: Proceedings of International Conference on Computer-Aided Verification (CAV) (7 2020)
26. Soos, M., Meel, K.S.: Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In: Proceedings of AAAI Conference on Artificial Intelligence (AAAI)(1 2019) (2019)
27. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Proc. of SAT. pp. 244–257 (2009)
28. Stockmeyer, L.: The complexity of approximate counting. In: Proc. of STOC. pp. 118–126 (1983)
29. Toda, S.: On the computational power of PP and (+)P. In: Proc. of FOCS. pp. 514–519. IEEE (1989)
30. Trevisan, L.: Lecture notes on computational complexity. Notes written in Fall (2002), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.71.9877&rep=rep1&type=pdf
31. Valiant, L.: The complexity of enumeration and reliability problems. SIAM Journal on Computing **8**(3), 410–421 (1979)
32. Zhao, S., Chaturapruek, S., Sabharwal, A., Ermon, S.: Closing the gap between short and long xors for model counting. In: Proc. of AAAI (2016)