Sparse Hashing for Scalable Approximate Model Counting: Theory and Practice*

Kuldeep S. Meel ⓡ
School of Computing, National University of Singapore

S. Akshay
Dept of CSE, Indian Institute of Technology, Bombay

Abstract

Given a CNF formula F on n variables, the problem of model counting, also referred to as $\#SAT$, is to compute the number of models or satisfying assignments of F. Recent years have witnessed a surge of effort towards developing efficient algorithmic techniques that combine the classical strongly 2-universal hash functions (from [Stockmeyer 1983]) with the remarkable progress in SAT solving over the past decade. These techniques augment the CNF formula F with random XOR constraints and invoke an NP oracle repeatedly on the resultant CNF-XOR formulas. In practice, the NP oracle calls are replaced by calls to a SAT solver and it is observed that runtime performance of modern SAT solvers (based on conflict-driven clause learning) on CNF-XOR formulas is adversely affected by the size of XOR constraints. The standard construction of 2-universal hash functions chooses every variable with probability $p = \frac{1}{2}$ leading to XOR constraints of size $\frac{n}{2}$ in expectation. Consequently, the main challenge is to design *sparse* hash functions, where variables can be chosen with smaller probability and lead to smaller sized XOR constraints, which can then replace strongly 2-universal hash functions.

In this paper, our goal is to address this challenge both from a theoretical and a practical perspective. First, we formalize a relaxation of universal hashing, called concentrated hashing, a notion implicit in prior works to design sparse hash functions. We then establish a novel and beautiful connection between concentration measures of these hash functions and isoperimetric inequalities on boolean hypercubes. This allows us to obtain tight bounds on variance as well as the dispersion index and show that $p = O(\frac{\log_2 m}{m})$ suffices for the design of sparse hash functions from $\{0,1\}^n$ to $\{0,1\}^m$ belonging to the concentrated hash family. Finally, we use sparse hash functions belonging to this concentrated hash family to develop new approximate counting algorithms. A comprehensive experimental evaluation of our algorithm on 1893 benchmarks demonstrates that the usage of sparse hash functions can lead to significant speedups. To the best of our knowledge, this work is the first study to demonstrate runtime improvement of approximate model counting algorithms through the usage of sparse hash functions, while still retaining strong theoretical guarantees (à la 2-universal hash functions).

*ⓡ The ordering of authors was chosen randomly. The publicly verifiable record of the randomization is available at https://www.aeaweb.org/journals/policies/random-author-order/search.

ACM ISBN 978-1-4503-7104-9/20/07…$15.00
https://doi.org/10.1145/3373718.3394809

CCS Concepts: • **Theory of computation** → **Automated reasoning**; • **Computing methodologies** → *Knowledge representation and reasoning*.

Keywords: Model Counting, Sparse Hashing, SAT-Solving, Universal Hash Functions

ACM Reference Format:
Kuldeep S. Meel ⓡ and S. Akshay. 2020. Sparse Hashing for Scalable Approximate Model Counting: Theory and Practice. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3373718.3394809

1 Introduction

Given a Boolean formula F in conjunctive normal form (CNF), the problem of model counting, also referred to as #SAT, is to compute the number of models of F. Model counting is a fundamental problem in computer science with a wide variety of applications ranging from quantified information leakage [Fredrikson and Jha 2014], probabilistic reasoning [Chakraborty et al. 2014; Ermon et al. 2013; Roth 1996; Sang et al. 2005], network reliability [Valiant 1979], quantitative verification [Baluta et al. 2019] and the like. For example, given a probabilistic model describing conditional dependencies between different variables in a system, the problem of probabilistic inference, which seeks to compute the probability of an event of interest given observed evidence, can be reduced to a collection of model counting queries [Roth 1996].

In his seminal paper, Valiant showed that #SAT is #P-complete, where #P is the set of counting problems associated with NP decision problems [Valiant 1979]. Theoretical investigations of #P have led to the discovery of deep connections in complexity theory, and there is strong evidence for its hardness [Arora and Barak 2009; Toda 1989]. In particular, Toda showed that every problem in the polynomial hierarchy could be solved by just one call to a #P oracle; more formally, $PH \subseteq P^{\#P}$ [Toda 1989].

Given the computational intractability of #SAT, researchers have focused on approximate variants. Stockmeyer presented a randomized hashing-based technique that can compute (ε, δ) approximation within the polynomial time, in $|F|, \varepsilon, \delta$, given access to a NP oracle where $|F|$ is the size of formula, ε is the error tolerance bound and δ is the confidence[1]. The computational intractability of NP dissuaded development of algorithmic implementations of Stockmeyer's hashing-based techniques and no practical tools for approximate counting existed until the 2000's [Gomes et al. 2006]. By extending Stockmeyer's framework, Chakraborty, Meel, and Vardi demonstrate a scalable (ε, δ)-counting algorithm, ApproxMC [Chakraborty et al. 2013]. Subsequently, several new algorithmic ideas have been incorporated to demonstrate the scalability of ApproxMC; the current version of ApproxMC is called ApproxMC4 [Chakraborty et al. 2016a; Soos et al. 2020; Soos and Meel 2019]. Recent years have seen a surge of interest in the design of hashing-based techniques for approximate counting [Chakraborty et al. 2014, 2016b, 2013; Ermon et al. 2013; Ivrii et al. 2016; Meel et al. 2016; Soos et al. 2020; Soos and Meel 2019].

The core theoretical idea of the hashing-based framework is to employ strongly 2-universal hash functions to partition the solution space, denoted by $sol(F)$ for a formula F, into *roughly equal small* cells, wherein a cell is called *small* if it has solutions less than or equal to a pre-computed threshold, thresh. An NP oracle 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. To ensure polynomially many NP calls, thresh is set to be polynomial in input parameter ε. The choice of the threshold gives rise to a tradeoff between the number of NP queries and size of each query. To achieve probabilistic amplification of the confidence, multiple invocations of underlying subroutines are performed.

A standard family of strongly 2-universal hash functions employed for this is the H_{xor} family comprising of functions expressed as conjunction of XOR constraints. In particular, viewing the set of variables y of the formula F as a vector of dimension $n \times 1$, one can represent the hash function $h : \{0,1\}^n \mapsto \{0,1\}^m$ as $h(y) = \boldsymbol{A}y + b$ where $\boldsymbol{A}$ is a $m \times n$ matrix while b is $m \times 1$ 0-1 vector and each entry of $\boldsymbol{A}$ and b is either 0 or 1. Each entry of $\boldsymbol{A}$ is chosen to be 1 with probability $p = 1/2$, therefore the average number of 1's in each row is $\frac{n}{2}$. Each row of $h(y)$ thus gives rise to XOR constraints involving $\frac{n}{2}$ variables in expectation. Similarly a cell α can be viewed as a 0-1 vector of size $m \times 1$. Now, the solutions of F in a given cell α are the solutions of the formula $F \wedge (\boldsymbol{A}y + b = \alpha)$. As the input formula F is in CNF, this formula is a conjunction of CNF and XOR-constraints, also called an CNF-XOR formula. Given a hash function h and a cell α, the random variable of interest, denoted by $|\mathsf{Cell}_{\langle F,h,\alpha \rangle}|$ is the number of solutions of F that h maps to cell α. As mentioned earlier, the NP-oracle is invoked (polynomially many times) to check if such a cell is small.

The practical implementation of these techniques employ a SAT solver to perform NP oracle calls. The performance of SAT solvers, however, degrades with increase in the number of variables in XOR constraints (also called their *width*) and therefore recent efforts have focused on design of *sparse* hash functions where each entry is chosen with $p \ll 1/2$ (p is also referred to as density) [Achlioptas et al. 2018; Achlioptas and Theodoropoulos 2017; Asteris and Dimakis 2016; Ermon et al. 2013; Gomes et al. 2007; Ivrii et al. 2016]. The primary theoretical challenge is that 2-universality has been crucial to obtain (ε, δ)-guarantees, and sparse hash functions are not 2-universal. In fact, despite intense theoretical and practical interest in the design of sparse hash functions, the practical implementation of all prior constructions have had to sacrifice theoretical guarantees (as further discussed in Section 2.2).

Given the applications of counting to critical domains such as network reliability, the loss of theoretical guarantees limits the applications of approximate model counters. Therefore, in this context, the main challenge is: **Is it possible to construct sparse hash functions and design algorithmic frameworks to achieve runtime performance improvement without losing theoretical guarantees?**

In this paper, we address this challenge. To this end, we formalize the implicit observation in prior works that hashing-based counting algorithms, similar to other applications of universal hashing, are primarily concerned with the application of concentration bounds. We start by providing, in Section 2, a definition of concentrated hash functions, a relaxation of universal hashing. The guarantees offered by concentrated hashing depend crucially on the size of the set, unlike in universal hashing. Next, we turn towards the construction of sparse hash functions that belong to the concentrated hash family. Finally, we explain how these sparse hash functions can be used to build an efficient algorithm for approximate model counting. More precisely, the technical contributions of this paper are the following:

1. We first obtain a characterization of $sol(F)$ that would achieve the maximum variance as well as dispersion index for $|\mathsf{Cell}_{\langle F,h,\alpha \rangle}|$ for sparse hash functions. In a significant departure from earlier works [Achlioptas et al. 2018; Asteris and Dimakis 2016; Ermon et al.

[1]Although Stockmeyer did not present a randomized variant in his 1983 paper, Jerrum, Valiant, and Vazirani credit Stockmeyer for the idea [Jerrum et al. 1986]

2014; Zhao et al. 2016] where the focus was to use analytical methods to obtain upper bound on the variance of $|\mathsf{Cell}_{\langle F,h,\alpha\rangle}|$, we focus on searching for the set $sol(F)$ that would achieve the maximum variance of $|\mathsf{Cell}_{\langle F,h,\alpha\rangle}|$. To do this, we utilize a beautiful connection between the maximizing of variance as well as dispersion index of $|\mathsf{Cell}_{\langle F,h,\alpha\rangle}|$ and minimizing the "t-boundary" (the number of pairs with Hamming distance at most t) of sets on the boolean hypercube on n dimensions. This allows us to obtain novel and stronger upper bounds by using deep results from Boolean functional analysis and isoperimetric inequalities [Beame and Rashtchian 2017; Rashtchian 2018]. This connection could possibly be applied in other contexts as well.

2. Utilizing the connection between dispersion index and "t-boundary" allows us to introduce a new family of hash functions, denoted by $\mathcal{H}^k_{Rennes}$, which consists of hash functions of the form $\boldsymbol{A}y + b$, where every entry of $\boldsymbol{A}[i]$ is set to 1 with $p_i = O(\frac{\log_2 i}{i})$. The construction of the new family marks a significant departure from prior families in the behavior of the density dependent on rows of the matrix $\boldsymbol{A}$. We believe $\mathcal{H}_{Rennes}$ is of independent interest and can be substituted for 2-universal hash functions in several applications of hashing.
3. Finally, we use the above concentrated hash family to develop a new approximate model counting algorithm ApproxMC5, building on the existing state-of-the-art algorithm ApproxMC4. The primary challenge lies in the design and analysis of a hashing-based algorithm that does not assume any bound on $|sol(F)|$ but is able to use concentrated hash functions whose behavior depends on the size of the set being hashed. A comprehensive experimental evaluation on 1893 benchmarks demonstrates that usage of $\mathcal{H}_{Rennes}$ in ApproxMC5 leads to significant speedup in runtime over ApproxMC4. It is worth viewing the runtime improvement in the context of prior work where significant slowdown was observed. To the best of our knowledge, *this work is the first study to demonstrate runtime improvement through sparse hash functions without loss of $(\varepsilon, \delta)-$guarantees, demonstrating the tightness of our bounds in practice.*

Structure of the paper: We define notations and preliminaries in Section 2 along with a survey of state of the art for design of sparse hash functions in the context of approximate model counting. We then outline the main technical contributions of this paper in Section 3. In Section 4, we utilize deep results from Boolean functional analysis and isoperimetric inequalities to bound the dispersion index as well as variance of $|\mathsf{Cell}_{\langle F,h,\alpha\rangle}|$. We then use the bounds on dispersion index to construct sparse hash families belong to concentrated hashing in Section 5. Section 6 deals with construction of approximate model counting algorithm that uses hash functions belong to concentrated family. We finally describe extensive empirical evaluation in Section 7 and conclude in Section 8. Missing proofs and explanatory examples can be found in the long version at [Meel ⓡ Akshay 2020].

2 Definitions and State of the Art

The model counting problem. Let F be a Boolean formula in conjunctive normal form (CNF), and let $\mathsf{Vars}(F)$ be the set of variables appearing in F. The set $\mathsf{Vars}(F)$ is also called the *support* of F. An assignment σ of truth values to the variables in $\mathsf{Vars}(F)$ is called a *satisfying assignment* or *witness* of F if it makes F evaluate to true. We denote the set of all witnesses of F by $sol(F)$. Throughout, we will use n to denote $|\mathsf{Vars}(F)|$.

We write $\Pr[\mathcal{Z} : \Omega]$ to denote the probability of outcome $\mathcal{Z}$ when sampling from a probability space Ω. For brevity, we omit Ω when it is clear from the context. The expected value of $\mathcal{Z}$ is denoted $\mathsf{E}[\mathcal{Z}]$ and its variance is denoted $\sigma^2[\mathcal{Z}]$. The quantity $\frac{\sigma^2[\mathcal{Z}]}{\mathsf{E}[\mathcal{Z}]}$ is called the dispersion index of the random variable $\mathcal{Z}$. Given a distribution $\mathcal{D}$, we use $\mathcal{Z} \sim \mathcal{D}$ to denote that $\mathcal{Z}$ is sampled from the distribution $\mathcal{D}$. Let Bern(p) denote the Bernoulli distribution with probability p such that if $\mathcal{Z}$ ~Bern(p), we have $\Pr[\mathcal{Z} = 1] = p$.

The *propositional model counting problem* is to compute $|sol(F)|$ for a given CNF formula F. A *probably approximately correct* (or PAC) counter is a probabilistic algorithm $\mathsf{ApproxCount}(\cdot,\cdot,\cdot)$ that takes as inputs formula F, tolerance $\varepsilon > 0$, and a confidence $\delta \in (0, 1]$, and returns a (ε, δ)-estimate c, i.e., $\Pr\left[\frac{|sol(F)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|sol(F)|\right] \geq 1-\delta$. PAC guarantees are also referred to as (ε, δ)-guarantees. A closely related notion is of projected model counting wherein we are interested in computing the cardinality of $sol(F)$ projected to a subset of variables $\mathcal{P} \subseteq \mathsf{Vars}(F)$. While we focus on the problem of model counting, the techniques developed in this paper apply to projected model counting too. In our empirical evaluation, we consider such benchmarks as well.

Universal hash functions. Let $n, m \in \mathbb{N}$ and $\mathcal{H}(n,m) \triangleq \{h : \{0,1\}^n \to \{0,1\}^m\}$ be a family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$. We use $h \xleftarrow{R} \mathcal{H}(n,m)$ to denote the probability space obtained by choosing a function h uniformly at random from $\mathcal{H}(n,m)$. To measure quality of a hash function we focus on the set of elements of S mapped to α by h, denoted $\mathsf{Cell}_{\langle S,h,\alpha\rangle}$ and its cardinality, $|\mathsf{Cell}_{\langle S,h,\alpha\rangle}|$.

Definition 2.1. A family of hash functions $\mathcal{H}(n,m)$ is *strongly 2-universal* [2] if $\forall x, y \in \{0,1\}^n$, $\alpha \in \{0,1\}^m$, $h \xleftarrow{R} \mathcal{H}(n,m)$,

$$\Pr[h(x) = \alpha] = \frac{1}{2^m} = \Pr[h(x) = h(y)] \qquad (1)$$

[2]The concept of 2-universal hashing proposed by Carter and Wegman [Carter and Wegman 1977] only required that $\Pr[h(x) = h(y)] \leq \frac{1}{2^m}$ and therefore, the phrase *strongly 2-universal* is often used as also noted by Vadhan in [Vadhan et al. 2012].

Proposition 2.2. *Let $\mathcal{H}(n,m)$ be a strongly 2-universal hash family and let $h \xleftarrow{R} \mathcal{H}(n,m)$, then $\forall S \subseteq \{0,1\}^n$, $|S| \geq 1$,*

$$\mathsf{E}[|\mathsf{Cell}_{\langle S,h,\alpha\rangle}|] = \frac{|S|}{2^m} \tag{2}$$

$$\frac{\sigma^2[|\mathsf{Cell}_{\langle S,h,\alpha\rangle}|]}{\mathsf{E}[|\mathsf{Cell}_{\langle S,h,\alpha\rangle}|]} \leq 1 \tag{3}$$

Equation (3) can thus be restated as saying that for universal hash functions, the dispersion index is at most 1.

Prefix hash families. While universal hash families have nice concentration bounds, they are not adaptive, in the sense that one cannot build on previous queries. In several applications of hashing, the dependence between different queries can be exploited to extract improvements in theoretical complexity and runtime performance. Thus, we are typically interested in a restricted class of hash functions, called a *prefix-family* of hash functions defined in [Chakraborty et al. 2016a] as follows. For $\alpha \in \{0,1\}^m$, $\alpha[i]$ represent i-th element of α.

Definition 2.3. Let $n \in \mathbb{N}$ and $\mathcal{H}(n,1)$ be a family of hash functions. A family of hash functions $\mathcal{H}(n,n)$ is called a *prefix-family* with respect to $\mathcal{H}(n,1)$ if for all $h \in \mathcal{H}(n,n)$, there exists $h_1, h_2, \cdots h_n \in \mathcal{H}(n,1)$ such that

1. $h(x)[i] = h_i(x)$
2. for all $i \in [n]$, the probability spaces for $\{h_i \mid h \xleftarrow{R} \mathcal{H}(n,n)\}$ and $\{g \mid g \xleftarrow{R} \mathcal{H}(n,1)\}$ are identical.

For every $m \in \{1,\dots n\}$, the m^{th} prefix-slice of h, denoted $h^{(m)}$, is a map from $\{0,1\}^n$ to $\{0,1\}^m$, such that $h^{(m)}(y)[i] = h_i(y)$, for all $y \in \{0,1\}^n$ and for all $i \in \{1,\dots m\}$. Similarly, the m^{th} prefix-slice of α, denoted $\alpha^{(m)}$, is an element of $\{0,1\}^m$ such that $\alpha^{(m)}[i] = \alpha[i]$ for all $i \in \{1,\dots m\}$. In this paper we will primarily be focussed on prefix-hash functions and concentration bounds on them. To avoid cumbersome terminology, we abuse notation and write $\mathsf{Cell}_{\langle S,m\rangle}$ (resp. $\mathsf{Cnt}_{\langle S,m\rangle}$) as a short-hand for $\mathsf{Cell}_{\langle S,h^{(m)},\alpha^{(m)}\rangle}$ (resp. $|\mathsf{Cell}_{\langle S,h^{(m)},\alpha^{(m)}\rangle}|$).

Symbol	Short for	Meaning
$\mathsf{Cell}_{\langle S,m\rangle}$	$\mathsf{Cell}_{\langle S,h^{(m)},\alpha^{(m)}\rangle}$	$S \cap \{y \mid h^{(m)}(y) = \alpha^{(m)}\}$
$\mathsf{Cnt}_{\langle S,m\rangle}$	$|\mathsf{Cell}_{\langle S,h^{(m)},\alpha^{(m)}\rangle}|$	$|\mathsf{Cell}_{\langle S,m\rangle}|$

Table 1. List of Important Notations

In what follows, for a formula F, we write $\mathsf{Cell}_{\langle F,m\rangle}$ (resp. $\mathsf{Cnt}_{\langle F,m\rangle}$) to mean $\mathsf{Cell}_{\langle sol(F),m\rangle}$ (resp. $\mathsf{Cnt}_{\langle sol(F),m\rangle}$). Finally, the usage of prefix-family ensures monotonicity of the random variable, $\mathsf{Cnt}_{\langle S,i\rangle}$, since from the definition of prefix-family, we have that for all i, $h^{(i+1)}(x) = \alpha^{(i+1)} \implies h^{(i)}(x) = \alpha^{(i)}$. Formally,

Proposition 2.4. *For all $1 \leq i < m$, $\mathsf{Cell}_{\langle S,i+1\rangle} \subseteq \mathsf{Cell}_{\langle S,i\rangle}$*

Explicit families and sparse hash functions. While the above definitions of hash families are abstract, applications to model counting need explicit hash functions. The most common explicit hash family used for this are as follows: Let $\mathcal{H}_{\{p_i\}_{1\leq i\leq m}} \triangleq \{h : \{0,1\}^n \to \{0,1\}^m\}$ be the family of functions of the form $h(x) = \boldsymbol{A}x + b$ with $\boldsymbol{A} \in \mathbb{F}_2^{m\times n}$ and $b \in \mathbb{F}_2^{m\times 1}$ where the entries of $\boldsymbol{A}[i]$ and b are independently generated according to $\mathsf{Bern}(p_i)$ and $\mathsf{Bern}(\frac{1}{2})$ respectively. Note that taking $p_i = \frac{1}{2}$ gives $\mathcal{H}_{\{\frac{1}{2},\frac{1}{2},\cdots\frac{1}{2}\}}(n,m)$, which is precisely the strongly 2-universal hashing family proposed by Carter and Wegman [Carter and Wegman 1977], also denoted as $H_{xor}(n,m)$ in earlier works [Meel et al. 2016]. p_i is referred to as the density of i-th row of $\boldsymbol{A}$ and $1-p_i$ is referred to as the sparsity of i-th row of $\boldsymbol{A}$. We will use the term *sparse hash functions* to refer to hash functions with $p_i \ll \frac{1}{2}$.

Observe that $\mathcal{H}_{\{p_i\}_{1\leq i\leq n}}$ is a prefix-family with $h^{(m)}(x) = \boldsymbol{A}^{(m)}x + b^{(m)}$, where $\boldsymbol{A}^{(m)}$ denotes the submatrix formed by the first m rows and n columns of $\boldsymbol{A}$ and $b^{(m)}$ is the first m entries of the vector b.

2.1 Concentrated hash functions

Several applications such as sketching and counting [Cormode and Muthukrishnan 2005; Stockmeyer 1983] involving universal hash functions invoke strongly 2-universality property solely to obtain Proposition 2.2, i.e., obtain strong concentration bounds, but as mentioned above this requires fixing $p_i = \frac{1}{2}$.

In this context, one might ask if one can relax the requirement of 2-universality, while still attaining similar bounds for expectation and dispersion index. In a spirit similar to other attempts to design sparse hash functions for approximate counting techniques, we seek to design hash functions whose behavior depends on the size of $|S|$. To this end, we formalize the concept of concentrated hash family.

Definition 2.5. Let $\mathsf{qs}, k \in \mathbb{N}$, $\rho \in (0, 1/2]$. A family of hash functions $\mathcal{H}(n,n)$ is *prefix-(ρ,qs,k)-concentrated*, if for each m with $\mathsf{qs} \leq m \leq n$, and $S \subseteq \{0,1\}^n$ where $|S| \leq k \cdot 2^m$, $\alpha \in \{0,1\}^n$, $h \xleftarrow{R} \mathcal{H}$, we have

$$\mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}] = \frac{|S|}{2^m} \tag{4}$$

$$\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{\mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]} \leq \rho \tag{5}$$

It is easy to see that this definition is monotonic in k and it generalizes strongly 2-universal hash functions. Note that the above definition differs from the property of strongly 2-universal hash functions in two ways: first, it bounds the dispersion index by a constant instead of 1, and second, the definition depends on size of S.

Proposition 2.6. *If $\mathcal{H}(n,n)$ is prefix-(ρ, qs, k)-concentrated, then $\mathcal{H}(n,n)$ is prefix-$(\rho', \mathrm{qs}', k')$-concentrated for all $\rho' \geq \rho$, $\mathrm{qs}' \geq \mathrm{qs}$, and $k' \leq k$.*

Finally, applying the usual Chebyshev and Paley-Zymund inequalities to this definition immediately gives us the following properties of concentrated hash families.

Proposition 2.7. *If $\mathcal{H}$ is prefix-(ρ, qs, k)-concentrated family, then for every $0 < \beta < 1$, $\mathrm{qs} \leq m \leq n$, and for all $|S| \leq 2^m \cdot k$, we have the following:*

1. $\Pr\left[\left|\mathsf{Cnt}_{\langle S,m\rangle} - \mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]\right| \geq \beta \mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]\right] \leq \frac{\rho}{\beta^2 \mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]}$
2. $\Pr\left[\mathsf{Cnt}_{\langle S,m\rangle} \leq \beta \mathsf{E}\left[\mathsf{Cnt}_{\langle S,m\rangle}\right]\right] \leq \frac{\rho}{\rho + (1-\beta)^2 \mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]}$

Indeed, the rationale behind the design of (ρ, k)-concentrated hash families is that one can design such families with significant sparsity. Such sparse hash functions can then contribute to runtime performance of the underlying applications. The notion of concentrated hashing bears some similarity to the notion of *strongly concentrated* random variables defined in [Ermon et al. 2014]. In particular, a prefix (ρ, qs, k) concentrated family implies that the random variable $\mathsf{Cnt}_{\langle S,m\rangle}$, for $m \geq \mathrm{qs}$, is strongly-$\left((\beta \mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}])^2, \frac{\beta^2 \mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]}{\rho}\right)$ concentrated. We refer the reader to the technical report [Meel ⓡ Akshay 2020] for the formal statement as well as its relations to other useful notions of hashing.

2.2 State of the Art

The current state of the art hashing-based techniques for approximate model counting can be broadly classified into two categories: the first category of techniques [Achlioptas et al. 2018; Achlioptas and Theodoropoulos 2017; Ermon et al. 2013; Trevisan 2002], henceforth called Cat1, compute a constant factor approximation by setting thresh to be a constant and use Stockmeyer's trick of constructing multiple copies of the input formula. The second class of techniques, henceforth called Cat2, consists of techniques [Chakraborty et al. 2013, 2016a; Meel et al. 2016] that directly compute an (ε, δ)-estimate by setting threshold $= \mathcal{O}(\frac{1}{\varepsilon^2})$, and hence invoking the underlying NP oracle $\mathcal{O}(\frac{1}{\varepsilon^2})$ times. The proofs of correctness for all the hashing-based techniques involve the usage of concentration bounds due to strong 2-universal hash functions. The Cat1 techniques require the coefficient of variation, defined as the ratio of standard deviation of $\mathsf{Cnt}_{\langle F,m\rangle}$ to $\mathsf{E}[\mathsf{Cnt}_{\langle F,m\rangle}]$, to be upper bounded by a constant while, for Cat2 techniques, it is sufficient to have the dispersion index be bounded by a constant. It is worth noting that the analyses for both the techniques allow one to focus on the case of $\mathsf{E}[\mathsf{Cnt}_{\langle F,m\rangle}]$ being greater than 1. In this case, if dispersion index is upper bounded by a constant, then so is the coefficient of variation (but not vice versa!). In this sense, Cat2 techniques are stronger than Cat1.

Recently, [Asteris and Dimakis 2016] and [Zhao et al. 2016] independently showed that 2-universality can be relaxed while using Cat1 techniques. More precisely, they showed that choosing entries with probability $p = \mathcal{O}(\log n/n)$ asymptotically suffices to guarantee that the coefficient of variation is upper bounded by constant, i.e., dispersion index is upper bounded by mean of $\mathsf{Cnt}_{\langle F,m\rangle}$ when $\log(|sol(F)|) \in \Omega(n)$. Furthermore, [Achlioptas and Theodoropoulos 2017] showed that (sparse) hash functions constructed using LDPC codes also asymptotically suffice to guarantee that the coefficient of variation is upper bounded by constant. However, these results come with three caveats:

1. Only Cat1 techniques can employ these sparse hash functions as they can provide upper bound on coefficient of variation but not dispersion index. On the other hand, Cat2 techniques scale significantly better than Cat1 techniques in practice. [Agarwal et al. 2020]
2. The asymptotically large constant in the upper bound of coefficient of variation makes the practical usage usage of the above hash functions infeasible as discussed extensively in prior work (cf: Section 9 of [Achlioptas et al. 2018]).
3. The results only hold true for $\log(|sol(F)|) \in \Omega(n)$, which is usually not the case for many practical applications.

In summary, when $p < \frac{1}{2}$, previous techniques are unable to obtain a constant upper bound on the dispersion index and therefore do not yield to usage in Cat2 techniques (and hence in developing efficient practical algorithms for approximate model counting).

3 Main Results

To accomplish the design of scalable approximate counters via sparse hashing, we follow a three step recipe: (i) derive an expression to bound the dispersion index (of the random variable $\mathsf{Cnt}_{\langle S,m\rangle}$) via boolean functional analysis and isoperimetric inequalities, (ii) construct a sparse (ρ, k)-concentrated hash family and (iii) design an approximate model counter which can take advantage of concentrated hashing. In this section, we highlight our strategy, the core ideas involved and the main theorem statements.

3.1 Bounding the Dispersion Index

The first step is to obtain a closed form expression for the upper bound on dispersion index for an arbitrary set $S \subseteq \{0,1\}^n$. To this end, we focus on obtaining an expression that depends on n, $|S|$ and the range of hash function m for $h^{(m)}$.

For $1 \leq i \leq n-1$, $p_i \in (0, \frac{1}{2}]$, consider the family $\mathcal{H}_{\{p_i\}}(n,n) \triangleq \{h : \{0,1\}^n \rightarrow \{0,1\}^n\}$ of functions of the form $h(x) = \boldsymbol{A}x + \boldsymbol{b}$ with $\boldsymbol{A} \in \mathbb{F}_2^{n\times n}$ and $\boldsymbol{b} \in \mathbb{F}_2^{n\times 1}$ where the entries of $\boldsymbol{A}[i]$ (for $1 \leq i \leq n$) and $\boldsymbol{b}$ are independently generated according to $\mathrm{Bern}(p_i)$ and $\mathrm{Bern}(\frac{1}{2})$ respectively.

For $1 \le m \le n$, let

$$q(w,m) = \prod_{j=1}^{m} \left(\frac{1}{2} + \frac{1}{2}(1-2p_j)^w \right)$$

$$r(w,m) = q(w,m) - \frac{1}{2^m}$$

Note that $r(w,m)$ is a decreasing function of w for a fixed m. With this we have the following bound on the dispersion index, which is one of the main technical contributions of this paper, of possible independent interest.

Theorem 3.1. *For $1 \le m \le n$, $S \subseteq \{0,1\}^n$, $\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{\mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]} \le \sum_{w=0}^{\ell} 2 \cdot \left(\frac{8e\sqrt{n\cdot\ell}}{w}\right)^w r(w,m)$ where $\ell = \lceil \log |S| \rceil$.*

A key ingredient of the proof is to relate the dispersion index (and the variance) of $\mathsf{Cnt}_{\langle S,m\rangle}$ to the Hamming distance between nodes of S. This allows us to show that the dispersion index is in fact maximized for a nicely behaved set (formally, a left compressed down set as formalized in Section 4). Now we invoke deep results from boolean functional analysis and isoperimetric inequalities [Beame and Rashtchian 2017; Rashtchian 2018; Rashtchian and Raynaud 2019], to bound the maximum value of the dispersion index.

We remark that the best known bounds for the dispersion index from prior work so far has been: for any $S \subseteq \{0,1\}^n$, $\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{\mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]} \le \sum_{w=0}^{\ell} \binom{n}{w} q(w,m)$. Since $\left(\frac{8e\sqrt{n\cdot\ell}}{w}\right)^w \le \cdot \binom{8e\sqrt{n\cdot\ell}}{w}$, we obtain an improvement from $\binom{n}{w}$ to $2 \cdot \binom{8e\sqrt{n\cdot\ell}}{w}$. This improvement combined with our new analysis of the bounds leads us to design sparse hash family without incurring large overhead. It is also worth pointing out that prior work has always upper bounded $r(w,m)$ by $q(w,m)$ but as our analysis in the next section shows, we obtain stronger bounds on the dispersion index due to careful manipulation of $r(w,m)$.

3.2 Construction of Sparse Concentrated Hash Family

The upper bound on dispersion index provided by Theorem 3.1 depends on $|S|$, and we turn to the notion of concentrated family for construction of sparse hash functions to capture dependence on $|S|$. To bound the dispersion index, we seek to increase the rate of decrease of the values of $r(w,m)$ with respect to m. To this end, we propose a hash family with varying density across different rows of the matrix.

Definition 3.2. Let $k,n \in \mathbb{N}$ and let $H^{-1} : [0,1] \to [0,\frac{1}{2}]$ be the inverse binary entropy function restricting its domain to $[0,\frac{1}{2}]$ so that the inverse is well defined. We then define $\mathcal{H}^k_{Rennes}(n,n) \triangleq \{h : \{0,1\}^n \to \{0,1\}^n\}$ to be the family of functions of the form $h(x) = \boldsymbol{A}x + \boldsymbol{b}$ with $\boldsymbol{A} \in \mathbb{F}_2^{n\times n}$ and $\boldsymbol{b} \in \mathbb{F}_2^{n\times 1}$ where the entries of $\boldsymbol{A}[i]$ (for $1 \le i \le n$) and $\boldsymbol{b}$ are independently generated according to $\mathrm{Bern}(p_i)$ and $\mathrm{Bern}(\frac{1}{2})$ respectively, where $p_i \ge min(\frac{1}{2}, \frac{16}{H^{-1}(\delta)} \cdot \frac{\log_2 i}{i})$ for $\delta = \frac{i}{i+\log_2 k}$, and for $1 \le i \le n-1$, $p_i \ge p_{i+1}, p_i \in (0,\frac{1}{2}]$.

It is worth observing that $\mathcal{H}_{Rennes}$ marks a significant departure from prior families in the behavior of the density dependent on rows of the matrix $\boldsymbol{A}$. The sparsity of $\mathcal{H}_{Rennes}$ is discussed in detail Section 6.3 showing that for even small i, p_i can be set to values significantly smaller than $\frac{1}{2}$.

Theorem 3.3. *For $1 \le m \le n$, let $h \xleftarrow{R} \mathcal{H}^k_{Rennes}$, $S \subseteq \{0,1\}^n$, $\mathsf{Cell}_{\langle S,m\rangle} = \{y \in S \mid h^{(m)}(y) = \alpha^{(m)}\}$, $|S| \le 2^m k$ for some $\alpha \in \{0,1\}^m$. Then for every value of $k > 1$ and $\rho > 1$, there exists $\mathsf{qs} \le n$ such that for all m with $\mathsf{qs} \le m \le n$, we have*

$$E[\mathsf{Cnt}_{\langle S,m\rangle}] = \frac{|S|}{2^m} \tag{6}$$

$$\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{E[\mathsf{Cnt}_{\langle S,m\rangle}]} \le \rho \tag{7}$$

Corollary 3.4. *$\mathcal{H}^k_{Rennes}$ is prefix-(ρ, qs, k)-concentrated.*

The proof begins with the expression in Theorem 3.1 and is based on analysis of dispersion index by considering separate cases for different sets of values of w. The case analysis especially for large values of w turns out to be rather technical and uses properties of distribution of binomial coefficients and Taylor expansion of $r(w,m)$, as detailed in Section 5.

3.3 Approx Model Counting using Concentrated Hashing

As noted in Section 2, the usage of (ρ, qs, k)-concentrated family does present the challenge of identification of application domains where such hash functions suffice. Typical usage of hash functions does not put restrictions on the size of the underlying set S whose elements are being hashed. For example, the standard proofs of hashing-based counting techniques employ hash functions in the context where there is no reasonable upper bound on $|S|$. Therefore, one wonders whether it is possible to design hashing-based counting techniques which can use concentrated hash functions without assuming an upper bound on $|S|$.

We answer the above question positively in the third and final technical contribution of this paper with the design of approximate model counter with rigorous (ε,δ) guarantees ApproxMC5, which employs a prefix $(\rho, \mathsf{qs}, \mathsf{pivot})$-concentrated hash family instead of a strongly 2-universal hash family.

Theorem 3.5. *For input formula F, tolerance parameter ε, confidence parameter δ, and concentrated hashing parameters ρ and qs, suppose $\mathsf{ApproxMC5}(F,\varepsilon,\delta,\rho)$ uses a prefix $(\rho, \mathsf{qs}, \mathsf{pivot})$-concentrated hash family with the value of $\mathsf{pivot} = 78.72 \cdot \rho (1+\frac{1}{\varepsilon})^2$ and returns an estimate c. Then, $\Pr\left[\frac{|sol(F)|}{1+\varepsilon} \le c \le (1+\varepsilon)|sol(F)|\right] \ge 1-\delta$. Furthermore, ApproxMC5 makes $\mathcal{O}(2^{\mathsf{qs}+3} + \frac{\log(n)\log(1/\delta)}{\varepsilon^2})$ calls to a SAT-oracle.*

ApproxMC5 builds on the earlier algorithm ApproxMC4 [Chakraborty et al. 2016a; Soos and Meel 2019], but differs in the crucial use of a sparse hash family instead of a 2-universal hash family. This essentially requires us to rework the entire theoretical guarantees, which we do in Section 6.

Finally, in Section 7, we evaluate the performance of ApproxMC5 using the sparse hash functions belonging to prefix $(1.1, 1, \text{pivot})$-concentrated hash family and demonstrate that it leads to significant speedup in runtime over ApproxMC4. To the best of our knowledge, this work is the first study to demonstrate runtime improvement using sparse hash functions without loss of (ε, δ)–guarantees.

4 Bounding the dispersion index

In this section, we prove Theorem 3.1. Recall that for $1 \leq i \leq n-1$, $p_i \in (0, \frac{1}{2}]$, $\mathcal{H}_{\{p_i\}}(n,n) \triangleq \{h : \{0,1\}^n \to \{0,1\}^n\}$ denotes the family of functions of the form $h(x) = \boldsymbol{A}x + \boldsymbol{b}$ with $\boldsymbol{A} \in \mathbb{F}_2^{n\times n}$ and $\boldsymbol{b} \in \mathbb{F}_2^{n\times 1}$ where the entries of $\boldsymbol{A}[i]$ (for $1 \leq i \leq n$) and $\boldsymbol{b}$ are independently generated according to $\text{Bern}(p_i)$ and $\text{Bern}(\frac{1}{2})$ respectively. Our first step is to compute the mean and bound the variance of $\mathsf{Cnt}_{\langle S,m\rangle}$. We start with a known result and a definition.

Lemma 4.1. *[Asteris and Dimakis 2016; MacKay 1999] For all $\tau \in \{0,1\}^n$, we have $Pr(\boldsymbol{A}^{(m)}\tau = \boldsymbol{0}) = q(w,m)$ where $w = w(\tau)$ is the Hamming weight of τ (note that $0^0 = 1$).*

We define $c_S(w,x) = |\{y \mid y \in S, d(x,y) = w\}|$, i.e., the number of vectors in S that are at a Hamming distance of w from x. We also define $c_S(w) = |\{(x,y) \mid x \in S, y \in S, d(x,y) = w\}|$, i.e., the number of pairs of vectors in S that are at Hamming distance w from each other. Then we immediately obtain the following proposition (see [Meel ⓡ Akshay 2020] for details).

Proposition 4.2. *The following expressions hold:*

1. $\mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}] = \frac{|S|}{2^m}$
2. $\sum_{y_1,y_2\in S} Pr[h^{(m)}(y_1) = h^{(m)}(y_2) = \alpha^{(m)}]$ $= 2^{-m}\sum_{x\in S}\sum_{w=0}^{n} c_S(w,x)q(w,m)$

Then, we may express the variance in terms of $c_S(w)$ and $r(w,m)$.

Lemma 4.3. $\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}] = \frac{\sum_{w=0}^{n} c_S(w)r(w,m)}{2^m}$

Proof. $\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}] = \sum_{y_1,y_2\in S} Pr[h^{(m)}(y_1) = h^{(m)}(y_2) = \alpha^{(m)}] - (\sum_{y\in S} Pr[h^{(m)}(y) = \alpha^{(m)}])^2$

$$= 2^{-m}\sum_{x\in S}\sum_{w=0}^{n} c_S(w,x)q(w,m) - \sum_{x\in S}\sum_{y\in S}\frac{1}{2^{2m}}$$

$$= 2^{-m}\sum_{x\in S}\sum_{w=0}^{n} c_S(w,x)q(w,m) - \sum_{x\in S}\sum_{w=0}^{n}\frac{c_S(w,x)}{2^{2m}} \quad (8)$$

$$= 2^{-m}\sum_{x\in S}\sum_{w=0}^{n} c_S(w,x)r(w,m)$$

Earlier works on bounding σ^2 observed that $c_S(w,x) \leq \binom{n}{w}$ and focused their efforts to bound the resulting expression. Interestingly, the following seemingly simple rewriting allows us to explore interesting bounds for σ^2. We rewrite Eq 8 as

$$\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}] = 2^{-m}\sum_{w=0}^{n} c_S(w)r(w,m) \quad (9)$$

where $c_S(w)$ is the number of pairs of vectors in S that are at Hamming distance w from each other. □

Next for all $m \in \{1,\ldots,n\}$ and every $S \subseteq \{0,1\}^n$ we use deep results from boolean functional analysis to bound the dispersion index, $\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{\mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]}$, as a function of $|S|$ and $r(w,m)$. We start by setting up some notation. For $x,y \in \{0,1\}^n$, we say $y \subseteq x$ whenever for all $i \in [n]$, $y_i = 1 \implies x_i = 1$. We say $S \subseteq \{0,1\}^n$ is a *down-set* if for all $x,y \in \{0,1\}^n$, $x \in S, y \subseteq x$ implies $y \in S$. We say S is *left-compressed* if, for all $x,y \in \{0,1\}^n$, $x \in S$ implies $y \in S$ whenever y satisfies the two conditions (1) $|x| = |y|$ and (2) $x \succcurlyeq_{lex} y$, i.e., x is lexicographically larger than y. For example, the set $\{000, 001, 100\}$ is a downset but it is not left compressed, while $\{000, 001, 010\}$ is both a downset and left-compressed.

In [Rashtchian 2018], it is shown that among all sets S of the same cardinality, for all $k \in [n]$, $\sum_{w=0}^{k} c_S(w)$ achieves its maximum value for some left-compressed and down set S. We extend this to obtain the following crucial lemma.

Lemma 4.4. *Let n be positive integer and and let $t : [n] \to \mathbb{R}^+$ be a monotonically non-increasing function. Among all subsets S of $\{0,1\}^n$ of same cardinality, the sum $\sum_{w=0}^{n} c_S(w)t(w)$ achieves its maximum value for some left-compressed and down set S.*

The above lemma allows us to use the expressions obtained for $c_S(w)$ by Rashtchian in [Rashtchian 2018; Rashtchian and Raynaud 2019].

Lemma 4.5. *[Rashtchian 2018; Rashtchian and Raynaud 2019] For a left-compressed and down set S, $c_S(w) \leq 2 \cdot \left(\frac{8e\sqrt{n\cdot\ell}}{w}\right)^w \cdot |S|$ where $\ell = \lceil\log|S|\rceil$.*

Proof. The proof is based on the bounds derived by Rashtchian in [Rashtchian 2018]. We give a few more details as we will need them later when we explain our implementation. More specifically, the proof uses Equations 4.2, 4.5, 4.8, and 4.10 from [Rashtchian 2018]. It is crucial to note that these equations hold only for a left-compressed and down set and not for an arbitrary set S. The proof follows by breaking into two cases based on the parity of w.

For even w, Rashtchian upper bounds the expressions obtained in Eq. 4.2 and 4.5 by Eq 4.8 in [Rashtchian 2018]. We rewrite Eq 4.8 by substituting $2t$ by w to obtain $c_S(w) \leq$

$2 \cdot \left(\frac{8e\sqrt{n\cdot\ell}}{w}\right)^w$. For odd w, Rashtchian upper bounds the upper bound for $c_S(w)$ obtained in Eq. 4.2 and 4.5 by Eq 4.10. We rewrite Eq 4.10 by noting that $w = 2t + 1$ to obtain $c_S(w) \leq 2 \cdot \left(\frac{8e}{w}\right)^w (\sqrt{n\cdot\ell})^{(w-1)}\ell$. Noting that $\ell \leq \sqrt{n\cdot\ell}$, we have $c_S(w) \leq 2 \cdot \left(\frac{8e\sqrt{n\cdot\ell}}{w}\right)^w$. Thus, combining these cases, we get our lemma. □

Thus, for any $S \subseteq \{0,1\}^n$ let us fix $\ell = \lceil \log|S| \rceil$. Then, $\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{\mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]} \leq \sum_{w=0}^{\ell} 2 \cdot \left(\frac{8e\sqrt{n\cdot\ell}}{w}\right)^w r(w,m)$, which completes the proof of our first main result, Theorem 3.1, i.e.,

Theorem 3.1. *For* $1 \leq m \leq n$, $S \subseteq \{0,1\}^n$, $\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{\mathsf{E}[\mathsf{Cnt}_{\langle S,m\rangle}]} \leq \sum_{w=0}^{\ell} 2 \cdot \left(\frac{8e\sqrt{n\cdot\ell}}{w}\right)^w r(w,m)$ *where* $\ell = \lceil \log|S| \rceil$.

This theorem gives a closed form expression for upper bound on dispersion index, which is amenable to numerical computations. In particular, given ℓ, one can compute the value of p_i's such that dispersion index is upper bounded by a constant. Next, we analyze the behavior of p_i's for a given upper bound on dispersion index and we construct concentrated hash functions based on their behavior.

5 A concentrated hash family

In this section, we finally construct a family of concentrated hash functions, which proves our second main Theorem 3.3, which we restate below.

Theorem 3.3. *For* $1 \leq m \leq n$, *let* $h \xleftarrow{R} \mathcal{H}^k_{Rennes}$, $S \subseteq \{0,1\}^n$, $\mathsf{Cell}_{\langle S,m\rangle} = \{y \in S \mid h^{(m)}(y) = \alpha^{(m)}\}$, $|S| \leq 2^m k$ *for some* $\alpha \in \{0,1\}^m$. *Then for every value of* $k > 1$ *and* $\rho > 1$, *there exists* $\mathsf{qs} \leq n$ *such that for all* m *with* $\mathsf{qs} \leq m \leq n$, *we have*

$$E[\mathsf{Cnt}_{\langle S,m\rangle}] = \frac{|S|}{2^m} \tag{6}$$

$$\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{E[\mathsf{Cnt}_{\langle S,m\rangle}]} \leq \rho \tag{7}$$

Proof. The first equation follows from Proposition 4.2. For the second, from Theorem 3.1 we have, for any $1 \leq m \leq n$,

$$\begin{aligned}\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{E[\mathsf{Cnt}_{\langle S,m\rangle}]} &= 1 + \sum_{w=1}^{\ell} c_S(w) r(w,m) \\ &\leq 1 + \sum_{w=1}^{\ell} 2 \cdot \left(\frac{8e\sqrt{n\ell}}{w}\right)^w r(w,m),\end{aligned}$$

where $\ell = \lceil m + \log_2(k) \rceil$.

Note that $\frac{m}{\delta} + 1 \geq \ell \geq \frac{m}{\delta}$. Note that

$$q(w,m) = \prod_{j=1}^{m}\left(\frac{1}{2} + \frac{1}{2}(1-2p_j)^w\right) \leq \left(\frac{1}{2} + \frac{1}{2}(1-2p_m)^w\right)^m$$

Now let us define $f(w) = \left(\frac{8e\sqrt{n\ell}}{w}\right)^w r(w,m)$. Then, we can divide into three cases:

Case 1: $1 \leq w \leq (2p_m)^{-1}$
We have $\log(r(w,m)) \leq -mwp_m$. To see this, following the reasoning from [Asteris and Dimakis 2016], we have when $w \leq \frac{1}{2p_m}$,

$$\begin{aligned}\log(r(w,m)) &\leq -m + m\log(1 + (1-2p_m)^w) \\ &\leq -m + m\log(1 + e^{-2p_m w}) \\ &\leq -m + m(1 - p_m w)\end{aligned}$$

where the last inequality follows from the fact that $\log_2(1 + e^{-x}) \leq 1 - \frac{1}{2}x$ for $0 \leq x \leq 1$ and that $0 \leq 2p_m w \leq 1$ in this interval. Thus

$$\log_2 f(w) \leq w\log 8e\sqrt{n\ell} - w\log w - mp_m w \tag{10}$$

Since $H^{-1}(\delta) \leq \delta/2$, we have $p_m \geq \frac{16}{H^{-1}(\delta)}\frac{\log m}{m} \geq \frac{32}{\delta}\frac{\log m}{m} \geq 32\frac{\log m}{m}$, since $\delta \leq 1$. Therefore

$$\begin{aligned}&\log_2 f(w) \leq w\log 8e\sqrt{n\ell} - w\log w - mp_m w \\ &\leq w\log 8e\sqrt{n\ell} - mp_m w \leq w\log 8e\sqrt{n\ell} - 32w\log m \\ &\leq w\log\frac{8e\sqrt{nl}}{m^{32}}\end{aligned}$$

Now, we pick $\mathsf{qs}_1 > (\sqrt{n}\frac{2\rho}{\rho-1}(\ell)^{3/2}8e)^{1/32}$. Note that this is possible, since $\ell \leq n$ and it suffices to choose $\mathsf{qs}_1 > 1.12(\frac{2\rho}{\rho-1})^{1/32}n^{1/16}$ which is in turn possible for any value of $\rho > 1$.

Then, we have for any $m \geq \mathsf{qs}_1$, $m^{32} > 8e\sqrt{n}(\frac{2\rho}{\rho-1})\ell^{3/2}$ which implies that $\frac{8e\sqrt{nl}}{m^{32}} < \frac{\rho-1}{2\rho\ell}$. Then we have

$$\log_2 f(w) \leq w\log\frac{8e\sqrt{nl}}{m^{32}} \leq w\log\frac{\rho-1}{2\rho\ell} \leq 1\cdot\log\frac{\rho-1}{2\rho\ell} \tag{11}$$

where the last inequality follows because, $\ell \geq 1$ (since $|S| \geq 2$), which means that $\log\frac{\rho-1}{2\rho\ell} < 0$.

Therefore, $\sum_{w=1}^{\ell} f(w) \leq \frac{\ell(\rho-1)}{2\rho\ell}$

$$\implies \frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{E[\mathsf{Cnt}_{\langle S,m\rangle}]} \leq 1 + 2\frac{\rho-1}{2\rho} < 1 + \rho - 1 = \rho$$

$$\implies \text{for } k > 1, \rho > 1, \text{for } \mathsf{qs}_1 \leq m \leq n, \frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{E[\mathsf{Cnt}_{\langle S,m\rangle}]} < \rho$$

Case 2: $(2p)^{-1} \leq w \leq \frac{mH^{-1}(\delta)}{16}$
We start by observing that $g(w) = \log f(w) = w\log 8e\sqrt{n\ell} - w\log w$ is increasing in the interval $w = 0$ to $w = \ell$. To see this, consider the derivative $g'(w) = \log(\frac{8e\sqrt{n\ell}}{w}) - 1$. Then $w \leq \ell \leq 8e\sqrt{n\ell}$ implies $2 \leq \frac{8e\sqrt{n\ell}}{w}$, which implies $g'(w) > 0$.

Now $w \leq \frac{mH^{-1}(\delta)}{16} \leq \frac{m\delta}{32} \leq \frac{m}{32}$ since $\delta \leq 1$ and $H^{-1}(\delta) \leq \frac{\delta}{2}$. Thus we have,

$$\log f(w) \leq \log f(\frac{m}{32}) \leq \frac{m}{32}\log(\frac{2^9 e\sqrt{n\ell}}{m})$$
$$= \frac{9m}{32} + m\log\left(\frac{e\sqrt{n\ell}}{m}\right)^{\frac{1}{32}}$$

Now we pick $m > \frac{e\sqrt{n\ell}}{40}$. Then we get $\log f(w) \leq 0.282m + 0.167m \leq 0.45m$.

On the other hand, we have $\log r(w,m) \leq -m + m\log(1 + exp(-2p_m w)) \leq -m + m\log(1 + exp(-1)) \leq -0.58m$

Thus, we get $\frac{\sigma^2[\mathsf{Cnt}_{\langle S,m\rangle}]}{E[\mathsf{Cnt}_{\langle S,m\rangle}]} \leq 1 + \sum_{w=1}^{\ell} 2 \cdot f(w)r(w,m) \leq 1 + \sum_{w=1}^{\ell} 2^{1-0.13m} \leq 1 + \frac{2\ell}{2^{0.13m}}$ Thus there exists $\mathsf{qs}_2 = \frac{1}{0.13}\log_2\left(\frac{2\ell}{\rho-1}\right)$ such that for $\mathsf{qs}_2 \leq m \leq n$, clearly this can be made less than any constant $\rho > 1$.

Case 3: $w \geq \frac{mH^{-1}(\delta)}{16}$. We start with a claim, the proof for which can be found in [Meel ⓡ Akshay 2020], namely: for $m \geq 2$, if $w \geq \frac{mH^{-1}(\delta)}{16}$, then $\log_2 r(w,m) < -m + 1 - \log_2 m$. From this, we obtain $\log_2 r(w,m) \leq -m + 1 - \log_2 m$, i.e., $r(w,m) \leq \frac{2\cdot 2^{-m}}{m}$. Also, recalling that we have $\Sigma_{w=1}^{\ell} c_s(w) \leq 2^{\ell}$, we obtain $\frac{\sigma^2}{\mu} \leq 1 + \sum_{w=1}^{\ell} c_s(w) max_w r(w,m) \leq 1 + 2^{\ell} \cdot \frac{2\cdot 2^{-m}}{m} = 1 + \frac{2k}{m}$. Thus for all k, we can pick $\mathsf{qs}_3 = \frac{2k}{\rho-1}$ such that for any $\mathsf{qs}_3 \leq m \leq n$ and $\rho > 1$, $\frac{\sigma^2}{\mu} \leq \rho$.

Combining the three cases and taking $\mathsf{qs} = \max\{\mathsf{qs}_1, \mathsf{qs}_2, \mathsf{qs}_3\}$, we obtain our desired result. It is worth noting that the smallest value of m (i.e., qs) for which Theorem 3.3 holds true depends on ρ and k. Furthermore, it is interesting to observe that the proof of Case 3 crucially depends on usage of $r(w,m)$ instead of $q(w,m)$ in the expression of $\frac{\sigma^2}{\mu}$ as the current proof techniques would only yield $\log_2 q(w,m) < -m + 1$, which would be insufficient to prove $\frac{\sigma^2}{\mu} \leq \rho$. □

6 A New Approximate Model Counting Algorithm: ApproxMC5

In this section, we seek to design algorithms that can use (ρ, qs, k)-concentrated hash functions for a small k, independent of the problem instance. In particular, we first revisit the state of the art approximate counting algorithm ApproxMC. We will refer to the algorithmic constructs presented in ApproxMC2 [Chakraborty et al. 2016a] since the subsequent versions, i.e., ApproxMC3 and ApproxMC4, have proposed algorithmic improvement to the underlying SAT calls only. We seek to modify ApproxMC2 so as to employ concentrated hash function; the final implementation of ApproxMC5 builds on top of ApproxMC4, allowing it to benefit from the improvements proposed in ApproxMC3 and ApproxMC4.

6.1 The Algorithm

The subroutine ApproxMC5 is presented in Algorithm 1. ApproxMC5 takes in a formula F, tolerance: ε, and confidence parameter δ, concentrated hashing parameters ρ and qs as input and returns an estimate of $|sol(F)|$ within tolerance ε and confidence at least $1 - \delta$. Similar to ApproxMC2, the key idea of ApproxMC5 is to partition the solution space of F into *roughly equal small* cells of solutions such that the $|sol(F)|$ can be estimated from the number of solutions in a randomly chosen cell scaled by the total number of cells. This idea requires two crucial ingredients:

1. hash functions to achieve desired properties of partitioning: As has been emphasized earlier, in this work, we mark a departure from prior work and employ concentrated hash functions instead of strongly 2-universal hash functions.
2. subroutine to check whether a cell is small, i.e., the number of solutions in the cell is less than an appropriately computed thresh. ApproxMC5 assumes access to the subroutine BoundedCount that takes in a formula F and a threshold thresh and returns an integer Y, such that $Y = \min(\mathsf{thresh}, |sol(F)|)$. Note that $Y = \mathsf{thresh}$ is used to indicate that the number of solutions is greater than or equal to thresh, which indicates that the cell is not *small.* We do not treat BoundedCount as an oracle in our analysis and instead as a subroutine which uses a NP oracle to enumerate solutions of F one by one until we have found thresh number of solutions or there are no more solutions. As such for BoundedCount to make polynomially many calls to NP oracle, thresh is polynomial in $\frac{1}{\varepsilon}$.
3. Subroutine, called LogSATSearch, to search for the right number of cells as discussed in detail below.

ApproxMC5 differs from ApproxMC2 primarily in the computation of thresh and usage of concentrated hash functions – the two critical components that distinguish several hashing-based counting techniques. The computation of thresh involves the parameter ρ to account for concentrated hashing and incurs an overhead proportional to ρ. As discussed later, for our empirical studies, we set ρ to 1.1. Unlike prior techniques, we introduce another parameter iniThresh that depends on thresh and qs to account for qs parameter of concentrated hash functions. ApproxMC5 first checks if the number of solutions of F is less than iniThresh and upon passing the check it simply returns the number of solutions of F. For interesting instances, the check fails and ApproxMC5 invokes the subroutine ApproxMC5Core t times and computes the median of the returned estimates by ApproxMC5Core.

The subroutine ApproxMC5Core lies at the core of ApproxMC5 and shares similarity with ApproxMC2Core employed in [Chakraborty et al. 2016a]. In contrast to ApproxMC2Core, the algorithmic description does not restrict the hash family

Algorithm 1 ApproxMC5($F, \varepsilon, \delta, \rho$, qs)

```
1: thresh ← $1 + 9.84 \cdot \rho \cdot \left(1 + \frac{\varepsilon}{1+\varepsilon}\right)\left(1 + \frac{1}{\varepsilon}\right)^2$;
2: iniThresh ← thresh $* 2^{\text{qs}+3}$
3: $Y$ ← BoundedCount($F$, iniThresh);
4: **if** ($Y$ < iniThresh) **then return** $Y$;
5: $t \leftarrow \lceil 17 \log_2(3/\delta) \rceil$;
6: nCells ← 2; $C$ ← emptyList; iter ← 0;
7: **repeat**
8:     iter ← iter + 1;
9:     nSols ← ApproxMC5Core($F, \rho$, thresh);
10:    AddToList($C$, nSols);
11: **until** (iter < $t$);
12: finalEstimate ← FindMedian($C$);
13: **return** finalEstimate
```

Algorithm 2 ApproxMC5Core(F, ρ, thresh)

```
1: Choose $h$ at random from $\mathcal{H}_\rho(n,n)$;
2: Choose $\alpha$ at random from $\{0,1\}^n$;
3: $Y$ ← BoundedCount($F \wedge (h^{(n)})^{-1}(\alpha^{(n)})$, thresh);
4: **if** ($Y$ = thresh) **then return** $2^n$
5: $m$ ← LogSATSearch($F, h, \alpha$, thresh);
6: $\mathrm{Cnt}_{\langle F,m\rangle}$ ← BoundedCount($F \wedge (h^{(m)})^{-1}(\alpha^{(m)})$, thresh);
7: **return** ($2^m \times \mathrm{Cnt}_{\langle F,m\rangle}$);
```

to H_{xor} in line 1. We use $\mathcal{H}_{\rho,\text{qs}}(n,n)$ as a placeholder for a hash family, whose properties would be inferred from the analysis of ApproxMC5 and stated formally in Lemma 6.2.

ApproxMC5Core takes in a formula F, thresh, and returns nSols as an estimate of $|sol(F)|$ within tolerance ε corresponding to thresh. To this end, ApproxMC5Core first chooses a hash function h from a prefix-family $\mathcal{H}_{\rho,\text{qs}}(n,n)$ and a cell α. As noted above, we use *prefix-slices* of h and α. After choosing h and α randomly, ApproxMC5Core checks if $\mathrm{Cnt}_{\langle F,n\rangle} <$ thresh. If not, ApproxMC5Core fails and returns 2^n.(A careful reader would note that we could have chosen any arbitrary number to return) Otherwise, it invokes sub-routine LogSATSearch to find a value of m (and hence, of $h^{(m)}$ and $\alpha^{(m)}$) such that $\mathrm{Cnt}_{\langle F,m\rangle} <$ thresh and $\mathrm{Cnt}_{\langle F,m-1\rangle} \geq$ thresh. The reason behind the particular choice of the value of m is that to obtain higher confidence in the counts returned by ApproxMC5Core, we would ideally like the $\mathrm{E}[\mathrm{Cnt}_{\langle F,m\rangle}]$ to be high so as to obtain better bounds through concentration inequalities. Of course, we can only handle the cases when $\mathrm{Cnt}_{\langle F,m\rangle}$ is polynomial to ensure polynomially many calls to NP oracle (SAT solver in practice). The implementation of LogSATSearch is provided in [Chakraborty et al. 2016a] and we use the procedure as-is. The invocation of BoundedCount in line 6 calculates $\mathrm{Cnt}_{\langle F,m\rangle}$.Finally, ApproxMC5Core returns ($2^m \times \mathrm{Cnt}_{\langle F,m\rangle}$), where 2^m is the number of cells that $sol(F)$ is partitioned into by $h^{(m)}$.

6.2 Analysis of ApproxMC5

We now present the analysis of ApproxMC5. The primary purpose of this section is to highlight the sufficiency of concentrated hashing for the theoretical guarantees of ApproxMC5.

Let Bad denote the event that ApproxMC5Core either returns $(\bot, \bot)$ or returns a pair (2^m, nSols) such that $2^m \times$nSols does not lie in the interval $I_{\text{Good}} = \left[\frac{|sol(F)|}{1+\varepsilon}, |sol(F)|(1+\varepsilon)\right]$. We wish to bound Pr [Bad] from above. Towards this end, for $i \in \{1,\ldots,n\}$, let T_i denote the event $\left(\mathrm{Cnt}_{\langle F,i\rangle} < \text{thresh}\right)$, and let L_i and U_i denote the events $\left(\mathrm{Cnt}_{\langle F,i\rangle} < \frac{|sol(F)|}{(1+\varepsilon)2^i}\right)$ and $\left(\mathrm{Cnt}_{\langle F,i\rangle} > \frac{|sol(F)|}{2^i}\left(1 + \frac{\varepsilon}{1+\varepsilon}\right)\right)$, respectively.

For any event E, let $\overline{E}$ denote its complement. Now, for Bad to happen, ApproxMC5Core must return (at some iteration i) with L_i or U_i. Further, if it returned at i, then T_i holds and T_{i-1} must not hold (else it would have returned at iteration $i-1$ itself). Thus, we obtain

$$\Pr[\text{Bad}] \leq \Pr\left[\bigcup_{i\in\{1,\ldots n\}} \left(\overline{T_{i-1}} \cap T_i \cap (L_i \cup U_i)\right)\right] \quad (12)$$

Note that we only get an upper bound (and not an equality) above because the interval I_{Good} considered has upper bound $|sol(F)|(1+\varepsilon)$, while U_i and thresh are defined using the factor $(1 + \frac{\varepsilon}{1+\varepsilon}) \leq 1 + \varepsilon$.

Our next goal is to simplify this upper bound. Let m^* be the smallest i such that $\frac{|sol(F)|}{2^i}(1+\varepsilon) \leq \text{thresh} - 1$. This value must exist since $|sol(F)| \geq$ iniThresh. Note that when $|sol(F)| <$ iniThresh, the algorithm returns the exact count and hence is guaranteed to be correct. Now, by substituting the chosen value of thresh and simplifying, we obtain

$$m^* = \left\lfloor \log_2 |sol(F)| - \log_2 \left(4.92 \cdot \rho \cdot \left(1 + \frac{1}{\varepsilon}\right)^2\right)\right\rfloor \quad (13)$$

From the definition of m^*, we have $2^{m^*+1} \geq \frac{2*|sol(F)|}{\text{thresh}-1}$. Since $|sol(F)| \geq$ iniThresh, we have $2^{m^*+1} \geq \frac{2*\text{thresh}*2^{\text{qs}+3}}{\text{thresh}-1}$, i.e., $m^*+1 \geq \text{qs}+4$, or $m^*-3 \geq \text{qs}$.

Similar to ApproxMC4, we show that for ApproxMC5, one can upper bound Bad by considering only five events, namely, $T_{m^*-3} L_{m^*-2}$, L_{m^*-1}, L_{m^*} and U_{m^*}. It is worth noting that the proof only requires usage of prefix-hash family in the algorithm with no further restrictions on nature of the prefix-hash family. In fact, the main property that we need from the prefix hash family, which follows from Proposition 2.4, is that

$$\forall j \in \{1,\ldots,n\}, T_j \implies T_{j+1} \quad (14)$$

Lemma 6.1. $\Pr[\text{Bad}] \leq \Pr[T_{m^*-3}] + \Pr[L_{m^*-2}] + \Pr[L_{m^*-1}] + \Pr[L_{m^*} \cup U_{m^*}]$

The following lemma utilizes the key property of concentrated hash families stated in Proposition 2.7 to bound the probabilities of the concerned events.

Lemma 6.2. *If $\mathcal{H}$ is prefix-$(\rho, \text{qs}, \text{pivot})$-concentrated family for* $\text{pivot} = 78.72 \cdot \rho(1 + \frac{1}{\varepsilon})^2$*, then the following bounds hold:*

1. $\Pr[L_{m^*} \cup U_{m^*}] \le \frac{1}{4.92}$
2. $\Pr[L_{m^*-1}] \le \frac{1}{10.84}$
3. $\Pr[L_{m^*-2}] \le \frac{1}{20.68}$
4. $\Pr[T_{m^*-3}] \le \frac{1}{62.5}$

Proof. Note that $\Pr[T_i] = \Pr[\mathsf{Cnt}_{\langle F,i \rangle} \le \text{thresh}]$ and $\Pr[L_i] = \Pr\left[\mathsf{Cnt}_{\langle F,i \rangle} \le (1+\varepsilon)^{-1}\mu_i\right]$. Furthermore, $\Pr[L_i \cup U_i] = \Pr\left[\mathsf{Cnt}_{\langle F,i \rangle} - \mu_i| \ge \frac{\varepsilon}{1+\varepsilon}\mu_i\right]$ To obtain bounds, we substitute values of m^*, thresh, μ_i, and we seek to apply Proposition 2.7 with appropriate values of β. We observe that to obtain (1), it is sufficient to employ $(\rho, m^*, \frac{\text{pivot}}{8})$ concentrated family; Similarly, to obtain (2), (3) , (4), it is sufficient to employ $(\rho, m^* - 1, \text{pivot}/4)$, $(\rho, m^* - 2, \frac{\text{pivot}}{2})$, $(\rho, m^* - 3, \text{pivot})$ concentrated families respectively. Proposition 2.6 allows us to conclude that $(\rho, m^* - 3, \text{pivot})$ concentrated family suffices to obtain the above bounds. Since $m^* - 3 \ge \text{qs}$, we conclude that $(\rho, \text{qs}, \text{pivot})$-concentrated family suffices to obtain the above bounds. □

Combining Lemma 6.1 with the observation that ApproxMC5Core is invoked $\mathcal{O}(\log \frac{1}{\delta})$ times and we return median as the estimate, we obtain the following correctness and time complexity for ApproxMC5 by using the standard Chernoff analysis for the amplification of probability bounds.

Theorem 3.5. *For input formula F, tolerance parameter ε, confidence parameter δ, and concentrated hashing parameters ρ and* qs*, suppose* ApproxMC5$(F, \varepsilon, \delta, \rho)$ *uses a prefix* $(\rho, \text{qs}, \text{pivot})$*-concentrated hash family with the value of* $\text{pivot} = 78.72 \cdot \rho(1+\frac{1}{\varepsilon})^2$ *and returns an estimate c. Then,* $\Pr\left[\frac{|sol(F)|}{1+\varepsilon} \le c \le (1+\varepsilon)|sol(F)|\right] \ge 1 - \delta$*. Furthermore,* ApproxMC5 *makes* $\mathcal{O}(2^{\text{qs}+3} + \frac{\log(n)\log(1/\delta)}{\varepsilon^2})$ *calls to a SAT-oracle.*

The correctness and time complexity of ApproxMC5 have exactly the same expression as that of ApproxMC4. Theorem 3.5 highlights that prefix-$(\rho, \text{qs}, \text{pivot})$–concentrated hash family are sufficient to provide (ε, δ) estimates. In fact, in our experimental results that we discuss next, we will use a sparse hash function belonging to this family.

6.3 Further Optimizations

As mentioned earlier, BoundedCount is a subroutine that takes in a formula F and threshold thresh, and uses a NP oracle to enumerate solutions of F one by one until we have found the desired threshold number of solutions or there are no more solutions. The practical implementation of BoundedCount replaces NP oracle with SAT solver and and as such for a fixed formula F, the runtime of BoundedCount depends on thresh. The usage of $(\rho, \text{qs}, \text{pivot})$-concentrated family leads to invocation of BoundedCount with threshold set to iniThresh in line 3 of ApproxMC4 algorithm. Therefore, for practical efficiency, it is desirable to construct concentrated families with as small values of qs as possible. The bound on qs provided by the proof of Theorem 3.3 is prohibitively large (qs > 70) even for $n = 10$. To this end, we turn to analytical techniques aided by scientific programming in Python.

For given n, k, and ρ, we seek to compute as small values of p_i as possible while satisfying $p_i \ge p_{i+1}$. As a first step, we observe that the upper bound for $c_S(w)$ employed above is a loose upper bound and accordingly, the bounds on the constants p_i (as well as k and the large enough value of m) obtained from our analysis above are very loose. To this end, we compute $c_S(w)$ based on the Eq 4.2 and Eq 4.5 obtained in [Rashtchian 2018], as indicated in the proof of Lemma 4.5. We then compute the values of p_i for $\text{qs} = 1$, $k = 512$, and $\rho = 1.1$. The particular values for ρ and k were chosen due to their usage in experimental evaluation of ApproxMC5. We call the resulting family $\mathcal{H}^{lsa}_{Rennes}$ and employ $\mathcal{H}^{lsa}_{Rennes}$ in our empirical evaluation.

Figure 1 plots the values of computed p_i vis-a-vis i We also plot another curve $f(i) = \frac{1.6\log_2(i+1)}{i}$. It is interesting to observe that the two curves fit nicely to each other. To illustrate the gap between observed and theoretical bound, we plot the bound on p obtained from Theorem 3.3 as $g(i) = \frac{32\log_2(i+1)}{\delta \cdot i}$ noting that $H^{-1}(\delta) < \frac{\delta}{2}$.

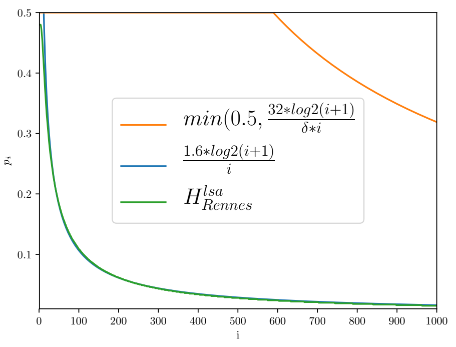

Figure 1. Trend of p_i vis-a-vis i

The large difference between the two plots clearly illustrates the potential for improvement of constants in Theorem 3.1 and we leave this as a natural direction of future work. Furthermore, we conjecture existence of sparse prefix hash functions with $p_m = \mathcal{O}(\frac{\log m}{m})$ belonging to $(\rho, 1, \kappa)$-concentrated family.

7 Experimental Evaluation

In this section, we evaluate the performance of our approximate model counting algorithm ApproxMC5 using the prefix $(1.1, 1, \text{pivot})$-concentrated hash family $\mathcal{H}^{lsa}_{Rennes}$ [3]. For all our experiments, we used $\varepsilon = 0.8$ and $\delta = 0.1$, which is

[3]Our theoretical analysis of ApproxMC5 allows all values of $\rho \ge 1$ and $\text{qs} > 1$; we leave further optimization of the choice of ρ as future work.

Benchmark	**Vars**	**Clauses**	$\lvert\mathcal{P}\rvert$	$\log_2$(Count)	ApproxMC4 **time**	ApproxMC5 **time**	**Speedup**
10B-1	15390	68337	174	56.17	4274.56	–	–
or-100-20-7-UC-40	200	539	200	56.55	3526.45	–	–
03B-4	27966	123568	114	28.55	983.72	1548.96	0.64
blasted_TR_b12_2_linear	2426	8373	107	63.93	32.07	56.75	0.57
blasted_squaring23	710	2268	61	23.11	0.66	1.21	0.55
blasted_case144	765	2340	138	82.07	102.65	202.06	0.51
modexp8-4-6	83953	316814	88	32.13	788.23	920.34	0.86
or-70-5-5-UC-20	140	360	140	43.91	675.1	788.74	0.86
min-28s	3933	13118	464	459.23	48.63	35.83	1.36
90-14-8-q	924	811	924	728.29	242.07	178.93	1.35
s9234a_7_4	6313	14555	247	246.0	4.77	2.45	1.95
min-8	1545	4230	288	284.78	8.86	4.59	1.93
s13207a_7_4	9386	20635	700	699.0	34.94	17.05	2.05
min-16	3065	8526	544	539.88	33.67	16.61	2.03
90-15-4-q	1065	911	1065	839.25	273.1	135.75	2.01
s35932_15_7	17918	44709	1763	1761.0	–	72.32	–
s38417_3_2	25528	57586	1664	1663.02	–	71.04	–
75-10-8-q	460	465	460	360.13	–	4850.28	–
90-15-8-q	1065	951	1065	840.0	–	3717.05	–

Table 2. Runtime performance comparison of ApproxMC5 vis-a-vis ApproxMC4. (Timeout: 5000 seconds)

in line with the chosen values for these parameters in previous studies on counting. The setting of $\varepsilon = 0.8$ yields pivot to be 512. Recall that prior empirical studies had to sacrifice theoretical guarantees due to their reliance on far fewer invocations of SAT solver than those dictated by the theoretical analysis [Achlioptas et al. 2018; Achlioptas and Theodoropoulos 2017; Ermon et al. 2014; Zhao et al. 2016]. In contrast, we use a faithful implementation of ApproxMC5 that retains theoretical guarantees of (ε, δ) approximation. ApproxMC5 is publicly available as an open source software at: https://github.com/meelgroup/approxmc.

To evaluate the runtime performance and quality of approximations computed by ApproxMC5, we conducted a comprehensive performance evaluation of counting algorithms involving 1896 benchmarks. Most practical applications of model counting reduce to projected counting and therefore, keeping in line with the prior work, we experiment with benchmarks arising from wide range of application areas including probabilistic reasoning, plan recognition, DQMR networks, ISCAS89 combinatorial circuits, quantified information flow, program synthesis, functional synthesis, logistics, as have been previously employed in studies on model counting [Chakraborty et al. 2016a; Lagniez and Marquis 2017]. We perform runtime comparisons with ApproxMC4 as ApproxMC4 was shown to be state of the art approximate counter with significant performance gain over other approximate counters [Soos et al. 2020; Soos and Meel 2019].

The objective of our experimental evaluation was to answer the following questions:

1. How does runtime performance of ApproxMC5 compare with that of ApproxMC4?
2. How far are the counts computed by ApproxMC5 from the exact counts?

The experiments were conducted on a high performance computer cluster, with each node consisting of an E5-2690 v3 CPU with 24 cores and 96GB of RAM such that each core's access was restricted to 4GB. The computational effort for the evaluation consisted of over 20,000 hours. We used timeout of 5,000 seconds for each experiment, which consisted of running a tool on a particular benchmark. To further optimize the running time for both ApproxMC4 and ApproxMC5, we used improved estimates of the iteration count t following an analysis similar to that in [Chakraborty et al. 2016a].

7.1 Results

Runtime performance

We present the runtime comparison of ApproxMC5 vis-a-vis ApproxMC4 in Table 2 on a subset of our benchmarks [4]. Column 1 specifies the name of the benchmark, while columns 2 and 3 list the number of variables and clauses, respectively. Column 4 Column 4 lists the $\log_2$ of the estimate returned by ApproxMC5. Columns 5 and 6 list the runtime (in seconds) of ApproxMC5 and ApproxMC4 respectively. Column 7 indicates speedup of ApproxMC5 over ApproxMC4. We observe:

1. ApproxMC5 significantly outperforms ApproxMC4 for a large set of benchmarks. We observe that ApproxMC5 is able to compute estimates for formulas for which ApproxMC4 timed out. Furthermore, ApproxMC5 is

[4]The entire set of benchmarks and the corresponding set of logs generated by ApproxMC4 and ApproxMC5 are available at https://doi.org/10.5281/zenodo.3766168

also significantly faster for most of the benchmarks where ApproxMC4 does not timeout.

2. Recall that the density of XORs decreases with increase in $\log_2 |sol(F)|$ and we observe that the performance of ApproxMC5 too improves further as the number of solutions of F increases. It is worth noting that for a subset of benchmarks, ApproxMC5 is slower than ApproxMC4.

Upon further investigation, we observe a strong correlation between the speedup and the $\log_2$ of the number of solutions. It is worth recalling that the number of XORs required to ensure that a randomly chosen cell is small is close to $\log_2$ of the number of solutions. Since for a fixed number of variables, the sparsity increases with the number of XORs, there is a tradedoff between the gains due to sparse XORs over the increased overhead of requirement of enumerating higher number of solutions due to increased thresh. It is worth viewing the runtime improvement in the context of prior work where significant slowdown was observed.

Approximation Quality

To measure the quality of approximation, we compared the approximate counts returned by ApproxMC5 with the counts computed by an exact model counter, viz. DSharp. Figure 2 shows the model counts computed by ApproxMC5, and the bounds obtained by scaling the exact counts with the tolerance factor ($\varepsilon = 0.8$) for a small subset of benchmarks. The y-axis represents model counts on log-scale while the x-axis represents benchmarks ordered in ascending order of model counts. We observe that for *all* the benchmarks, ApproxMC5 computed counts within the tolerance. Furthermore, for each instance, the observed tolerance (ε_{obs}) was calculated as $\max(\frac{|sol(F)|}{\text{AprxCount}}-1, \frac{\text{AprxCount}}{|sol(F)|}-1)$, where AprxCount is the estimate computed by ApproxMC5. We observe that the arithmetic mean of ε_{obs} across all benchmarks is 0.05 – far better than the theoretical guarantee of 0.8.

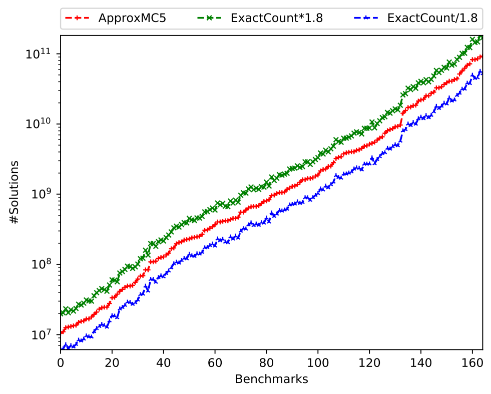

Figure 2. Plot showing counts obtained by ApproxMC5 vis-a-vis exact counts from DSharp

8 Conclusion

Our investigations were motivated by the runtime performance of SAT solvers on sparse hash functions. This led us to introduce a relaxation of universal hash functions, christened as (ρ, qs, k)-concentrated hash functions. The usage of (ρ, qs, k)–concentrated hash functions ensure that dispersion index for the random variable, $\mathsf{Cnt}_{\langle F,m\rangle}$ is bounded by the constant ρ. We use our bounds to construct sparse hash functions, named $\mathcal{H}^{k}_{Rennes}$ where each entry of $A[i]$ is chosen with probability $p_i = O(\frac{\log_2 i}{i})$. Finally, we replace strong 2-universal hash functions with $\mathcal{H}^{lsa}_{Rennes}$ (an analytically computed variant of $\mathcal{H}^{k}_{Rennes}$) and implement the resulting algorithm demonstrating significant speedup compared to the state-of-the-art in approximate model counters.

We believe that the sparse hash functions constructed here could have potential applications in other domains such as discrete integration, streaming, and the like. This work suggests two interesting directions of future research:

- Design of explicit constructions of sparse hash functions belonging to (ρ, qs, k)-concentrated family for all values of qs, ideally for $\text{qs} = 1$.
- Design of hashing-based techniques where the usage of sparse hash functions performs as good as or better than those based on dense XORs for almost all the benchmarks.

Acknowledgments. The authors would like to thank Yash Pote, Mate Soos, and Bhavishya for their generous help in experimental evaluation, Supratik Chakraborty and Aditya Shrotri for providing crucial feedback and in particular, discovery of a bug; the repair of which was possible due to generous help from Cyrus Rashtchian and Paul Beame. Part of the work was performed during both authors' stays in Rennes and IIT Delhi, and Meel's visits to IIT Bombay; the authors are grateful to INRIA Rennes, IIT Delhi, and IIT Bombay for the hospitality. Meel owes gratitude to Ashish Gupta for organizing a memorable trip and the ensuing discussions that provided crucial insights into linking variance to the structure of solution space.

This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Program [NRF-NRFFAI1-2019-0004], NUS ODPRT Grant [R-252-000-685-13], and Sung Kah Kay Assistant Professorship Endowment, DST-SERB MATRICES grant and DST-INRIA CEFIPRA Indo-French project EQuaVe.

References

Dimitris Achlioptas, Zayd Hammoudeh, and Panos Theodoropoulos. 2018. Fast and Flexible Probabilistic Model Counting. In *Proc. of SAT*. 148–164.

Dimitris Achlioptas and Panos Theodoropoulos. 2017. Probabilistic model counting with short XORs. In *Proc. of SAT*. Springer, 3–19.

Durgesh Agarwal, Bhavishya, and Kuldeep S. Meel. 2020. On the Size of XORs in Approximate Model Counting. In *Proc. of SAT*.

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach.* Cambridge Univ. Press.

Megasthenis Asteris and Alexandros G Dimakis. 2016. *LDPC codes for discrete integration.* Technical Report. Technical report, UT Austin.

Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S Meel, and Prateek Saxena. 2019. Quantitative verification of neural networks and its security applications. In *Proc. of CCS*. 1249–1264.

Paul Beame and Cyrus Rashtchian. 2017. Massively-parallel similarity join, edge-isoperimetry, and distance correlations on the hypercube. In *Proc. of SODA*. Society for Industrial and Applied Mathematics, 289–306.

Lawrence J. Carter and Mark N Wegman. 1977. Universal classes of hash functions. In *Proc. of STOC*. ACM, 106–112.

Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. 2014. Distribution-Aware Sampling and Weighted Model Counting for SAT. In *Proc. of AAAI*. 1722–1730.

Supratik Chakraborty, Kuldeep S. Meel, Rakesh Mistry, and Moshe Y. Vardi. 2016b. Approximate Probabilistic Inference via Word-Level Counting. In *Proc. of AAAI*.

Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. 2013. A Scalable Approximate Model Counter. In *Proc. of CP*. 200–216.

Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. 2016a. Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls. In *Proc. of IJCAI*.

Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2013. Optimization With Parity Constraints: From Binary Codes to Discrete Integration. In *Proc. of UAI*.

Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2014. Low-density Parity Constraints for Hashing-Based Discrete Integration. In *Proc. of ICML*. 271–279.

Matthew Fredrikson and Somesh Jha. 2014. Satisfiability Modulo Counting: A New Approach for Analyzing Privacy Properties. In *Proc. of CSL-LICS*. 42:1–42:10.

Carla P. Gomes, Joerg Hoffmann, Ashish Sabharwal, and Bart Selman. 2007. Short XORs for model counting: from theory to practice. In *Proc. of SAT*. 100–106. http://dl.acm.org/citation.cfm?id=1768142.1768155

Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2006. Model counting: A new strategy for obtaining good bounds. In *Proc. of AAAI*, Vol. 21. 54–61.

Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. 2016. On computing minimal independent support and its applications to sampling and counting. *Constraints* (2016), 1–18. https://doi.org/10.1007/s10601-015-9204-z

Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. 1986. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science* 43, 2-3 (1986), 169–188. http://portal.acm.org/citation.cfm?id=11534.11537

Jean-Marie Lagniez and Pierre Marquis. 2017. An improved decision-DNNF compiler. In *Proc. of IJCAI*, Vol. 2017.

David JC MacKay. 1999. Good error-correcting codes based on very sparse matrices. *IEEE transactions on Information Theory* 45, 2 (1999), 399–431.

Kuldeep S. Meel ⓡ S. Akshay. 2020. Sparse Hashing for Scalable Approximate Model Counting: Theory and Practice. *arXiv preprint arXiv:2004.14692*. http://arxiv.org/abs/2004.14692

Kuldeep S. Meel, Moshe Vardi, Supratik Chakraborty, Daniel J Fremont, Sanjit A Seshia, Dror Fried, Alexander Ivrii, and Sharad Malik. 2016. Constrained Sampling and Counting: Universal Hashing Meets SAT Solving. In *Proc. of Beyond NP Workshop*.

Cyrus Rashtchian. 2018. *New Algorithmic Tools for Distributed Similarity Search and Edge Estimation.* Ph.D. Dissertation.

Cyrus Rashtchian and William Raynaud. 2019. Edge Isoperimetric Inequalities for Powers of the Hypercube. *arXiv preprint arXiv:1909.10435* (2019).

Dan Roth. 1996. On the hardness of approximate reasoning. *Artificial Intelligence* 82, 1 (1996), 273–302. https://doi.org/10.1016/0004-3702(94)00092-1

Tian Sang, Paul Beame, and Henry Kautz. 2005. Performing Bayesian inference by weighted model counting. In *Prof. of AAAI*. 475–481.

Mate Soos, Stephan Gocht, and Kuldeep S. Meel. 2020. Accelerating Approximate Techniques for Counting and Sampling Models Through Refined CNF-XOR Solving. In *Proc. of CAV*.

Mate Soos and Kuldeep S Meel. 2019. 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)*.

Larry J. Stockmeyer. 1983. The complexity of approximate counting. In *Proc. of STOC*. 118–126.

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

Luca Trevisan. 2002. 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.

Salil P Vadhan et al. 2012. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science* 7, 1–3 (2012), 1–336.

Leslie G. Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.

Shengjia Zhao, Sorathan Chaturapruek, Ashish Sabharwal, and Stefano Ermon. 2016. Closing the Gap Between Short and Long XORs for Model Counting. In *Proc. of AAAI*.