On Hashing-Based Approaches to Approximate DNF-Counting*†‡

Kuldeep S. Meel[§1], Aditya A. Shrotri[2], and Moshe Y. Vardi[3]

1 **National University of Singapore, Singapore, Singapore**
meel@comp.nus.edu.sg
2 **Rice University, Houston, USA**
as128@rice.edu
3 **Rice University, Houston, USA**
vardi@rice.edu

Abstract

Propositional model counting is a fundamental problem in artificial intelligence with a wide variety of applications, such as probabilistic inference, decision making under uncertainty, and probabilistic databases. Consequently, the problem is of theoretical as well as practical interest. When the constraints are expressed as DNF formulas, Monte Carlo-based techniques have been shown to provide a fully polynomial randomized approximation scheme (FPRAS). For CNF constraints, hashing-based approximation techniques have been demonstrated to be highly successful. Furthermore, it was shown that hashing-based techniques also yield an FPRAS for DNF counting without usage of Monte Carlo sampling. Our analysis, however, shows that the proposed hashing-based approach to DNF counting provides poor time complexity compared to the Monte Carlo-based DNF counting techniques. Given the success of hashing-based techniques for CNF constraints, it is natural to ask: Can hashing-based techniques provide an efficient FPRAS for DNF counting? In this paper, we provide a positive answer to this question. To this end, we introduce two novel algorithmic techniques: *Symbolic Hashing* and *Stochastic Cell Counting*, along with a new hash family of *Row-Echelon hash functions*. These innovations allow us to design a hashing-based FPRAS for DNF counting of similar complexity (up to polylog factors) as that of prior works. Furthermore, we expect these techniques to have potential applications beyond DNF counting.

1998 ACM Subject Classification G.1.2 Special Function Approximation, F.4.1 Logic and Constraint Programming

Keywords and phrases Model Counting, Approximation, DNF, Hash Functions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.41

1 Introduction

Propositional model counting is a fundamental problem in artificial intelligence with a wide range of applications including probabilistic inference, databases, decision making under

* The author list has been sorted alphabetically by last name; this should not be used to determine the extent of authors' contributions.
† The full version of this paper is available at https://arxiv.org/abs/1710.05247.
‡ Work supported in part by NSF grants CCF-1319459 and IIS-1527668 and by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering".
§ Kuldeep S. Meel is supported by the IBM PhD Fellowship and the Lodieska Stockbridge Vaughn Fellowship.

uncertainty, and the like [1, 6, 7, 23]. Given a Boolean formula ϕ, the problem of propositional model counting , also referred to as #SAT, is to compute the number of solutions of ϕ [28]. Depending on whether ϕ is expressed as a CNF or DNF formula, the corresponding model counting problems are denoted as #CNF or #DNF, respectively. Both #CNF and #DNF have a wide variety of applications. For example, probabilistic-inference queries reduce to solving #CNF instances [1, 20, 21, 23], while evaluation of queries for probabilistic database reduce to #DNF instances [6]. Consequently, both #CNF and #DNF have been of theoretical as well as practical interest over the years [16, 18, 22, 24]. In his seminal paper, Valiant [28] showed that both #CNF and #DNF are #P-complete, a class of problems that are believed to be intractable in general.

Given the intractability of #CNF and #DNF, much of the interest lies in the approximate variants of #CNF and #DNF, wherein for given tolerance and confidence parameters ε and δ, the goal is to compute an estimate C such that C is within a $(1+\varepsilon)$ multiplicative factor of the true count with confidence at least $1-\delta$. While both #CNF and #DNF are #P-complete in their exact forms, the approximate variants differ in complexity: approximating #DNF can be accomplished in fully polynomial randomized time [5, 17, 18], but approximate #CNF is NP-hard [24]. Consequently, different techniques have emerged to design scalable approximation techniques for #DNF and #CNF.

In the context of #DNF, the works of Karp, Luby, and Madras [17, 18] led to the development of highly efficient Monte-Carlo based techniques, whose time complexity is linear in the size of the formula. On the other hand, hashing-based techniques have emerged as a scalable approach to the approximate model counting of CNF formulas [3, 4, 10, 13, 24], and are effective even for problems with existing FPRAS such as *network reliability* [8]. These hashing-based techniques employ 2-universal hash functions to partition the space of satisfying solutions of a CNF formula into cells such that a randomly chosen cell contains only a *s*mall number of solutions. Furthermore, it is shown that the number of solutions across the cells is *roughly equal* and, therefore, an estimate of the total count can be obtained by counting the number of solutions in a cell and scaling the obtained count by the number of cells. Since the problem of counting the number of solutions in a cell when the number of solutions is *small* can be accomplished efficiently by invoking a SAT solver, the hashing-based techniques can take advantage of the recent progress in the development of efficient SAT solvers. Consequently, algorithms such as ApproxMC [3, 4] have been shown to scale to instances with hundreds of thousands of variables.

While Monte Carlo techniques introduced in the works of Karp et al. have shown to not be applicable in the context of approximate #CNF [18], it was not known whether hashing-based techniques could be employed to obtain efficient algorithms for #DNF. Recently, significant progress in this direction was achieved by Chakraborty, Meel and Vardi [4], who showed that hashing-based framework of ApproxMC could be employed to obtain FPRAS for #DNF counting[1]. There is, however, no precise complexity analysis in [4]. In this paper, we provide a complexity analysis of the proposed scheme of Chakraborty et al., which is worse than quartic in the size of formula. In comparison, state-of- the-art approaches achieve complexity linear in the number of variables and cubes for #DNF counting. This begs the question: *How powerful is the hashing-based framework in the context of DNF counting? In particular, can it lead to algorithms competitive in runtime complexity with state-of-the-art?*

[1] It is worth noting that several hashing-based algorithms based on [10, 27] do not lead to FPRAS schemes for #DNF despite close similarity to Chakraborty et al.'s approach

In this paper, we provide a positive answer to this question. To achieve such a significant reduction in complexity, we offer three novel algorithmic techniques: (i) A new class of 2-universal hash functions that enable fast enumeration of solutions using Gray Codes, (ii) Symbolic Hashing, and (iii) Stochastic Cell Counting. These techniques allow us to achieve the complexity of $\tilde{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$, which is within polylog factors of the complexity achieved by Karp et al. [18]. Here, m and n are the number of cubes and variables respectively while ε and δ are the tolerance and confidence of approximation. Furthermore, we believe that these techniques are not restricted to #DNF. Given recent breakthroughs achieved in the development of hashing-based CNF-counting techniques, we believe our techniques have the potential for a wide variety of applications.

The rest of the paper is organized as follows: we introduce notation in section 2 and discuss related work in section 3. We describe our main contributions in section 4, analyze the resulting algorithm in section 5 and discuss future work and conclude in section 6.

2 Preliminaries

DNF Formulas and Counting

We use Greek letters ϕ, θ and ψ to denote boolean formulas. A formula ϕ over boolean variables $x_1, x_2, \ldots, x_{\mathsf{n}}$ is in Disjunctive Normal Form (DNF) if it is a disjunction over conjunctions of variables or their negations. We use X to denote the set of variables appearing in the formula. Each occurrence of a variable or its negation is called a *literal*. Disjuncts in the formula are called *cubes* and we denote the i^{th} cube by ϕ^{Ci}. Thus $\phi = \phi^{C1} \vee \phi^{C2} \vee \ldots \vee \phi^{C\mathsf{m}}$ where each ϕ^{Ci} is a conjunction of *literals*. We will use n and m to denote the number of variables and number of cubes in the input DNF formula, respectively. The number of literals in a cube ϕ^{Ci} is called its *width* and is denoted by $width[\phi^{Ci}]$.

An assignment to all the variables can be represented by a vector $\boldsymbol{x} \in \{0,1\}^{\mathsf{n}}$ with 1 corresponding to $true$ and 0 to $false$. $U = \{0,1\}^{\mathsf{n}}$ is the set of all possible assignments, which we refer to as the *universe* or state space interchangeably. An assignment $\boldsymbol{x}$ is called a satisfying assignment for a formula ϕ if ϕ evaluates to $true$ under $\boldsymbol{x}$. In other words $\boldsymbol{x}$ satisfies ϕ and is denoted as $\boldsymbol{x} \models \phi$. Note that an assignment $\boldsymbol{x}$ will satisfy a DNF formula ϕ if $\boldsymbol{x} \models \phi^{Ci}$ for some i. The DNF-Counting Problem is to count the number of satisfying assignments of a DNF formula.

Next, we formalize the concept of a counting problem. Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation which is decidable in polynomial time and there is a polynomial p such that for every $(s,t) \in R$ we have $|t| \leq p(|s|)$. The decision problem corresponding to R asks if for a given s there exists a t such that $(s,t) \in R$. Such a problem is in NP. Here, s is a called the *problem instance* and t is called the *witness*. We denote the set of all witnesses for a given s by R_s. The *counting problem* corresponding to R is to calculate the size of the witness set $|R_s|$ for a given s. Such a problem is in #P[28]. The DNF-Counting problem is an example of this formalism: A formula ϕ is a problem instance and a satisfying assignment $\boldsymbol{x}$ is a witness of ϕ. The set of satisfying assignments or the solution space is denoted R_ϕ and the goal is to compute $|R_\phi|$. It is known that the problem is #P-Complete, which is believed to be intractable [26]. Therefore, we look at what it means to efficiently and accurately approximate this problem.

A *fully polynomial randomized approximation scheme* (FPRAS) is a randomized algorithm that takes as input a problem instance s, a tolerance $\varepsilon \in (0,1)$ and confidence parameter $\delta \in (0,1)$ and outputs a random variable c such that $\Pr[\frac{1}{1+\varepsilon}|R_s| \leq c \leq (1+\varepsilon)|R_s|] \geq 1-\delta$ and the running time of the algorithm is polynomial in $|s|$, $1/\varepsilon$, $\log(1/\delta)$ [17]. Notably,

while exact DNF-counting is inter-reducible with exact CNF-counting, the approximate versions of the two problems are not because multiplicative approximation is not closed under complementation.

Matrix Notation

We use x,y, z,... to denote scalar variables. We use subscripts $x_1, x_2, \ldots$ as required. In this paper we are dealing with operations over the boolean ring, where the variables are boolean, 'addition' is the XOR operation ($\oplus$) and 'multiplication' is the AND operation ($\wedge$). We use the letters i,j,k,l as indices or to denote positions. We denote sets by non-boldface capital letters. We use capital boldface letters $\boldsymbol{A}$,$\boldsymbol{B}$,... to denote matrices, small boldface letters $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$,... to denote vectors. $\boldsymbol{A}^{[\mathsf{p}\times\mathsf{q}]}$ denotes a matrix of p rows and q columns, while $\boldsymbol{u}^{[\mathsf{q}]}$ denotes a vector of length q. $\boldsymbol{0}^{[\mathsf{q}]}$ and $\boldsymbol{1}^{[\mathsf{q}]}$ are the all 0s and all 1s vectors of length n, respectively. We omit the dimensions when clear from context. $\boldsymbol{x}[i]$ denotes the i^{th} element of $\boldsymbol{x}$, while $\boldsymbol{A}[i,j]$ denotes the element in the ith row and j^{th} column of $\boldsymbol{A}$. $\boldsymbol{A}[r_1 : r_2, c_1 : c_2]$ denotes the sub-matrix of $\boldsymbol{A}$ between rows r_1 and r_2 excluding r_2 and columns c_1 and c_2 excluding c_2. Similarly $\boldsymbol{v}[i : j]$ denotes the sub-vector of $\boldsymbol{v}$ between index i and index j excluding j. The i^{th} row of $\boldsymbol{A}$ is denoted $\boldsymbol{A}[i,:]$ and j^{th} column as $\boldsymbol{A}[:,j]$. The $\mathsf{p}\times(\mathsf{q}_1+\mathsf{q}_2)$ matrix formed by concatenating rows of matrices $\boldsymbol{A}^{[\mathsf{p}\times\mathsf{q}_1]}$ and $\boldsymbol{B}^{[\mathsf{p}\times\mathsf{q}_2]}$ is written in block notation as $[\boldsymbol{A} \mid \boldsymbol{B}]$, while $[\frac{\boldsymbol{A}}{\boldsymbol{B}}]$ represents concatenation of columns. Similarly the $(\mathsf{q}_1+\mathsf{q}_2)$-length concatenation of vectors $\boldsymbol{v}^{[\mathsf{q}_1]}$ and $\boldsymbol{w}^{[\mathsf{q}_2]}$ is $[\boldsymbol{v} \mid \boldsymbol{w}]$. The dot product between matrix $\boldsymbol{A}$ and vector $\boldsymbol{x}$ is written as $\boldsymbol{A}.\boldsymbol{x}$. The vector formed by element-wise XOR of vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ is denoted $\boldsymbol{v}\oplus\boldsymbol{w}$.

Hash Functions

A hash function $h : \{0,1\}^{\mathsf{q}} \to \{0,1\}^{\mathsf{p}}$ partitions the elements of of the domain $\{0,1\}^{\mathsf{q}}$ into 2^{p} cells. $h(\boldsymbol{x}) = \boldsymbol{y}$ implies that h maps the assignment $\boldsymbol{x}$ to the cell $\boldsymbol{y}$. $h^{-1}(\boldsymbol{y}) = \{\boldsymbol{x} | h(\boldsymbol{x}) = \boldsymbol{y}\}$ is the set of assignments that map to the cell $\boldsymbol{y}$. In the context of counting, 2-universal families of hash functions, denoted by $H(\mathsf{q},\mathsf{p},2)$, are of particular importance. When h is sampled uniformly at random from $H(\mathsf{q},\mathsf{p},2)$, 2-universality entails

1. $\Pr[h(\boldsymbol{x}_1) = h(\boldsymbol{x}_2)] \leq 2^{-\mathsf{p}}$ for all $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$
2. $\Pr[h(\boldsymbol{x}) = \boldsymbol{y}] = 2^{-\mathsf{p}}$ for every $\boldsymbol{x} \in \{0,1\}^{\mathsf{q}}$ and $\boldsymbol{y} \in \{0,1\}^{\mathsf{p}}$.

Of particular interest is the random XOR family of hash functions, which is defined as $H_{XOR}(\mathsf{q},\mathsf{p}) = \{\boldsymbol{A}.\boldsymbol{x}\oplus\boldsymbol{b} \mid \boldsymbol{A}[i,j] \in \{0,1\}$ *and* $\boldsymbol{b}[i] \in \{0,1\}\ ;\ 0 \leq i < \mathsf{p},\ 0 \leq j < \mathsf{q}\}$. Selecting $\boldsymbol{A}[i,j]s$ and $\boldsymbol{b}[i]$s randomly from $\{0,1\}$ is equivalent to drawing uniformly at random from this family. A pair $\boldsymbol{A}$ and $\boldsymbol{b}$ now defines a hash function $h_{\boldsymbol{A},\boldsymbol{b}}$ as follows: $h_{\boldsymbol{A},\boldsymbol{b}}(x) = \boldsymbol{A}.\boldsymbol{x}\oplus\boldsymbol{b}$. This family was shown to be 2-universal in [2]. For a hash function $h \in H_{XOR}(\mathsf{q},\mathsf{p})$, we have that $h(\boldsymbol{x}) = \boldsymbol{y}$ is a system of linear equations modulo 2: $\boldsymbol{A}.\boldsymbol{x}\oplus\boldsymbol{b} = \boldsymbol{y}$. From another perspective, it can be viewed as a boolean formula $\psi = \bigwedge_{i=1}^{\mathsf{p}}(\bigoplus_{j=1}^{\mathsf{q}}(\boldsymbol{A}[i,j]\wedge\boldsymbol{x}[j]))\oplus\boldsymbol{b}[i] = \boldsymbol{y}[i]$. The solutions to this formula are exactly the elements of the set $h^{-1}(\boldsymbol{y})$.

Gaussian Elimination

Solving a system of linear equations over q variables and p constraints can be done by row reduction technique known variously as *Gaussian Elimination* or *Gauss-Jordan Elimination*. A matrix is in *Row-Echelon form* if rows with at least one nonzero element are above any rows of all zeros. The matrix is in *Reduced Row-Echelon form* if, in addition, every leading non-zero element in a row is 1 and is the only nonzero entry in its column. We refer to the

technique for obtaining the *Reduced Row-Echelon* form of a matrix as Gaussian Elimination. We refer the reader to any standard text on linear algebra (cf., [25]) for details. For a matrix in Reduced Row-Echelon form, the row-rank is simply the number of non-zero rows.

For a system of linear equations $\boldsymbol{A}.\boldsymbol{x} \oplus \boldsymbol{b} = \boldsymbol{y}$, if the row-rank of the augmented matrix is same as row-rank of $\boldsymbol{A}$, then the system is consistent and the number of solutions is $2^{\mathsf{q}-rowrank(\boldsymbol{A})}$ where q is the number of variables in the system of equations. Moreover, if $\boldsymbol{A}$ is in Reduced Row-Echelon form, then the values of the variables corresponding to leading 1s in each row are completely determined by the values assigned to the remaining variables. The variables corresponding to the leading 1's are called *dependent* variables and the remaining variables are *free*. Let X_F and $X \setminus X_F$ denote the set of free and dependent variables respectively. Let $f = |X_F|$. Clearly $f = \mathsf{q} - rowrank(\boldsymbol{A})$. For each possible assignment to the free variables we get an assignment to the dependent variables by propagating the values through the augmented matrix in $\mathcal{O}(\mathsf{q}^2)$ time. Thus we can enumerate all 2^f satisfying assignments to a system of linear equations $\boldsymbol{A}.\boldsymbol{x} \oplus \boldsymbol{b} = \boldsymbol{y}$ if $\boldsymbol{A}$ in Reduced Row-Echelon form.

Gray Codes

A Gray code [14] is an ordering of 2^l binary numbers for some $l \geq 1$ with the property that every pair of consecutive numbers in the sequence differ in exactly one bit. Thus starting from $\mathbf{0}^l$ we can iteratively construct the entire Gray code sequence by flipping one bit in each step. We assume access to a procedure *nextGrayBit* that in each call returns the position of the next bit that is to be flipped. Such a procedure can be implemented in constant time by a trivial modification of Algorithm L in [19].

3 Related Work

Propositional model counting has been of theoretical as well as practical interest over the years [16, 17, 22, 26]. Early investigations showed that both #CNF and #DNF are #P-complete [28]. Consequently, approximation algorithms have been explored for both problems. A major breakthrough for approximate #DNF was achieved by the seminal work of Karp and Luby [17], which provided a Monte Carlo-based FPRAS for #DNF. The proposed FPRAS was improved by follow-up work of Karp, Luby and Madras [18] and Dagum et al. [5], achieving the best known complexity of $\mathcal{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$. In this work, we bring certain ideas of Karp et al. into the hashing framework with significant adaptations.

For #CNF, early work on approximate counting resulted in hashing-based schemes that required polynomially many calls to an NP-oracle [24, 27]. No practical algorithms materialized from the these schemes due to the impracticality of the underlying NP queries. Subsequent attempts to circumvent hardness led to the development of several hashing and sampling-based approaches that achieved scalability but provided very weak or no guarantees [13, 11]. Due to recent breakthroughs in the design of hashing-based techniques, several tools have been developed recently that can handle formulas involving hundreds of thousands of variables while providing rigorous formal guarantees. Overall, these tools can be broadly classified by their underlying hashing-based technique as: (i) obtain a constant factor approximation and then use identical copies of the input formula to obtain ε approximations [10], or (ii) directly obtain ε guarantees[3, 4]. The first technique when applied to DNF formulas is not an FPRAS. In contrast, Chakraborty, Meel and Vardi [4] recently showed that tools based on the latter approach, such as ApproxMC2, do provide FPRAS for #DNF counting. Chakraborty et al. did not analyze the complexity of the algorithm in their work. We now provide a precise complexity analysis of ApproxMC2 for

#DNF. To that end, we first describe the ApproxMC framework on which ApproxMC2 is built.

3.1 ApproxMC Framework

Chakraborty et al. introduced in [3] a hashing-based framework called ApproxMC that requires linear (in n) number of SAT calls. Subsequently in ApproxMC2, the number of SAT calls was reduced from linear to logarithmic (in n). The core idea of ApproxMC is to employ 2−universal hash functions to partition the solution space into *roughly equal small* cells, wherein a cell is called *small* if it has less than or equal to hiThresh solutions, such that hiThresh is a function of ε. 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 $\mathsf{hiThresh}+1$ solutions. Following the terminology of [3], we refer to the above described procedure as BSAT (bounded SAT). To determine the number of cells, ApproxMC performs a search that requires $\mathcal{O}(\log \mathsf{n})$ steps and the estimate is returned as the count of the solutions in a randomly picked small cell scaled by the total number of cells. To amplify confidence to the desired levels of $1-\delta$, ApproxMC invokes the estimation routine $\mathcal{O}(\log \frac{1}{\delta})$ times and reports the median of all such estimates. Hence, the number of BSAT invocations is $\mathcal{O}(\log \mathsf{n} \log(\frac{1}{\delta}))$.

FPRAS for #DNF

The key insight of Chakraborty et al. [4] is that the BSAT procedure can be done in polynomial time when the input formula to ApproxMC is in DNF. In particular, the input to every invocation of BSAT is a formula that is a conjunction of the input DNF formula and a set of XOR constraints derived from the hash function. Chakraborty et al. observed that one can iterate over all the cubes of the input formula, substitute each cube into the set of XOR constraints separately, and employ Gaussian Elimination to enumerate the solutions of the simplified XOR constraints. Note that at no step would one have to enumerate more than hiThresh solutions. Since Gaussian Elimination is a polynomial-time procedure, BSAT can be accomplished in polynomial time as well. Chakraborty et al. did not provide a precise complexity analysis of BSAT. We now provide such an analysis. For lack of space we defer all proofs to the full version. The following lemma states the time complexity of the BSAT routine.

▶ **Lemma 1.** *The complexity of* BSAT *when the input formula to* ApproxMC2 *is in DNF is* $\mathcal{O}(\mathsf{mn}^3 + \mathsf{mn}^2/\varepsilon^2)$.

We can now complete the complexity analysis:

▶ **Lemma 2.** *The complexity of* ApproxMC2 *is* $\mathcal{O}((\mathsf{mn}^3 + \mathsf{mn}^2/\varepsilon^2) \log \mathsf{n} \log(1/\delta))$ *when the input formula is in DNF.*

4 Efficient Hashing-based DNF Counter

We now present three key novel algorithmic innovations that allow us to design hashing-based FPRAS for #DNF with complexity similar to Monte Carlo-based state-of-the-art techniques. We first introduce a new family of 2-universal hash functions that allow us to circumvent the need for expensive Gaussian Elimination. We then discuss the concept of Symbolic Hashing, which allows us to design hash functions over a space different than the assignment space, allowing us to achieve significant reduction in the complexity of search procedure for the

Algorithm 1 enumNextREX$(\mathbf{D}, \boldsymbol{u}, \boldsymbol{v}, k)$

1: $\boldsymbol{v}' \leftarrow \boldsymbol{v}$;
2: $\boldsymbol{v}'[k] \leftarrow \neg \boldsymbol{v}[k]$;
3: $\boldsymbol{u}' \leftarrow \boldsymbol{u} \oplus \mathbf{D}[., k]$;
4: **return** $(\boldsymbol{u}', \boldsymbol{v}')$

number of the cells. Finally, we show that BSAT can be replaced by an efficient stochastic estimator. These three techniques allow us to achieve significant reduction in the complexity of hashing-based DNF counter without loss of theoretical guarantees.

4.1 Row-Echelon XOR Hash Functions

The complexity analysis presented in Section 3 shows that the expensive Gaussian Elimination contributes significantly to poor time complexity of ApproxMC2. Since the need for Gaussian Elimination originates from the usage of H_{XOR}, we seek a family of 2-universal hash functions that circumvents this need. We now introduce a Row-Echelon XOR family of hash functions defined as $H_{REX}(\mathsf{q}, \mathsf{p}) = \{\boldsymbol{A}.\boldsymbol{x} \oplus \boldsymbol{b} \mid \boldsymbol{A}^{[\mathsf{p}\times\mathsf{q}]} = [\boldsymbol{I}^{[\mathsf{p}\times\mathsf{p}]} : \mathbf{D}^{[\mathsf{p}\times(\mathsf{q}-\mathsf{p})]}]\}$ where $\boldsymbol{I}$ is the identity matrix, $\mathbf{D}$ and $\boldsymbol{b}$ are random $0/1$ matrix and vector respectively. In particular, we ensure that for every $\mathbf{D}[i,j]$ and $\boldsymbol{b}[i]$ we have $\Pr[\mathbf{D}[i,j] = 1] = \Pr[\mathbf{D}[i,j] = 0] = 0.5$ and also $\Pr[\boldsymbol{b}[i] = 1] = \Pr[\boldsymbol{b}[i] = 0] = 0.5$. Note that $\mathbf{D}$ and $\boldsymbol{b}$ completely define a hash function from H_{REX}. The following theorem establishes the desired properties of universality for H_{REX}. The proof is deferred to full version for lack of space.

▶ **Theorem 3.** *H_{REX} is 2-universal.*

The naive way of enumerating satisfying assignments for a given $\mathbf{D}^{[\mathsf{p}\times(\mathsf{q}-\mathsf{p})]}$, $\boldsymbol{b}^{[\mathsf{p}]}$, and $\boldsymbol{y}^{[\mathsf{p}]}$ is to iterate over all 2^f assignments to the free variables in sequence starting from $\mathbf{0}^{[f]}$ to $\mathbf{1}^{[f]}$, where $f = (\mathsf{q} - \mathsf{p})$. For each assignment $\boldsymbol{v}^{[f]}$ to the free variables, the corresponding assignment to the dependent variables $\boldsymbol{u}^{[\mathsf{q}-f]}$ can be calculated as $\boldsymbol{u} = (\mathbf{D}.\boldsymbol{v}) \oplus \boldsymbol{b} \oplus \boldsymbol{y}$, which requires $O(\mathsf{pq})$ time. Can we do better?

We answer the above question positively by iterating over the 2^f assignments to the free variables out of sequence. In particular, we iterate using the Gray code sequence for f bits. The procedure is outlined in enumNextREX (Algorithm 1). The algorithm takes the hash matrix $\mathbf{D}$, an assignment to the free variables $\boldsymbol{v}$, and an assignment to the dependent variables $\boldsymbol{u}$ as inputs, and outputs the next free-variable assignment $\boldsymbol{v}'$ in the Gray sequence and the corresponding assignment $\boldsymbol{u}'$ to the dependent variables. k represents the position of the bit that is changed between $\boldsymbol{v}$ and $\boldsymbol{v}'$. Thus enumNextREX constructs a satisfying assignment to a Row-Echelon XOR hash function in each invocation in $\mathcal{O}(\mathsf{q})$ time.

4.2 Symbolic Hashing

For DNF formulas, R_ϕ can be exponentially sparse compared to U, which is undesirable[2]. It is possible, however, to transform U to another space U' and the solution space R_ϕ to R'_ϕ such that the ratio $|U'|/|R'_\phi|$ is polynomially bounded and $|R_\phi| = |R'_\phi|$. For DNF formulas, the new universe U' is defined as $U' = \{(\boldsymbol{x}, \phi^{Ci}) \mid \boldsymbol{x} \models \phi^{Ci}\}$. Thus, corresponding to each $\boldsymbol{x} \models \phi$ that satisfies cubes $\phi^{Ci_1}, ..\phi^{Ci_{\boldsymbol{x}}}$ in ϕ, we have the states $\{(\boldsymbol{x}, \phi^{Ci_1}), (\boldsymbol{x}, \phi^{Ci_2})..(\boldsymbol{x}, \phi^{Ci_{\boldsymbol{x}}})\}$ in

[2] Number of steps of ApproxMC2 search procedure increases with sparsity

Algorithm 2 SymbolicDNFApproxMCCore(ϕ, hiThresh)

1: $\mathsf{w} \leftarrow$ width of cubes;
2: $\mathsf{q} \leftarrow \mathsf{n} - \mathsf{w} + \log \mathsf{m}$;
3: $sI \leftarrow \mathsf{n} - \mathsf{w} - \log \mathsf{hiThresh}$;
4: $(\hat{\mathbf{D}}^{[(\mathsf{q}-1)\times(\mathsf{q}-sI)]}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}) \leftarrow \mathsf{SampleBase}(\mathsf{q}, sI)$;
5: $\mathsf{p} \leftarrow \mathsf{LogSATSearch}(\phi, \hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}, \mathsf{hiThresh}, sI, \mathsf{q}-1)$;
6: $\mathsf{solCount} \leftarrow \mathsf{BSAT}(\phi, \hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}, \mathsf{hiThresh}, \mathsf{p}, \mathsf{q}, sI)$;
7: **return** $(2^{\mathsf{p}}, \mathsf{solCount})$

U'. Next, the solution space is defined as $R'_\phi = \{(\boldsymbol{x}, \phi^{Ci}) \mid \boldsymbol{x} \models \phi^{Ci} \text{ and } \forall j < i, \boldsymbol{x} \not\models \phi^{Cj}\}$ for a fixed ordering of the cubes. The definition of R'_ϕ ensures that $|R_\phi| = |R'_\phi|$. This transformation is due to Karp and Luby [17].

The key idea of Symbolic Hashing is to perform 2-universal hashing symbolically over the transformed space. In particular, the sampled hash function partitions the space U' instead of U. Therefore, we employ hash functions from $H_{REX}(\mathsf{q}, \mathsf{p})$ over $\mathsf{q} = \mathsf{n} - \mathsf{w} + \log \mathsf{m}$ variables instead of n variables. Note that the variables of a satisfying assignment $\boldsymbol{z} \in \{0,1\}^{\mathsf{q}}$ to the hash function are now different from the variables to a satisfying assignment $\boldsymbol{x} \in \{0,1\}^{\mathsf{n}}$ of the input formula ϕ. We interpret $\boldsymbol{z}$ as follows: the last $\log \mathsf{m}$ bits of $\boldsymbol{z}$ are converted to a number i such that $1 \leq i \leq \mathsf{m}$. Now ϕ^{Ci} corresponds to a partial assignment of $width[\phi^{Ci}]$ variables in that cube. For simplicity, we assume that each cube is of the same width w.[3] The remaining $\mathsf{n} - \mathsf{w}$ bits of $\boldsymbol{z}$ are interpreted to be the assignment to the $\mathsf{n} - \mathsf{w}$ variables not in ϕ^{Ci} giving a complete assignment $\boldsymbol{x}$. Thus we get a pair $(\boldsymbol{x}, \phi^{Ci})$ from $\boldsymbol{z}$ such that $\boldsymbol{x} \models \phi^{Ci}$. For a fixed ordering of variables and cubes we see that there is a bijection between $(\boldsymbol{x}, \phi^{Ci})$ and $\boldsymbol{z}$ and hence the 2-universality guarantee holds over the partitioned space of U'.

4.3 Stochastic Cell-Counting

To estimate the number of solutions in a cell, we need to check for every tuple $(\boldsymbol{x}, \phi^{Ci})$ generated using symbolic hash function as described above: if $(\boldsymbol{x}, \phi^{Ci}) \in R'_\phi$. Such a check would require iteration over cubes ϕ^{Cj} for $1 \leq j \leq (i-1)$ and returning no if $\boldsymbol{x} \models \phi^{Cj}$ for some j and yes otherwise. This would result in procedure with $O(\mathsf{mn})$ complexity.

Our key observation is that a precise count of the number of solutions in a cell is not required and therefore, one can employ a stochastic estimator for the number of solutions in a cell. We proceed as follows: we define the *coverage* of an assignment $\boldsymbol{x}$ as $cov(\boldsymbol{x}) = \{j | \boldsymbol{x} \models \phi^{Cj}\}$. Note that $\sum_{(\boldsymbol{x}, \phi^{Ci}) \in U'} \frac{1}{|cov(\boldsymbol{x})|} = |R_\phi|$.

We define a random variable $\mathsf{c}_{\boldsymbol{x}}$ as the number of steps taken to uniformly and independently sample from $\{1, 2, \ldots, \mathsf{m}\}$, a number j such that $\boldsymbol{x} \models \phi^{Cj}$. For a randomly chosen j, the probability $\Pr[\boldsymbol{x} \models \phi^{Cj}] = |cov(\boldsymbol{x})|/\mathsf{m}$, which follows the Bernoulli distribution. The random variable $\mathsf{c}_{\boldsymbol{x}}$ is the number of Bernoulli trials for the first success, which follows the geometric distribution. Therefore, $\mathsf{E}[\mathsf{c}_{\boldsymbol{x}}] = \mathsf{m}/|cov(\boldsymbol{x})|$, and $\mathsf{E}[\mathsf{c}_{\boldsymbol{x}}/\mathsf{m}] = 1/|cov(\boldsymbol{x})|$. The estimator $\mathsf{c}_{\boldsymbol{x}}/\mathsf{m}$ has been previously employed by Karp et al. [18]. Here, we show that it can also be used for Stochastic Cell-Counting: we define the estimator for the number of solutions in a cell as $\Omega_{\boldsymbol{y}} = \sum_{(\boldsymbol{x}, \phi^{Ci}) \in h^{-1}(\boldsymbol{y})} \mathsf{c}_{\boldsymbol{x}}/\mathsf{m}$.

[3] We can handle non-uniform width cubes by sampling ϕ^{Ci} with probability $\frac{2^{\mathsf{n}-width[\phi^{Ci}]}}{\sum_{j=1}^{\mathsf{m}} 2^{\mathsf{n}-width[\phi^{Cj}]}}$ instead of uniformly

Algorithm 3 SymbolicDNFApproxMC$(\phi, \varepsilon, \delta)$

1: hiThresh $\leftarrow 2 * (1 + 9.84\left(1 + \frac{\varepsilon}{1+\varepsilon}\right)\left(1 + \frac{1}{\varepsilon}\right)^2)$;
2: $t \leftarrow \lceil 17 \log_2(3/\delta) \rceil$;
3: EstimateList $\leftarrow$ emptyList; iter $\leftarrow 0$;
4: **repeat**
5: iter $\leftarrow$ iter $+ 1$;
6: (cellCount, solCount) $\leftarrow$ SymbolicDNFApproxMCCore$(\phi,$ hiThresh$)$;
7: **if** (cellCount $\neq \perp$) **then** AddToList(EstimateList, solCount $\times$ cellCount);
8: **until** (iter $\geq t$);
9: finalEstimate $\leftarrow$ FindMedian(EstimateList);
10: **return** finalEstimate

Algorithm 4 SampleBase(q, sI)

1: Sample $\mathbf{G}$ uniformly from $\{0,1\}^{[sI \times (\mathsf{q}-sI)]}$;
2: Sample uniformly an upper triangular matrix $\mathbf{E}^{[(\mathsf{q}-sI-1)\times(\mathsf{q}-sI)]}$ with $\mathbf{E}[i,i] = 1$ for all i.
3: $\hat{\mathbf{D}} \leftarrow [\frac{\mathbf{G}}{\mathbf{E}}]$;
4: Sample $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ uniformly from $\{0,1\}^{\mathsf{q}-1}$;
5: **return** $\hat{\mathbf{D}}, \hat{b}, \hat{y}$

4.4 The Full Algorithm

We now incorporate the above techniques into ApproxMC2 and call the revised algorithm SymbolicDNFApproxMC, which is presented as Algorithm 3. First, note that expression for hiThresh is twice that for ApproxMC2. Then, in line 4, a matrix $\hat{\mathbf{D}}$ and vectors $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ are obtained, which are employed to construct an appropriate hash function and cell during the search procedure of SymbolicDNFApproxMCCore. SymbolicDNFApproxMC makes $t = O(\log(1/\delta))$ calls to SymbolicDNFApproxMCCore (line 4-8) and returns median of all the estimates (lines 9-10) to boost the probability of success to $1 - \delta$.

We now discuss the subroutine SymbolicDNFApproxMCCore, which is an adaptation of ApproxMC2Core but with significant differences. First, for DNF formulas with cube width w, the number of solutions is lower bounded by $2^{\mathsf{n}-w}$. Therefore, instead of starting with 1 hash constraint, we can safely start with $sI = \mathsf{n} - w - \log \mathsf{hiThresh}$ constraints (lines 3-4). Thereafter, SymbolicDNFApproxMCCore calls LogSATSearch in line 5 to find the right number p of constraints. The cell count with p constraints is calculated in line 6 and the estimate $(2^{\mathsf{p}}, \mathsf{solCount})$ is returned in line 7.

SampleBase algorithm constructs the base matrix $\hat{\boldsymbol{D}}$ and base vectors $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ required for sampling from H_{REX} family. $\mathbf{G}$ is a random matrix of dimension $sI \times (\mathsf{q} - sI)$ and $\mathbf{E}$ is a random upper triangular matrix of dimension $(\mathsf{q} - sI - 1) \times (\mathsf{q} - sI)$ with all diagonal elements 1. In line 3, $\hat{\boldsymbol{D}}$ is constructed as the vertical concatenation $[\frac{\mathbf{G}}{\mathbf{E}}]$.

LogSATSearch (algorithm 5) performs a binary search to find the number of constraints p at which the cell count falls below hiThresh. For DNF formula with cube width of w, since the number of solutions is bounded between $2^{\mathsf{n}-w}$ and $\mathsf{m} * 2^{\mathsf{n}-w}$, we need to perform search for p between $\mathsf{n} - w$ and $\mathsf{n} - w + \log \mathsf{m}$. Therefore, binary search can take at most $O(\log \log \mathsf{m})$ steps to find correct p.

Symbolic Hashing is implemented in Algorithm 6 (BSAT). In line 2, we obtain a hash function from $H_{REX}(\mathsf{q}, \mathsf{p})$ over $\mathsf{q} = \mathsf{n} - \mathsf{w} + \log \mathsf{m}$ variables by calling Extract. We assume access to a procedure *nextGrayBit* in line 10 that returns the position of the bit that is

Algorithm 5 LogSATSearch($\phi, \hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}$, hiThresh, low, hi)

```
1: lowerFib ← 0; upperFib ← hi − low + 1; p ← low;
2: FailRecord[0] ← 1; FailRecord[hi − low + 1] ← 0;
3: FailRecord[i] ← ⊥ for all i other than 0 and hi − low + 1;
4: while true do
5:     C_BSAT ← BSAT(φ, D̂, b̂, ŷ, hiThresh, p, q, sI);
6:     if (C_BSAT ≥ hiThresh) then
7:         if (FailRecord[p + 1 − low + 1] = 0) then return p + 1;
8:         FailRecord[i] ← 1 for all i ∈ {1, ... p − low + 1};
9:         lowerFib ← p − low + 1;
10:        p ← (upperFib + lowerFib)/2;
11:    else
12:        if (FailRecord[p − 1 − low + 1] = 1) then return p;
13:        FailRecord[i] ← 0 for all i ∈ {p, ... hi − low + 1};
14:        upperFib ← p − low + 1;
15:        p ← (upperFib + lowerFib)/2;
```

Algorithm 6 BSAT($\phi, \hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}$, hiThresh, p, q, sI)

```
1: count ← 0;
2: D, b, y ← Extract(D̂, b̂, ŷ, p, q, sI);
3: u^p ← b ⊕ y;
4: v^(q−p) ← 0^(q−p);
5: for (j = 0; j < 2^(q−p); j++) do
6:     z ← [u : v];
7:     (x, φ^Ci) = interpret(z);
8:     count = count + CheckSAT(x, φ^Ci, count, hiThresh);
9:     if count ≥ hiThresh then return hiThresh;
10:    k ← nextGrayBit(q − p, j);
11:    (u, v) ← enumNextREX(D, u, v, k);
12: return count
```

flipped between two consecutive assignments. A satisfying assignment $\boldsymbol{z}$ to the hash function is constructed in line 6. $\boldsymbol{z}$ is interpreted to generate a pair $(\boldsymbol{x}, \phi^{Ci})$ in line 7 which is checked for satisfiability in line 8. The final cell count is returned in line 12.

In CheckSAT (algorithm 7), we implement the stochastic cell counting procedure. The key idea is to sample cubes uniformly at random from $\{1, 2 \ldots \mathsf{m}\}$ till a cube ϕ^{Cj} is found such that $\boldsymbol{x} \models \phi^{Cj}$ (lines 2-5). The number of cubes sampled $\mathsf{c}_{\boldsymbol{x}}$ divided by total number of cubes m is the estimate returned (line 6).

Procedure LogSATSearch in SymbolicDNFApproxMC is based upon the LogSATSearch in ApproxMC2[4]. As noted in the analysis of ApproxMC2, such a logarithmic search procedure requires that the solution space for a hash function with $\mathsf{p}+1$ hash constraints is a subset of the solution space with p hash constraints. Furthermore, we want to preserve Row-Echelon nature of the resulting hash constraints. To this end, we first construct $\mathbf{D}^{[\mathsf{q}\times(\mathsf{q}-\mathsf{p})]}$ and $\boldsymbol{b}^{[\mathsf{q}-1]}$ as follows:

To seed the construction procedure, in SampleBase (algorithm 4) we first randomly sample a 0/1 vector $\hat{\boldsymbol{b}}$ of size $\mathsf{q}-1$ which is the maximum number of hash constraints possible. We

Algorithm 7 CheckSAT($\boldsymbol{x}, \phi^{Ci}, count, \mathsf{hiThresh}$)

1: $\mathsf{c}_{\boldsymbol{x}} \leftarrow 0$;
2: **while** $count + \mathsf{c}_{\boldsymbol{x}}/m < \mathsf{hiThresh}$ **do**
3: Uniformly sample j from $\{1, 2, .., \mathsf{m}\}$;
4: $\mathsf{c}_{\boldsymbol{x}} \leftarrow \mathsf{c}_{\boldsymbol{x}} + 1$;
5: **if** $\boldsymbol{x} \models \phi^{Cj}$ **then**
6: **return** $\mathsf{c}_{\boldsymbol{x}}/\mathsf{m}$;
7: **return** $\mathsf{c}_{\boldsymbol{x}}/\mathsf{m}$

Algorithm 8 Extract($\hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}, \mathsf{p}, \mathsf{q}, sI$)

1: $\hat{\mathbf{D}}' \leftarrow \hat{\mathbf{D}}[0 : \mathsf{p}, 0 : \mathsf{q} - sI]$; $\boldsymbol{b} \leftarrow \hat{\boldsymbol{b}}[0 : \mathsf{p}]$;
2: $\boldsymbol{y} \leftarrow \hat{\boldsymbol{y}}[0 : \mathsf{p}]$;
3: **for** $(i = sI;\ i < \mathsf{p}; i++)$ **do**
4: **for** $(j = 0;\ j < i; j++)$ **do**
5: **if** $\hat{\mathbf{D}}'[j, (i - sI)] == 1$ **then**
6: $\hat{\mathbf{D}}'[j, .] \leftarrow \hat{\mathbf{D}}'[j, .] \oplus \hat{\mathbf{D}}'[i, .]$;
7: $\boldsymbol{b}[j] \leftarrow \boldsymbol{b}[j] \oplus \boldsymbol{b}[i]$;
8: $\boldsymbol{y}[j] \leftarrow \boldsymbol{y}[j] \oplus \boldsymbol{y}[i]$;
9: $\mathbf{D} \leftarrow \hat{\mathbf{D}}'[0 : \mathsf{p}, \mathsf{p} - sI : \mathsf{q} - sI]$;
10: **return** $\mathbf{D}, \boldsymbol{b}, \boldsymbol{y}$

then construct a 0/1 matrix $\hat{\mathbf{D}}$ as follows: $\hat{\mathbf{D}}^{[(\mathsf{q}-1)\times(\mathsf{q}-sI)]} = [\frac{\mathbf{G}}{\mathbf{E}}]$ where matrix $\mathbf{G}^{[sI\times(\mathsf{q}-sI)]}$ is a random 0/1 matrix with sI rows, and matrix $\mathbf{E}^{[(\mathsf{q}-sI)\times(\mathsf{q}-sI)]}$ is defined as as follows:

- $\mathbf{E}[i, j] = 1$ if $i = j$
- $\mathbf{E}[i, j] = 0$ if $i > j$
- $\Pr[\mathbf{E}[i, j] = 1] = \Pr[\mathbf{E}[i, j] = 0] = 0.5$ if $i < j$

The reason for this definition of $\hat{\mathbf{D}}$ is that for DNF counting we have a good lower bound on the number of hash constraints we can start with. The number of rows in $\mathbf{G}$ corresponds to this lower bound. The definition of $\mathbf{E}$ ensures that the rows of $\mathbf{E}$ are linearly independent which results in a monotonically shrinking solution space.

The Extract procedure (algorithm 8) takes $\hat{\mathbf{D}}$,$\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ and a number p as input and returns $\mathbf{D}$,$\boldsymbol{b}$ and cell $\boldsymbol{y}$ such that $(\mathbf{D}, \boldsymbol{b})$ represents a hash function from H_{REX} with p constraints and $\boldsymbol{y}$ represents a cell. A precondition for Extract is $sI \leq \mathsf{p} \leq \mathsf{q} - 1$. In lines 1 and 2, the first p rows of $\hat{\mathbf{D}}$ and first p elements of $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ are selected as $\hat{\mathbf{D}}'$, $\boldsymbol{b}$ and $\boldsymbol{y}$ respectively. The first sI rows of $\hat{\mathbf{D}}'$ form the matrix $\mathbf{G}$ in the definition of $\hat{\mathbf{D}}$ and the remaining $\mathsf{p} - sI$ rows of $\hat{\mathbf{D}}'$ are the first $\mathsf{p} - sI$ rows of matrix $\mathbf{E}$. Each row from sI to p is used to reduce the preceding rows in lines 5 to 8 so that the only non-zero elements of the first $\mathsf{p} - sI$ columns are the leading 1s in rows sI to p. Thus Extract ensures that for a given $\hat{\mathbf{D}}$,$\boldsymbol{b}$ and $\hat{\boldsymbol{y}}$, the solution space of $\mathbf{D}^{[\mathsf{p}\times(\mathsf{q}-\mathsf{p})]}$,$\boldsymbol{b}^{[\mathsf{p}]}$ and $\boldsymbol{y}^{[\mathsf{p}]}$ is a superset of solution space of $\mathbf{D}^{[(\mathsf{p}+1)\times(\mathsf{q}-\mathsf{p}-1)]}$,$\boldsymbol{b}^{[\mathsf{p}+1]}$ and $\boldsymbol{y}^{[\mathsf{p}+1]}$ for all p.

5 Analysis

In order to prove the correctness of SymbolicDNFApproxMC, we first state the following helper lemma. We defer the proofs to the full version due to lack of space.

▶ **Lemma 4.** *For every* $1 \leq \mathsf{p} \leq \mathsf{q}$ *and let* $\mu_{\mathsf{p}} = |R_\phi|/2^{\mathsf{p}}$. *For every* $\beta > 0$ *and* $0 < \varepsilon < 1$ *we have*

1. $\Pr[|\Omega_{\boldsymbol{y}} - \mu_{\mathsf{p}}| > \frac{\varepsilon}{(1+\varepsilon)}\mu_{\mathsf{p}}] \leq \frac{2}{\frac{\varepsilon^2}{(1+\varepsilon)^2}\mu_{\mathsf{p}}}$
2. $\Pr[\Omega_{\boldsymbol{y}} \leq \beta\mu_{\mathsf{p}}] \leq \frac{2}{2+(1-\beta^2)\mu_{\mathsf{p}}}$

The difference in lemma 4 and lemma 1 in [4] is that the probability bounds differ by a factor of 2. We account for this difference by making hiThresh in SymbolicDNFApproxMC twice the value of hiThresh in ApproxMC2. Therefore the rest of the proof of Theorem 7 (below) is exactly the same as the proof of Theorem 4 of [4]. For completeness, we first restate lemmas 2 and 3 from [4] below.

In the following, T_{p} denotes the event $(\Omega_{\boldsymbol{y}} < \mathsf{hiThresh})$, and L_{p} and U_{p} denote the events $(\Omega_{\boldsymbol{y}} < \frac{|R_\phi|}{(1+\varepsilon)2^{\mathsf{p}}})$ and $(\Omega_{\boldsymbol{y}} > \frac{|R_\phi|}{2^{\mathsf{p}}}(1 + \frac{\varepsilon}{1+\varepsilon}))$ respectively. p^* denotes the integer $\lfloor \log_2 |R_\phi| - \log_2(4.92(1 + \frac{1}{\varepsilon})^2) \rfloor$

▶ **Lemma 5.** *The following bounds hold:* 1) $\Pr[T_{\mathsf{p}^*-3}] \leq \frac{1}{62.5}$ 2) $\Pr[L_{\mathsf{p}^*-2}] \leq \frac{1}{20.68}$ 3) $\Pr[L_{\mathsf{p}^*-1}] \leq \frac{1}{10.84}$ 4) $\Pr[L_{\mathsf{p}^*} \cup U_{\mathsf{p}^*}] \leq \frac{1}{4.92}$

Let B denote the event that SymbolicDNFApproxMC returns a pair $(2^{\mathsf{p}}, nSols)$ such that $2^{\mathsf{p}} * nSols$ does not lie in the interval $[\frac{|R_\phi|}{1+\varepsilon}, |R_\phi|(1+\varepsilon)]$.

▶ **Lemma 6.** $\Pr[B] \leq 0.36$

▶ **Theorem 7.** *Let* SymbolicDNFApproxMC$(\phi, \varepsilon, \delta)$ *return count* c. *Then* $\Pr[|R_\phi|/(1+\varepsilon) \leq c \leq (1+\varepsilon)|R_\phi|] \geq 1 - \delta$.

Theorem 7 follows from lemmas 4, 5 and 6 and noting that SymbolicDNFApproxMC boosts the probability of correctness of the count returned by SymbolicDNFApproxMCCore to $1 - \delta$ by using median of $t = O(\log(1/\delta))$ calls.

▶ **Theorem 8.** SymbolicDNFApproxMC *runs in* $\tilde{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$ *time.*[4]

6 Conclusion

Hashing-based techniques have emerged as a promising approach to obtain counting algorithms and tools that scale to large instances while providing strong theoretical guarantees. This has led to an interest in designing hashing-based algorithms for counting problems that are known to be amenable to fully polynomial randomized approximation schemes. The prior hashing-based approach [4] provided FPRAS for DNF but with complexity much worse than state-of-the-art techniques. In this work, we introduced (i) Symbolic Hashing, (ii) Stochastic Cell-Counting, and (iii) a new 2-universal family of hash functions, and obtained a hashing-based FPRAS for #DNF with complexity similar to state-of-the-art.

Given the recent interest in hashing-based techniques and generality of our contributions, we believe concepts introduced in this paper can lead to design of hashing-based techniques for other classes of constraints. For example, all prior versions of ApproxMC relied on deterministic SAT solvers for exactly counting the solutions in a cell for #CNF. The technique of Stochastic Cell-Counting opens up the door for the usage of probabilistic SAT solvers for #CNF. Furthermore, a salient feature of the H_{REX} family is the sparsity of its hash functions. In fact, the sparsity increases with the addition of constraints. Sparse hash functions have been shown to be desirable for efficiently solving CNF+XOR constraints [15, 9, 12]. An interesting direction for future work is to test H_{REX} family with CNF formulas.

[4] We say $f(n) \in \tilde{O}(g(n))$ if $\exists k : f(n) \in \mathcal{O}(g(n)\log^k(g(n))$

Acknowledgements. The authors thank Jeffrey Dudek, Supratik Chakraborty and Dror Fried for valuable discussions.

References

1 Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #sat and bayesian inference. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 340–351. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238208.

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

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

4 S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. of IJCAI*, 2016.

5 Paul Dagum, Richard Karp, Michael Luby, and Sheldon Ross. An optimal algorithm for monte carlo estimation. *SIAM Journal on computing*, 29(5):1484–1496, 2000.

6 Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal—The International Journal on Very Large Data Bases*, 16(4):523–544, 2007.

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

8 Leonardo Duenas-Osorio, Kuldeep S Meel, Roger Paredes, and Moshe Y Vardi. Counting-based reliability estimation for power-transmission grids. In *AAAI*, pages 4488–4494, 2017.

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

10 Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proc. of ICML*, pages 334–342, 2013.

11 V. Gogate and R. Dechter. Approximate counting by sampling the backtrack-free search space. In *Proc. of the AAAI*, volume 22, page 198, 2007.

12 C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. Short XORs for Model Counting; From Theory to Practice. In *SAT*, pages 100–106, 2007.

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

14 Frank Gray. Pulse code communication, 17 1953. US Patent 2,632,058.

15 Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, 21(1):41–58, 2016. doi:10.1007/s10601-015-9204-z.

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

17 R.M. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. *Proc. of FOCS*, 1983.

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

19 Donald E Knuth. Generating all n-tuples. *The Art of Computer Programming*, 4, 2004.

20 James D Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research*, pages 101–133, 2006.

21 Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996. doi:10.1016/0004-3702(94)00092-1.

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

23 T. Sang, P. Beame, and H. Kautz. Performing bayesian inference by weighted model counting. In *Prof. of AAAI*, pages 475–481, 2005.

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

25 Gilbert Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.

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

27 L. Trevisan. 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`.

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