

RICE UNIVERSITY

On Hashing-Based Approaches to Approximate
DNF-Counting

by

Aditya A. Shrotri

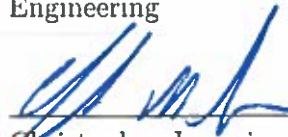
A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:



Moshe Y. Vardi, Chair
Karen Ostrum George Distinguished
Service Professor in Computational
Engineering



Christopher Jermaine
Professor of Computer Science



Devika Subramanian
Professor of Computer Science

Houston, Texas

October, 2017

ABSTRACT

On Hashing-Based Approaches to Approximate DNF-Counting

by

Aditya A. Shrotri

Propositional model counting is a fundamental problem in AI. For DNF formulas, Monte Carlo-based techniques provide a fully polynomial randomized approximation scheme (FPRAS). For CNF constraints, hashing-based techniques are highly successful. It was recently shown that hashing techniques also yield an FPRAS for DNF counting. Our analysis, however, shows that the proposed hashing approach provides poor time complexity compared to the Monte Carlo techniques, for DNF Counting. Given the success of hashing techniques for CNF constraints, it is natural to ask: Can hashing techniques provide an efficient FPRAS for DNF counting? We provide a positive answer to this question. We introduce two novel algorithmic techniques: *Symbolic Hashing* and *Stochastic Cell Counting*, and a new family of *Row-Echelon hash functions*. We design a hashing-based FPRAS of similar complexity (up to poly-log factors) as that of prior works. We also provide an empirical comparison of the various approaches.

Acknowledgements

First, I would like to thank my adviser Prof. Moshe Y. Vardi for his astute guidance during the past three years. This thesis would not have been possible without his support. His suggestions for improvements proved to be indispensable.

I would also like to thank Dr. Kuldeep S. Meel, not only as a collaborator on this work, but also for his mentorship during the early days of grad school. As an office mate, working late nights was always something to look forward to.

I am grateful to all the members of our research group LAPIS, especially Dr. Dror Fried for verifying some of the proofs and also for helping me polish the presentation of this work despite our quibbles. I thank Suguman Bansal and Shufang Zhu for keeping me company during the late nights in office, the car rides home, coffee breaks and for keeping me on track for deadlines. I could always rely on Jeff Dudek, Kevin Smith, Lucas Tabajara and Abhinav Verma for talking about both technical and non-technical stuff whenever I was stuck.

I would like to thank my roommate Yash Khemka for keeping the house in a respectable state in spite of my neglect. Going through similar experiences in the second year of grad school helped ease some of the anxiety. I also thank the regular members of the badminton and cricket groups for much needed recreation in the evenings.

Finally, I am indebted to my family, especially my grandparents, my uncles and aunt and their families for always being in my corner. I thank Dr. Tanvi Modi without whom life in grad school would have been very bland; even meaningless. I would like to extend my deepest gratitude to my parents Ulka and Aniruddha Shrotri without whose love, support and understanding I would not be where I am today.

Contents

Abstract	ii
List of Illustrations	vi
List of Tables	vii
1 Introduction	1
1.1 Background	1
1.2 Contributions	3
1.3 Organization	4
2 Preliminaries	5
2.1 DNF Formulas and Counting	5
2.2 Monte Carlo Algorithms	6
2.2.1 Vanilla Monte Carlo	7
2.2.2 DKLR Monte Carlo	8
2.3 Monte Carlo for DNF	8
2.3.1 KL Counter	9
2.3.2 KLM Counter	9
2.3.3 Vazirani Counter	9
2.4 Matrix Notation	10
2.5 Hash Functions	11
2.6 Gaussian Elimination	11
2.7 Gray Codes	12
3 Related Work	13

3.1	ApproxMC Framework	14
3.1.1	FPRAS for #DNF	15
4	Efficient Hashing-based DNF Counter	20
4.1	Row-Echelon XOR Hash Functions	20
4.2	Symbolic Hashing	23
4.3	Stochastic Cell-Counting	24
4.4	The Full Algorithm	24
5	Analysis	31
6	Empirical Evaluation	35
6.1	Previous Experimental Work	35
6.2	Algorithm Suite	36
6.3	Implementation Details	38
6.4	Set Up	38
6.5	Benchmarks	38
6.6	Experimental Results	41
6.6.1	DNFAproxMC outperforms SymbolicDNFAproxMC on the benchmarks we tested	42
6.6.2	No algorithm consistently outperforms all other algorithms . .	43
6.6.3	DNFAproxMC outperforms all other algorithms on certain benchmarks	43
6.7	Other techniques	45
7	Conclusion	47
	Bibliography	49

Illustrations

6.1	Solution density μ (estimated) vs #link variables for $n = m = 30000, w = 100$	40
6.2	DNFAproxMC is the best performer when $n = m = 30000, 15 \leq w \leq 40, \eta = 20000, \varepsilon = 0.8, \delta = 0.36, 300s$ timeout. (All algorithms except DNFAproxMC and KLM Counter timed out)	42
6.3	KLM Counter is the best performer when $n = m = 30000, w = 100, 2000 \leq \eta \leq 26000, \varepsilon = 0.8, \delta = 0.36, 300s$ timeout.	44
6.4	Naive Counter w/ DKLR-MC is the best performer when $n = m = 30000, 2 \leq w \leq 9, \eta = 400, \varepsilon = 0.8, \delta = 0.36, 300s$ timeout.	44
6.5	KL Counter w/ DKLR-MC is the best performer when $n = m = 30000, w = 100, \eta = 1, \varepsilon \leq 0.3, \delta = 0.36, 300s$ timeout.	45

Tables

Chapter 1

Introduction

1.1 Background

Propositional model counting is a fundamental problem in artificial intelligence with a wide range of applications including probabilistic inference, databases, decision making under uncertainty, and the like [1, 2, 3, 4]. 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 [5]. 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, 6, 7, 4], while evaluation of queries for probabilistic database reduce to $\#DNF$ instances [2]. Consequently, both $\#CNF$ and $\#DNF$ have been of theoretical as well as practical interest over the years [8, 9, 10, 11]. In his seminal paper, Valiant [5] 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 [12, 13, 9], but approximate $\#CNF$ is NP-hard [11]. 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 [13, 9] 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 [14, 15, 16, 17, 11], and are effective even for problems with existing FPRAS such as *network reliability* [18]. 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 small 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** [14, 15] 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 $\#CNF$ [9], 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 [15], who showed that hashing-based framework of **ApproxMC** could be employed

to obtain FPRAS for $\#DNF$ counting*. There is, however, no precise complexity analysis in [15]. 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?*

1.2 Contributions

In this thesis, 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 $\tilde{O}(mn \log(1/\delta)/\varepsilon^2)$, which is within polylog factors of the complexity achieved by Karp et al. [9]. 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. This theoretical part of the thesis is based on [20].

We also provide an empirical evaluation of the hashing-based and monte-carlo based techniques on randomly generated DNF Formulas. Our experiments reveal that the problem of approximate DNF-Counting is much more nuanced than what

*It is worth noting that several hashing-based algorithms based on [16, 19] do not lead to FPRAS for $\#DNF$ despite close similarity to Chakraborty et al.’s approach

the worst case complexity analysis suggests. In particular, algorithms with better worst-case complexity do not necessarily perform better than others in practice. Furthermore, different algorithms perform better on different classes of benchmarks. This suggests that the 'best' approach to approximate DNF-Counting might be to use a portfolio of the better-performing algorithms.

1.3 Organization

The rest of the thesis is organized as follows: we introduce notation in chapter 2 and discuss related work in chapter 3. We describe our main contributions in chapter 4, analyze the resulting algorithm in chapter 5, describe empirical evaluation in chapter 6 and discuss future work and conclude in chapter 7.

Chapter 2

Preliminaries

2.1 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, \dots, x_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^{C_i} . Thus $\phi = \phi^{C_1} \vee \phi^{C_2} \vee \dots \vee \phi^{C_m}$ where each ϕ^{C_i} 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^{C_i} is called its *width* and is denoted by $\text{width}[\phi^{C_i}]$.

An assignment to all the variables can be represented by a vector $\mathbf{x} \in \{0, 1\}^n$ with 1 corresponding to *true* and 0 to *false*. $U = \{0, 1\}^n$ is the set of all possible assignments, which we refer to as the *universe* or state space interchangeably. An assignment \mathbf{x} is called a satisfying assignment for a formula ϕ if ϕ evaluates to *true* under \mathbf{x} . In other words \mathbf{x} satisfies ϕ and is denoted as $\mathbf{x} \models \phi$. Note that an assignment \mathbf{x} will satisfy a DNF formula ϕ if $\mathbf{x} \models \phi^{C_i}$ 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 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[5]. The DNF-Counting problem is an example of this formalism: A formula ϕ is a problem instance and a satisfying assignment \mathbf{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 [21]. 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 $|s|$, $1/\varepsilon$, $\log(1/\delta)$ [13]. 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.

2.2 Monte Carlo Algorithms

Monte Carlo algorithms are randomized algorithms whose output can be wrong with a certain (usually small) probability [22]. They rely on drawing independent random samples to obtain numerical results. We refer the reader to [23] for further details. In the context of counting, the abstract Monte Carlo framework for finding $|R_s|$ in some universe U is shown in Algorithm 1.

Algorithm 1 Monte Carlo Framework for Counting

```

1:  $C \leftarrow 0$ 

2: repeat  $N$  times

3:   Select an element  $t \in U$  uniformly at random

4:   if  $t \in R_s$  then

5:      $C \leftarrow C + \frac{1}{N}$ 

6:  $Y \leftarrow C * |U|$ 

7: return  $Y$ 

```

C is an unbiased estimator for $\mu = |R_s|/|U|$ which is also called the density of solutions, and Y is an unbiased estimator for $|R_s|$. A concrete algorithm can be obtained from the above framework by instantiating U and R_s for the specific problem being solved.

Using Chebyshev's inequality, it can be shown that if $N \geq \frac{V[C]}{E[C]^2} \log(2/\delta)/\varepsilon^2$ then $\Pr[\frac{1}{1+\varepsilon}|R_s| \leq Y \leq (1+\varepsilon)|R_s|] \geq 1 - \delta$. In general, it is usually only possible to find a lower bound on N since $\frac{V[C]}{E[C]^2}$ is not easy to compute. There are two flavors of algorithm 1 based on how N is obtained:

2.2.1 Vanilla Monte Carlo

We refer to the standard practice of fixing N before invoking algorithm 1 as Vanilla Monte Carlo (V-MC). As $V[C] < E[C]$, $\frac{V[C]}{E[C]^2}$ can be approximated by $\frac{1}{\mu}$. A good lower bound on μ , therefore, suffices to compute N . If it is not possible to obtain a tight bound on μ or μ is very small, then the algorithm becomes slow. However, there is no restriction to the range of values for the V-MC estimator.

2.2.2 DKLR Monte Carlo

We refer to the sampling framework introduced by Dagum et al. [12] as DKLR Monte Carlo (DKLR-MC), after the initials of the authors. The number of samples required to obtain (ε, δ) guarantees is calculated as part of the algorithm. The algorithm proceeds in 3 phases: in the first phase a loose bound on μ is obtained. In phase 2 a bound on variance is obtained using the previous bound on μ . In phase 3 the final count is obtained based on the estimates of variance and μ obtained previously. Dagum et al. show that their algorithm requires close to theoretically optimal number of samples. Therefore this technique is superior to V-MC when a tight a priori bound is not available. However, unlike V-MC, DKLR-MC only allows estimators with values constrained to the interval $[0,1]$. V-MC is thus more general than DKLR-MC.

2.3 Monte Carlo for DNF

Algorithm 1 is an FPRAS if N is polynomial in size of t , and if steps 1 and 2 can be performed in polynomial time. This depends on the specific problem under consideration. For a DNF formula ϕ with n variables and m cubes, we can apply the above algorithm naively by defining U to be set of all assignments over n variables and R_ϕ to be the set of satisfying assignments to ϕ . The best lower bound on R_ϕ is 2^{n-w} , where w is the minimum width of all cubes of ϕ . If w is a small constant, then $\frac{1}{\mu} \geq \frac{1}{2^w}$ which is polynomial in n and m and hence we get an FPRAS. If however w is $O(n)$, then $\frac{1}{\mu}$ becomes exponential in n and the algorithm is no longer an FPRAS. However, by redefining U and R_ϕ in ways that ensure $\frac{1}{\mu}$ polynomial for all ϕ , Karp et al. developed counters that are FPRAS for $\#DNF$.

2.3.1 KL Counter

Karp et al. [13] developed the first FPRAS for #DNF, which we refer to as KL Counter. They defined a new universe $U' = \{(\mathbf{x}, \phi^{C_i}) \mid \mathbf{x} \models \phi^{C_i}\}$, and the corresponding solution space R'_ϕ as $R'_\phi = \{(\mathbf{x}, \phi^{C_i}) \mid \mathbf{x} \models \phi^{C_i} \text{ and } \forall j < i, \mathbf{x} \not\models \phi^{C_j}\}$ for a fixed ordering of the cubes. They showed that $|R_\phi| = |R'_\phi|$ and that the ratio $|U'|/|R'_\phi| \leq m$ and is therefore polynomially bounded. Their estimator is a 0/1 estimator and hence can be used with either the V-MC framework as in the original paper, or also with the DKLR-MC framework. The worst case complexity of their algorithm is $\mathcal{O}(m^2 n \log(2/\delta)/\varepsilon^2)$.

2.3.2 KLM Counter

Karp et al. [9] improved the running time of the KL Counter in their algorithm which we refer to as the KLM Counter. They defined 'coverage' of an assignment \mathbf{x} in U' as $\text{cov}(\mathbf{x}) = \{j \mid \mathbf{x} \models \phi^{C_j}\}$. The first key insight is that $|R'_\phi| = \sum_{(\mathbf{x}, \phi^{C_i}) \in U'} \frac{1}{|\text{cov}(\mathbf{x})|}$. The second insight was to define an estimator for $1/|\text{cov}(\mathbf{x})|$ using the geometric distribution. The geometric distribution has support in $(0, \infty)$ and hence the corresponding estimator cannot be used with the DKLR-MC framework. However, the worst case complexity of KLM Counter is $\mathcal{O}(mn \log(2/\delta)/\varepsilon^2)$ which is better than that of KL Counter.

2.3.3 Vazirani Counter

An easy way to adapt the KLM Counter for use with the DKLR-MC framework is to calculate $|\text{cov}(\mathbf{x})|$ exactly by iterating over all cubes instead of using the geometric estimator, since $1/|\text{cov}(\mathbf{x})|$ lies in $(0, 1]$. Although this adaptation of KLM Counter is straightforward, we refer to it as the Vazirani Counter since this variant was first

described by Vazirani [24]. Vazirani Counter has been used as a baseline for experimental evaluation of other algorithms for #DNF [25]. Its advantage is that its variance is smaller than that of the KL Counter and so DKLR-MC requires fewer samples to achieve the same error bounds. However, the time for generating a sample can be considerably more since all cubes have to be checked.

2.4 Matrix Notation

We use x, y, z, \dots to denote scalar variables. We use subscripts x_1, x_2, \dots 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 $\mathbf{A}, \mathbf{B}, \dots$ to denote matrices, small boldface letters $\mathbf{u}, \mathbf{v}, \mathbf{w}, \dots$ to denote vectors. $\mathbf{A}^{[p \times q]}$ denotes a matrix of p rows and q columns, while $\mathbf{u}^{[q]}$ denotes a vector of length q . $\mathbf{0}^{[q]}$ and $\mathbf{1}^{[q]}$ are the all 0s and all 1s vectors of length n , respectively. We omit the dimensions when clear from context. $\mathbf{x}[i]$ denotes the i^{th} element of \mathbf{x} , while $\mathbf{A}[i, j]$ denotes the element in the i^{th} row and j^{th} column of \mathbf{A} . $\mathbf{A}[r_1 : r_2, c_1 : c_2]$ denotes the sub-matrix of \mathbf{A} between rows r_1 and r_2 excluding r_2 and columns c_1 and c_2 excluding c_2 . Similarly $\mathbf{v}[i : j]$ denotes the sub-vector of \mathbf{v} between index i and index j excluding j . The i^{th} row of \mathbf{A} is denoted $\mathbf{A}[i, :]$ and j^{th} column as $\mathbf{A}[:, j]$. The $p \times (q_1 + q_2)$ matrix formed by concatenating rows of matrices $\mathbf{A}^{[p \times q_1]}$ and $\mathbf{B}^{[p \times q_2]}$ is written in block notation as $[\mathbf{A} \mid \mathbf{B}]$, while $[\frac{\mathbf{A}}{\mathbf{B}}]$ represents concatenation of columns. Similarly the $(q_1 + q_2)$ -length concatenation of vectors $\mathbf{v}^{[q_1]}$ and $\mathbf{w}^{[q_2]}$ is $[\mathbf{v} \mid \mathbf{w}]$. The dot product between matrix \mathbf{A} and vector \mathbf{x} is written as $\mathbf{A} \cdot \mathbf{x}$. The vector formed by element-wise XOR of vectors \mathbf{v} and \mathbf{w} is denoted $\mathbf{v} \oplus \mathbf{w}$.

2.5 Hash Functions

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

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

Of particular interest is the random XOR family of hash functions, which is defined as $H_{XOR}(\mathbf{q}, \mathbf{p}) = \{\mathbf{A}\cdot\mathbf{x} \oplus \mathbf{b} \mid \mathbf{A}[i, j] \in \{0, 1\} \text{ and } \mathbf{b}[i] \in \{0, 1\} ; 0 \leq i < \mathbf{p}, 0 \leq j < \mathbf{q}\}$. Selecting $\mathbf{A}[i, j]$ s and $\mathbf{b}[i]$ s randomly from $\{0, 1\}$ is equivalent to drawing uniformly at random from this family. A pair \mathbf{A} and \mathbf{b} now defines a hash function $h_{\mathbf{A}, \mathbf{b}}$ as follows: $h_{\mathbf{A}, \mathbf{b}}(\mathbf{x}) = \mathbf{A}\cdot\mathbf{x} \oplus \mathbf{b}$. This family was shown to be 2-universal in [26]. For a hash function $h \in H_{XOR}(\mathbf{q}, \mathbf{p})$, we have that $h(\mathbf{x}) = \mathbf{y}$ is a system of linear equations modulo 2: $\mathbf{A}\cdot\mathbf{x} \oplus \mathbf{b} = \mathbf{y}$. From another perspective, it can be viewed as a boolean formula $\psi = \bigwedge_{i=1}^{\mathbf{p}} (\bigoplus_{j=1}^{\mathbf{q}} (\mathbf{A}[i, j] \wedge \mathbf{x}[j])) \oplus \mathbf{b}[i] = \mathbf{y}[i]$. The solutions to this formula are exactly the elements of the set $h^{-1}(\mathbf{y})$.

2.6 Gaussian Elimination

Solving a system of linear equations over \mathbf{q} variables and \mathbf{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., [27]) 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 $\mathbf{A}\mathbf{x} \oplus \mathbf{b} = \mathbf{y}$, if the row-rank of the augmented matrix is same as row-rank of \mathbf{A} , then the system is consistent and the number of solutions is $2^{\mathfrak{q} - \text{rowrank}(\mathbf{A})}$ where \mathfrak{q} is the number of variables in the system of equations. Moreover, if \mathbf{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 $f = \mathfrak{q} - \text{rowrank}(\mathbf{A})$. 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 $\mathcal{O}(\mathfrak{q}^2)$ time. Thus we can enumerate all 2^f satisfying assignments to a system of linear equations $\mathbf{A}\mathbf{x} \oplus \mathbf{b} = \mathbf{y}$ if \mathbf{A} in Reduced Row-Echelon form.

2.7 Gray Codes

A Gray code [28] 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 [29].

Chapter 3

Related Work

Propositional model counting has been of theoretical as well as practical interest over the years [8, 13, 10, 21]. Early investigations showed that both $\#CNF$ and $\#DNF$ are $\#P$ -complete [5]. 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 [13], which provided a Monte Carlo-based FPRAS for $\#DNF$. The proposed FPRAS was improved by follow-up work of Karp, Luby and Madras [9] and Dagum et al. [12], achieving the best known complexity of $\mathcal{O}(mn \log(1/\delta)/\varepsilon^2)$. 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 [11, 19]. No practical algorithms materialized from 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 [17, 30]. 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 [16], or (ii) directly

obtain ε guarantees [14, 15]. The first technique when applied to DNF formulas is not an FPRAS. In contrast, Chakraborty, Meel and Vardi [15] 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 [14] 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 `hiThresh` + 1 solutions. Following the terminology of [14], we refer to the above described procedure as **BSAT** (bounded SAT). To determine the number of cells, **ApproxMC** performs a search that requires $\mathcal{O}(\log n)$ 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 $\mathcal{O}(\log n \log(\frac{1}{\delta}))$.

3.1.1 FPRAS for #DNF

The key insight of Chakraborty et al. [15] is that the BSAT procedure can be done in polynomial time when the input formula to ApproxMC is in DNF. The underlying structure of ApproxMC for DNF is the same as ApproxMC for CNF, except for the BSAT procedure. ApproxMC2 (algorithm 2) makes $t = O(\log(1/\delta))$ calls to ApproxMC2Core in lines 6-10 and returns the median of the invocations as the final result in line 12.

ApproxMC2Core (algorithm 3) makes a call to LogSATSearch (algorithm 4) which performs a binary / galloping search to find the correct number p of hashes. It then invokes BSAT with p of hash constraints and returns the count so obtained.

BSAT (algorithm 5) receives as input the formula and p hash constraints. In line 2 a matrix \mathbf{A}' in Reduced Row Echelon form is obtained by performing Gaussian Elimination over the matrix formed by concatenating a cube with the input hash matrix. If the number of solutions is greater than `hiThresh`, then it returns. Otherwise, solutions to \mathbf{A}' are enumerated in lines 6-8. Lines 9-10 ensure that at most `mhiThresh` solutions are enumerated in an invocation of BSAT. 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. To start, 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 $\mathcal{O}(mn^3 + mn^2/\varepsilon^2)$.*

Proof. When the input formula ϕ to ApproxMC2 is in DNF, BSAT is invoked with a formula of the form $\phi \wedge \psi$ where ψ is a conjunction of XOR constraints. For each cube

ϕ^{C_i} , BSAT proceeds by performing Gaussian Elimination (line 3) on $\phi^{C_i} \wedge \psi$. Since the number of XOR constraints can be $\mathcal{O}(n)$, Gaussian elimination can take $\mathcal{O}(n^3)$ time resulting in a cumulative complexity of $\mathcal{O}(mn^3)$ for all cubes. At most $\text{hiThresh} = \mathcal{O}(1/\varepsilon^2)$ solutions to each $\phi^{C_i} \wedge \psi$ may have to be enumerated and each enumeration requires $\mathcal{O}(n^2)$ time. Therefore the complexity of enumeration is $\mathcal{O}(mn^2/\varepsilon^2)$. Thus the BSAT runs in $\mathcal{O}(mn^3 + mn^2/\varepsilon^2)$ time in the worst case. \square

We can now complete the complexity analysis:

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

Proof. ApproxMC2 makes $\mathcal{O}(\log n \log(1/\delta))$ calls to BSAT. Substituting the complexity of BSAT from lemma 1, we get $\mathcal{O}((mn^3 + mn^2/\varepsilon^2) \log n \log(1/\delta))$ complexity for ApproxMC2. \square

Algorithm 2 $\text{ApproxMC2}(\phi, \varepsilon, \delta)$

```

1: hiThresh  $\leftarrow 1 + 9.84 \left(1 + \frac{\varepsilon}{1+\varepsilon}\right) \left(1 + \frac{1}{\varepsilon}\right)^2$ ;
2:  $Y \leftarrow \text{BSAT}(\phi, \text{null}, \text{null}, \text{null}, \text{hiThresh}, 0, n)$ ;
3: if ( $Y < \text{hiThresh}$ ) then return  $Y$ ;
4:  $t \leftarrow \lceil 17 \log_2(3/\delta) \rceil$ ;
5: cellCount  $\leftarrow 2$ ;  $C \leftarrow \text{emptyList}$ ; iter  $\leftarrow 0$ ;
6: repeat
7:   iter  $\leftarrow \text{iter} + 1$ ;
8:   (cellCount, solCount)  $\leftarrow \text{ApproxMC2Core}(\phi, \text{hiThresh}, \text{cellCount})$ ;
9:   if (cellCount  $\neq \perp$ ) then AddToList( $C, \text{solCount} \times \text{cellCount}$ );
   (iter  $< t$ );
10: finalEstimate  $\leftarrow \text{FindMedian}(C)$ ;
11: return finalEstimate

```

Algorithm 3 $\text{ApproxMC2Core}(\phi, \text{hiThresh}, \text{prevCellCount})$

```

1: Choose  $\mathbf{A}$  at random from  $H_{xor}(n, n-1)$ ;
2: Choose  $\mathbf{b}, \mathbf{y}$  at random from  $\{0, 1\}^{n-1}$ ;
3:  $Y \leftarrow \text{BSAT}(\phi, \mathbf{A}, \mathbf{b}, \mathbf{y}, \text{hiThresh}, n-1, n)$ ;
4: if ( $Y \geq \text{hiThresh}$ ) then return ( $\perp, \perp$ );
5: mprev  $\leftarrow \log_2 \text{prevCellCount}$ ;
6:  $p \leftarrow \text{LogSATSearch}(\phi, \mathbf{A}, \mathbf{b}, \mathbf{y}, \text{hiThresh}, \text{mprev})$ ;
7: solCount  $\leftarrow \text{BSAT}(\phi, \mathbf{A}, \mathbf{b}, \mathbf{y}, \text{hiThresh}, p, n)$ ;
8: return ( $2^p, \text{solCount}$ );

```

Algorithm 4 LogSATSearch($\phi, \mathbf{A}, \mathbf{b}, \mathbf{y}, \text{hiThresh}, \text{mprev}$)

```

1: lowerFib  $\leftarrow$  0; upperFib  $\leftarrow$  n - 1; p  $\leftarrow$  mprev;
2: FailRecord[0]  $\leftarrow$  1; FailRecord[n - 1]  $\leftarrow$  0;
3: FailRecord[i]  $\leftarrow$   $\perp$  for all  $i$  other than 0 and n - 1;
4: while true do
5:    $Y \leftarrow$  BSAT( $\phi, \mathbf{A}, \mathbf{b}, \mathbf{y}, \text{hiThresh}, p, n$ );
6:   if ( $|Y| \geq \text{hiThresh}$ ) then
7:     if (FailRecord[p + 1] = 0) then return p + 1;
8:     FailRecord[i]  $\leftarrow$  1 for all  $i \in \{1, \dots, p\}$ ;
9:     lowerFib  $\leftarrow$  p;
10:    if ( $|p - \text{mprev}| < 3$ ) then p  $\leftarrow$  p + 1;
11:    else if ( $2 \cdot p < |S|$ ) then p  $\leftarrow$  2 · p;
12:    else p  $\leftarrow$  (upperFib + p)/2;
13:  else
14:    if (FailRecord[p - 1] = 1) then return p;
15:    FailRecord[i]  $\leftarrow$  0 for all  $i \in \{p, \dots, n\}$ ;
16:    upperFib  $\leftarrow$  p;
17:    if ( $|p - \text{mprev}| < 3$ ) then p  $\leftarrow$  p - 1;
18:    else p  $\leftarrow$  (p + lowerFib)/2;

```

Algorithm 5 BSAT($\phi, \mathbf{A}, \mathbf{b}, \mathbf{y}, \text{hiThresh}, p, q$)

```

1:  $Sols \leftarrow \emptyset$ 
2: for ( $j = 0; j < m; j++$ ) do
3:    $\mathbf{A}' \leftarrow \text{GaussElim}(\mathbf{A} \wedge \phi^{C_j})$ 
4:   if  $q - \text{rank}(\mathbf{A}') \geq \log \text{hiThresh}$  then return  $\text{hiThresh}$ 
5:   for  $\mathbf{v} \in \{\mathbf{0}^{[q - \text{rank}(\mathbf{A}')]}, \dots, \mathbf{1}^{[q - \text{rank}(\mathbf{A}')]}\}$  do
6:      $\mathbf{A}'' \leftarrow \mathbf{A}'[0 : q - \text{rank}(\mathbf{A}'), q - \text{rank}(\mathbf{A}') : q]$ 
7:      $\mathbf{u} \leftarrow \mathbf{A}'' \cdot \mathbf{v} \oplus \mathbf{b} \oplus \mathbf{y}$ 
8:      $Sols \leftarrow Sols \cup \{\mathbf{u} : \mathbf{v}\}$ 
9:   if  $|Sols| \geq \text{hiThresh}$  then
10:     break
11: return  $|Sols|$ 

```

Chapter 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 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 $H_{REX}(\mathbf{q}, \mathbf{p}) = \{\mathbf{A}\cdot\mathbf{x} \oplus \mathbf{b} \mid \mathbf{A}^{[p \times q]} = [\mathbf{I}^{[p \times p]} \ : \ \mathbf{D}^{[p \times (q-p)]]}\}$ where \mathbf{I} is the identity matrix, \mathbf{D} and \mathbf{b} are random 0/1 matrix and vector respectively. In particular, we ensure that for every $\mathbf{D}[i, j]$ and $\mathbf{b}[i]$ we have

$\Pr[\mathbf{D}[i, j] = 1] = \Pr[\mathbf{D}[i, j] = 0] = 0.5$ and also $\Pr[\mathbf{b}[i] = 1] = \Pr[\mathbf{b}[i] = 0] = 0.5$. Note that \mathbf{D} and \mathbf{b} completely define a hash function from H_{REX} . The following theorem establishes the desired properties of universality for H_{REX} .

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

Proof. Let h be a random hash function from $H_{REX}(\mathbf{q}, \mathbf{p})$ with $\mathbf{A}^{[\mathbf{p} \times \mathbf{q}]}$ as its matrix. Let $\mathbf{x}_1, \mathbf{x}_2$ be any two assignments such that $\mathbf{x}_1 \neq \mathbf{x}_2$. To prove 2-universality of H_{REX} , we need to show that for all \mathbf{x} and \mathbf{y} :

$$\Pr[h(\mathbf{x}_1) = h(\mathbf{x}_2)] \leq \frac{1}{2^{\mathbf{p}}} \quad (4.1)$$

$$\Pr[h(\mathbf{x}) = \mathbf{y}] = \frac{1}{2^{\mathbf{p}}} \quad (4.2)$$

Equation 4.2 follows from the random choice of \mathbf{b} . In particular, for every \mathbf{x} and \mathbf{y} for a chosen \mathbf{A} , there is a unique \mathbf{b} such that $\mathbf{A} \cdot \mathbf{x} \oplus \mathbf{b} = \mathbf{y}$. Since there are $2^{\mathbf{p}}$ possible choices for \mathbf{b} , we have $\Pr[\mathbf{A} \cdot \mathbf{x} \oplus \mathbf{b} = \mathbf{y}] = \frac{1}{2^{\mathbf{p}}}$.

Let $\mathbf{x} = \mathbf{x}_1 \oplus \mathbf{x}_2$. Note that $\mathbf{x} \neq \mathbf{0}^{[\mathbf{q}]}$ since $\mathbf{x}_1 \neq \mathbf{x}_2$. We prove that $\Pr[h(\mathbf{x}) = \mathbf{0}^{[\mathbf{q}]}] \leq 1/2^{\mathbf{p}}$ which is equivalent to 4.1. This is the same as showing $\Pr[\mathbf{A} \cdot \mathbf{x} = \mathbf{0}^{[\mathbf{q}]}] \leq 1/2^{\mathbf{p}}$. We can write $\mathbf{A} = [\mathbf{I}^{[\mathbf{p} \times \mathbf{p}]} \quad : \quad \mathbf{D}^{[\mathbf{p} \times (\mathbf{q} - \mathbf{p})]}]$ and \mathbf{x} as $\mathbf{x} = [\mathbf{u}^{[\mathbf{p}]} \quad : \quad \mathbf{v}^{[(\mathbf{q} - \mathbf{p})]}]$ in block notation. Then we have $\mathbf{A} \cdot \mathbf{x} = \mathbf{I} \cdot \mathbf{u} \oplus \mathbf{D} \cdot \mathbf{v}$. Since $\mathbf{x} \neq \mathbf{0}$, either $\mathbf{u} \neq \mathbf{0}$ or $\mathbf{v} \neq \mathbf{0}$ leading to the following three cases:

Case 1: If $\mathbf{u} \neq \mathbf{0}^{[\mathbf{p}]}$ and $\mathbf{v} = \mathbf{0}^{[\mathbf{q} - \mathbf{p}]}$, we get $\mathbf{I} \cdot \mathbf{u} \neq \mathbf{0}^{[\mathbf{p}]}$ and $\mathbf{D} \cdot \mathbf{v} = \mathbf{0}^{[\mathbf{p}]}$. Therefore

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{I} \cdot \mathbf{u} \oplus \mathbf{D} \cdot \mathbf{v} \neq \mathbf{0}^{[\mathbf{p}]}.$$

Case 2: If $\mathbf{u} = \mathbf{0}^{[\mathbf{p}]}$ and $\mathbf{v} \neq \mathbf{0}^{[\mathbf{q} - \mathbf{p}]}$, we get $\mathbf{I} \cdot \mathbf{u} = \mathbf{0}^{[\mathbf{p}]}$. From the proof of Theorem 1 in [31] we have $\Pr[\mathbf{D} \cdot \mathbf{v} = \mathbf{0}^{[\mathbf{p}]}] = \frac{1}{2^{\mathbf{p}}}$. Therefore $\Pr[\mathbf{A} \cdot \mathbf{x} = \mathbf{0}^{[\mathbf{p}]}] = \frac{1}{2^{\mathbf{p}}}$

Case 3: If $\mathbf{u} \neq \mathbf{0}^{[\mathbf{p}]}$ and $\mathbf{v} \neq \mathbf{0}^{[\mathbf{q} - \mathbf{p}]}$, we get $\mathbf{I} \cdot \mathbf{u} \neq \mathbf{0}^{[\mathbf{p}]}$ and $\Pr[\mathbf{D} \cdot \mathbf{v} = \mathbf{0}^{[\mathbf{p}]}] = \frac{1}{2^{\mathbf{p}}}$.

$$\text{Therefore } \Pr[\mathbf{A} \cdot \mathbf{x} = \mathbf{0}^{[\mathbf{p}]}] = \frac{1}{2^{\mathbf{p}}}$$

Algorithm 6 enumNextREX($\mathbf{D}, \mathbf{u}, \mathbf{v}, k$)

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

□

The naive way of enumerating satisfying assignments for a given $\mathbf{D}^{[p \times (q-p)]}$, $\mathbf{b}^{[p]}$, and $\mathbf{y}^{[p]}$ is to iterate over all 2^f assignments to the free variables in sequence starting from $\mathbf{0}^{[f]}$ to $\mathbf{1}^{[f]}$, where $f = (q - p)$. For each assignment $\mathbf{v}^{[f]}$ to the free variables, the corresponding assignment to the dependent variables $\mathbf{u}^{[q-f]}$ can be calculated as $\mathbf{u} = (\mathbf{D} \cdot \mathbf{v}) \oplus \mathbf{b} \oplus \mathbf{y}$, which requires $O(pq)$ 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 6). The algorithm takes the hash matrix \mathbf{D} , an assignment to the free variables \mathbf{v} , and an assignment to the dependent variables \mathbf{u} as inputs, and outputs the next free-variable assignment \mathbf{v}' in the Gray sequence and the corresponding assignment \mathbf{u}' to the dependent variables. k represents the position of the bit that is changed between \mathbf{v} and \mathbf{v}' . Thus enumNextREX constructs a satisfying assignment to a Row-Echelon XOR hash function in each invocation in $\mathcal{O}(q)$ time.

4.2 Symbolic Hashing

For DNF formulas, R_ϕ can be exponentially sparse compared to U , which is undesirable*. 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 $U' = \{(\mathbf{x}, \phi^{C_i}) \mid \mathbf{x} \models \phi^{C_i}\}$. Thus, corresponding to each $\mathbf{x} \models \phi$ that satisfies cubes $\phi^{C_{i_1}}, \dots, \phi^{C_{i_x}}$ in ϕ , we have the states $\{(\mathbf{x}, \phi^{C_{i_1}}), (\mathbf{x}, \phi^{C_{i_2}}), \dots, (\mathbf{x}, \phi^{C_{i_x}})\}$ in U' . Next, the solution space is defined as $R'_\phi = \{(\mathbf{x}, \phi^{C_i}) \mid \mathbf{x} \models \phi^{C_i} \text{ and } \forall j < i, \mathbf{x} \not\models \phi^{C_j}\}$ 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 [13].

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 $H_{REX}(\mathbf{q}, \mathbf{p})$ over $\mathbf{q} = \mathbf{n} - \mathbf{w} + \log \mathbf{m}$ variables instead of \mathbf{n} variables. Note that the variables of a satisfying assignment $\mathbf{z} \in \{0, 1\}^{\mathbf{q}}$ to the hash function are now different from the variables to a satisfying assignment $\mathbf{x} \in \{0, 1\}^{\mathbf{n}}$ of the input formula ϕ . We interpret \mathbf{z} as follows: the last $\log \mathbf{m}$ bits of \mathbf{z} are converted to a number i such that $1 \leq i \leq \mathbf{m}$. Now ϕ^{C_i} corresponds to a partial assignment of $\text{width}[\phi^{C_i}]$ variables in that cube. For simplicity, we assume that each cube is of the same width \mathbf{w} .[†] The remaining $\mathbf{n} - \mathbf{w}$ bits of \mathbf{z} are interpreted to be the assignment to the $\mathbf{n} - \mathbf{w}$ variables not in ϕ^{C_i} giving a complete assignment \mathbf{x} . Thus we get a pair (\mathbf{x}, ϕ^{C_i}) from \mathbf{z} such that $\mathbf{x} \models \phi^{C_i}$. For a fixed ordering of variables and cubes we see that there is a bijection between (\mathbf{x}, ϕ^{C_i})

*Number of steps of ApproxMC2 search procedure increases with sparsity

[†]We can handle non-uniform width cubes by sampling ϕ^{C_i} with probability $\frac{2^{\mathbf{n} - \text{width}[\phi^{C_i}]}}{\sum_{j=1}^{\mathbf{m}} 2^{\mathbf{n} - \text{width}[\phi^{C_j}]}}$ instead of uniformly

and \mathbf{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 (\mathbf{x}, ϕ^{C_i}) generated using symbolic hash function as described above: if $(\mathbf{x}, \phi^{C_i}) \in R'_\phi$. Such a check would require iteration over cubes ϕ^{C_j} for $1 \leq j \leq (i - 1)$ and returning no if $\mathbf{x} \models \phi^{C_j}$ for some j and yes otherwise. This would result in procedure with $O(mn)$ 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 \mathbf{x} as $cov(\mathbf{x}) = \{j | \mathbf{x} \models \phi^{C_j}\}$. Note that $\sum_{(\mathbf{x}, \phi^{C_i}) \in U'} \frac{1}{|cov(\mathbf{x})|} = |R_\phi|$.

We define a random variable $\mathbf{c}_\mathbf{x}$ as the number of steps taken to uniformly and independently sample from $\{1, 2, \dots, m\}$, a number j such that $\mathbf{x} \models \phi^{C_j}$. For a randomly chosen j , the probability $\Pr[\mathbf{x} \models \phi^{C_j}] = |cov(\mathbf{x})|/m$, which follows the Bernoulli distribution. The random variable $\mathbf{c}_\mathbf{x}$ is the number of Bernoulli trials for the first success, which follows the geometric distribution. Therefore, $\mathbb{E}[\mathbf{c}_\mathbf{x}] = m/|cov(\mathbf{x})|$, and $\mathbb{E}[\mathbf{c}_\mathbf{x}/m] = 1/|cov(\mathbf{x})|$. The estimator $\mathbf{c}_\mathbf{x}/m$ has been previously employed by Karp et al. [9]. 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 $\Omega_\mathbf{y} = \sum_{(\mathbf{x}, \phi^{C_i}) \in h^{-1}(\mathbf{y})} \mathbf{c}_\mathbf{x}/m$.

4.4 The Full Algorithm

We now incorporate the above techniques into ApproxMC2 and call the revised algorithm SymbolicDNFAproxMC, which is presented as Algorithm 7. First, note that expression for hiThresh is twice that for ApproxMC2. Then, in line 4, a matrix $\hat{\mathbf{D}}$

Algorithm 7 SymbolicDNFAproxMC($\phi, \varepsilon, \delta$)

```

1: hiThresh  $\leftarrow 2 * (1 + 9.84 (1 + \frac{\varepsilon}{1+\varepsilon}) (1 + \frac{1}{\varepsilon})^2)$ ;
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)  $\leftarrow$  SymbolicDNFAproxMCCore( $\phi$ , hiThresh);
7:   if (cellCount  $\neq \perp$ ) then AddToList(EstimateList, solCount  $\times$  cellCount);
   (iter <  $t$ );
8: finalEstimate  $\leftarrow$  FindMedian(EstimateList);
9: return finalEstimate

```

and vectors $\hat{\mathbf{b}}$ and $\hat{\mathbf{y}}$ are obtained, which are employed to construct an appropriate hash function and cell during the search procedure of SymbolicDNFAproxMCCore. SymbolicDNFAproxMC makes $t = O(\log(1/\delta))$ calls to SymbolicDNFAproxMCCore (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 SymbolicDNFAproxMCCore, 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^{n-w} . Therefore, instead of starting with 1 hash constraint, we can safely start with $sI = n - w - \log \text{hiThresh}$ constraints (lines 3-4). Thereafter, SymbolicDNFAproxMCCore calls LogSATSearch in line 5 to find the right number \mathbf{p} of constraints. The cell count with \mathbf{p} constraints is calculated in line 6 and the estimate ($2^{\mathbf{p}}$, solCount) is returned in line 7.

SampleBase algorithm constructs the base matrix $\hat{\mathbf{D}}$ and base vectors $\hat{\mathbf{b}}$ and $\hat{\mathbf{y}}$ required for sampling from H_{REX} family. \mathbf{G} is a random matrix of dimension $sI \times$

Algorithm 8 SymbolicDNFAproxMCCore(ϕ , hiThresh)

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

Algorithm 9 SampleBase(q, sI)

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

$(q - sI)$ and \mathbf{E} is a random upper triangular matrix of dimension $(q - sI - 1) \times (q - sI)$ with all diagonal elements 1. In line 3, $\hat{\mathbf{D}}$ is constructed as the vertical concatenation $\begin{bmatrix} \mathbf{G} \\ \mathbf{E} \end{bmatrix}$.

LogSATSearch (algorithm 10) 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^{n-w} and $m * 2^{n-w}$, we need to perform search for p between $n - w$ and $n - w + \log m$. Therefore, binary search can take at most $O(\log \log m)$ steps to find correct p .

Symbolic Hashing is implemented in Algorithm 11 (BSAT). In line 2, we obtain

Algorithm 10 $\text{LogSATSearch}(\phi, \hat{\mathbf{D}}, \hat{\mathbf{b}}, \hat{\mathbf{y}}, \text{hiThresh}, \text{low}, \text{hi})$

```

1: lowerFib  $\leftarrow$  0; upperFib  $\leftarrow$   $hi - low + 1$ ;  $p \leftarrow low$ 
2: FailRecord[0]  $\leftarrow$  1; FailRecord[ $hi - low + 1$ ]  $\leftarrow$  0;
3: FailRecord[ $i$ ]  $\leftarrow$   $\perp$  for all  $i$  other than 0 and  $hi - low + 1$ ;
4: while true do
5:    $C_{BSAT} \leftarrow$  BSAT( $\phi, \hat{\mathbf{D}}, \hat{\mathbf{b}}, \hat{\mathbf{y}}, \text{hiThresh}, p, q, sI$ );
6:   if ( $C_{BSAT} \geq \text{hiThresh}$ ) then
7:     if (FailRecord[ $p + 1 - low + 1$ ] = 0) then return  $p + 1$ ;
8:     FailRecord[ $i$ ]  $\leftarrow$  1 for all  $i \in \{1, \dots, p - low + 1\}$ ;
9:     lowerFib  $\leftarrow$   $p - low + 1$ ;
10:     $p \leftarrow (upperFib + lowerFib)/2$ ;
11:  else
12:    if (FailRecord[ $p - 1 - low + 1$ ] = 1) then return  $p$ ;
13:    FailRecord[ $i$ ]  $\leftarrow$  0 for all  $i \in \{p, \dots, hi - low + 1\}$ ;
14:    upperFib  $\leftarrow$   $p - low + 1$ ;
15:     $p \leftarrow (upperFib + lowerFib)/2$ ;

```

a hash function from $H_{REX}(q, p)$ over $q = n - w + \log m$ variables by calling `Extract`. We assume access to a procedure `nextGrayBit` in line 10 that returns the position of the bit that is flipped between two consecutive assignments. A satisfying assignment \mathbf{z} to the hash function is constructed in line 6. \mathbf{z} is interpreted to generate a pair (\mathbf{x}, ϕ^{Ci}) in line 7 which is checked for satisfiability in line 8. The final cell count is returned in line 12.

In `CheckSAT` (algorithm 12), we implement the stochastic cell counting procedure. The key idea is to sample cubes uniformly at random from $\{1, 2, \dots, m\}$ till a cube

Algorithm 11 BSAT($\phi, \hat{\mathbf{D}}, \hat{\mathbf{b}}, \hat{\mathbf{y}}, \text{hiThresh}, \mathbf{p}, \mathbf{q}, sI$)

```

1:  $count \leftarrow 0$ ;
2:  $\mathbf{D}, \mathbf{b}, \mathbf{y} \leftarrow \text{Extract}(\hat{\mathbf{D}}, \hat{\mathbf{b}}, \hat{\mathbf{y}}, \mathbf{p}, \mathbf{q}, sI)$ ;
3:  $\mathbf{u}^{\mathbf{p}} \leftarrow \mathbf{b} \oplus \mathbf{y}$ ;
4:  $\mathbf{v}^{\mathbf{q}-\mathbf{p}} \leftarrow \mathbf{0}^{\mathbf{q}-\mathbf{p}}$ ;
5: for ( $j = 0$ ;  $j < 2^{\mathbf{q}-\mathbf{p}}$ ;  $j++$ ) do
6:    $\mathbf{z} \leftarrow [\mathbf{u} : \mathbf{v}]$ ;
7:    $(\mathbf{x}, \phi^{C^j}) = \text{interpret}(\mathbf{z})$ ;
8:    $count = count + \text{CheckSAT}(\mathbf{x}, \phi^{C^j}, count, \text{hiThresh})$ ;
9:   if  $count \geq \text{hiThresh}$  then return  $hiThresh$ ;
10:   $k \leftarrow \text{nextGrayBit}(\mathbf{q} - \mathbf{p}, j)$ ;
11:   $(\mathbf{u}, \mathbf{v}) \leftarrow \text{enumNextREX}(\mathbf{D}, \mathbf{u}, \mathbf{v}, k)$ ;
12: return  $count$ 

```

ϕ^{C^j} is found such that $\mathbf{x} \models \phi^{C^j}$ (lines 2-5). The number of cubes sampled $c_{\mathbf{x}}$ divided by total number of cubes m is the estimate returned (line 6).

Procedure `LogSATSearch` in `SymbolicDNFAproxMC` is based upon `LogSATSearch` in `ApproxMC2` [15]. As noted in the analysis of `ApproxMC2`, such a logarithmic search procedure requires that the solution space for a hash function with $\mathbf{p} + 1$ hash constraints is a subset of the solution space with \mathbf{p} hash constraints. Furthermore, we want to preserve Row-Echelon nature of the resulting hash constraints. To this end, we first construct $\mathbf{D}^{[\mathbf{q} \times (\mathbf{q}-\mathbf{p})]}$ and $\mathbf{b}^{[\mathbf{q}-1]}$ as follows:

To seed the construction procedure, in `SampleBase` (algorithm 9) we first randomly sample a 0/1 vector $\hat{\mathbf{b}}$ of size $\mathbf{q} - 1$ which is the maximum number of hash constraints possible. We then construct a 0/1 matrix $\hat{\mathbf{D}}$ as follows: $\hat{\mathbf{D}}^{[(\mathbf{q}-1) \times (\mathbf{q}-sI)]} = \begin{bmatrix} \mathbf{G} \\ \mathbf{E} \end{bmatrix}$ where

Algorithm 12 CheckSAT($\mathbf{x}, \phi^{C^i}, count, hiThresh$)

```

1:  $\mathbf{c}_x \leftarrow 0$ ;
2: while  $count + \mathbf{c}_x/m < hiThresh$  do
3:   Uniformly sample  $j$  from  $\{1, 2, \dots, m\}$ ;
4:    $\mathbf{c}_x \leftarrow \mathbf{c}_x + 1$ ;
5:   if  $\mathbf{x} \models \phi^{C^j}$  then
6:     return  $\mathbf{c}_x/m$ ;
7: return  $\mathbf{c}_x/m$ 

```

matrix $\mathbf{G}^{[sI \times (q-sI)]}$ is a random 0/1 matrix with sI rows, and matrix $\mathbf{E}^{[(q-sI) \times (q-sI)]}$ is defined 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 13) takes $\hat{\mathbf{D}}, \hat{\mathbf{b}}$ and $\hat{\mathbf{y}}$ and a number \mathbf{p} as input and returns \mathbf{D}, \mathbf{b} and cell \mathbf{y} such that (\mathbf{D}, \mathbf{b}) represents a hash function from H_{REX} with \mathbf{p} constraints and \mathbf{y} represents a cell. A precondition for Extract is $sI \leq \mathbf{p} \leq q-1$. In lines 1 and 2, the first \mathbf{p} rows of $\hat{\mathbf{D}}$ and first \mathbf{p} elements of $\hat{\mathbf{b}}$ and $\hat{\mathbf{y}}$ are selected as $\hat{\mathbf{D}}', \mathbf{b}$ and \mathbf{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 $\mathbf{p} - sI$ rows of $\hat{\mathbf{D}}'$ are the first $\mathbf{p} - sI$ rows of matrix \mathbf{E} . Each

Algorithm 13 Extract($\hat{\mathbf{D}}, \hat{\mathbf{b}}, \hat{\mathbf{y}}, p, q, sI$)

```

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

```

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 $p - sI$ columns are the leading 1s in rows sI to p . Thus Extract ensures that for a given $\hat{\mathbf{D}}, \mathbf{b}$ and $\hat{\mathbf{y}}$, the solution space of $\mathbf{D}^{[p \times (q-p)]}, \mathbf{b}^{[p]}$ and $\mathbf{y}^{[p]}$ is a superset of solution space of $\mathbf{D}^{[(p+1) \times (q-p-1)]}, \mathbf{b}^{[p+1]}$ and $\mathbf{y}^{[p+1]}$ for all p .

Chapter 5

Analysis

In order to prove the correctness of SymbolicDNFAproxMC, we first state and prove the following helper lemma.

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

$$1. \Pr[|\Omega_{\mathbf{y}} - \mu_p| > \frac{\varepsilon}{(1+\varepsilon)}\mu_p] \leq \frac{2}{\frac{\varepsilon^2}{(1+\varepsilon)^2}\mu_p}$$

$$2. \Pr[\Omega_{\mathbf{y}} \leq \beta\mu_p] \leq \frac{2}{2+(1-\beta^2)\mu_p}$$

Proof. The *coverage* of an assignment is $\text{cov}(\mathbf{x}) = \{j | \mathbf{x} \models \phi^{Cj}\}$. We have $\Pr[\mathbf{x} \models \phi^{Cj}] = |\text{cov}(\mathbf{x})|/m$ when j is drawn uniformly at random from $\{1, 2, \dots, m\}$. Also, $\sum_{(\mathbf{x}, \phi^{C_i}) \in U'} \frac{1}{|\text{cov}(\mathbf{x})|} = |R_\phi|$, and $\mathbb{E}[\mathbf{c}_{\mathbf{x}}] = m/|\text{cov}(\mathbf{x})|$ and $\mathbb{E}[\mathbf{c}_{\mathbf{x}}^2] = \frac{2m^2}{|\text{cov}(\mathbf{x})|^2} - \frac{m}{|\text{cov}(\mathbf{x})|}$.

Let $\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}$ be a random variable such that $\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}} = \mathbf{c}_{\mathbf{x}}/m$ if $h(\mathbf{x}) = \mathbf{y}$ and $(\mathbf{x}, \phi^{C_i}) = \text{interpret}(\mathbf{z})$, and $\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}} = 0$ otherwise, where the number of constraints in h is p . Let $\eta = \Pr[h(\mathbf{x}) = \mathbf{y}] = 1/2^p$. Then $\mathbb{E}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}] = \eta * \mathbb{E}[\frac{\mathbf{c}_{\mathbf{x}}}{m}] = \frac{\eta}{|\text{cov}(\mathbf{x})|}$.

Now, $\mathbb{V}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}] = \mathbb{E}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}^2] - (\mathbb{E}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}])^2$. But $\mathbb{E}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}^2] = \eta * \mathbb{E}[(\mathbf{c}_{\mathbf{x}}/m)^2] = \eta(\frac{2}{|\text{cov}(\mathbf{x})|^2} - \frac{1}{m*|\text{cov}(\mathbf{x})|})$ and $(\mathbb{E}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}])^2 = \frac{\eta^2}{|\text{cov}(\mathbf{x})|^2}$. Substituting back, we get $\mathbb{V}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}] = \frac{2\eta - \eta^2}{|\text{cov}(\mathbf{x})|^2} - \frac{\eta}{m*|\text{cov}(\mathbf{x})|}$.

Define $\tau_{\mathbf{y}} = \sum_{(\mathbf{x}, \phi^{C_i}) \in U'} \gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}$. Clearly $\tau_{\mathbf{y}} = |\Omega_{\mathbf{y}}|$. We have $\mathbb{E}[\tau_{\mathbf{y}}] = \sum_{(\mathbf{x}, \phi^{C_i}) \in U'} \mathbb{E}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}] = \sum_{(\mathbf{x}, \phi^{C_i}) \in U'} \frac{\eta}{|\text{cov}(\mathbf{x})|} = \eta|R_\phi|$. Also since $\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}$ s are 2-universal, $\mathbb{V}[\tau_{\mathbf{y}}] \leq \sum_{(\mathbf{x}, \phi^{C_i}) \in U'} \mathbb{V}[\gamma_{(\mathbf{x}, \phi^{C_i}), \mathbf{y}}] = (2\eta - \eta^2) \sum_{(\mathbf{x}, \phi^{C_i}) \in U'} \frac{1}{|\text{cov}(\mathbf{x})|^2} -$

$\frac{\eta}{m} \sum_{(\mathbf{x}, \phi^{C^i}) \in U'} \frac{1}{|\text{cov}(\mathbf{x})|}$. But $\sum_{(\mathbf{x}, \phi^{C^i}) \in U'} \frac{1}{|\text{cov}(\mathbf{x})|^2} \leq |R_\phi|$. Therefore $\mathbf{V}[\tau_{\mathbf{y}}] \leq 2\eta|R_\phi|$ which implies $\mathbf{V}[\tau_{\mathbf{y}}] \leq 2\mathbf{E}[\tau_{\mathbf{y}}]$.

Applying Chebyshev's inequality we get $\Pr[|\Omega_{\mathbf{y}} - \mu_{\mathbf{p}}| \geq \frac{\varepsilon}{1+\varepsilon}\mu_{\mathbf{p}}] \leq \frac{\mathbf{V}[\tau_{\mathbf{y}}]}{\frac{\varepsilon^2}{(1+\varepsilon)^2}\mu_{\mathbf{p}}^2}$. Rearranging the terms and simplifying, we get the first part of the lemma.

Using Paley-Zygmund inequality, we get $\Pr[\Omega_{\mathbf{y}} \leq \beta\mu_{\mathbf{p}}] \leq 1 - \frac{(1-\beta)^2\mu_{\mathbf{p}}^2}{\mathbf{V}[\tau_{\mathbf{y}}] + (1-\beta^2)\mu_{\mathbf{p}}^2}$ from which we get the second part of the lemma. \square \square

The difference in lemma 4 and lemma 1 in [15] is that the probability bounds differ by a factor of 2. We account for this difference by making `hiThresh` in `SymbolicDNFAproxMC` twice the value of `hiThresh` in `ApproxMC2`. Therefore the rest of the proof of Theorem 7 is exactly the same as the proof of Theorem 4 of [15]. For completeness, we restate lemmas 2 and 3 from [15] below.

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

Lemma 5. *The following bounds hold:*

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

\square

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

Lemma 6. $\Pr[B] \leq 0.36$ □

Proof Sketch. $\Pr[B] \leq \Pr[\bigcup_{i \in [n]} (\overline{T_{i-1}} \cap T_i \cap (L_i \cup U_i))]$. We first note that $\forall i, |R_{F,h^i,\alpha^i}| \leq \text{hiThresh} \implies |R_{F,h^{i+1},\alpha^{i+1}}| \leq \text{hiThresh}$. Simplifying the expression for $\Pr[B]$ requires us to note the following: (i) $\forall i \leq m^* - 3, T_i \cap (L_i \cup U_i) = T_i$ and $T_i \subseteq T_{m^*-3}$, (ii) $\Pr[\bigcup_{i \in [m^*,n]} \overline{T_{i-1}} \cap T_i \cap (L_i \cup U_i)] \leq \Pr[\overline{T_{m^*-1}} \cap (L_{m^*} \cup U_{m^*})] \leq \Pr[L_{m^*} \cup U_{m^*}]$, (iii) for $i \in \{m^* - 2, m^* - 1\}$, since $\text{hiThresh} \leq \mu^i(1 + \frac{\varepsilon}{1+\varepsilon})$, we have $T_i \cap U_i = \emptyset$. Therefore, $\Pr[B] \leq \Pr[T_{m^*-3}] + \Pr[L_{m^*-2}] + \Pr[L_{m^*-1}] + \Pr[L_{m^*} \cup U_{m^*}]$. Using Lemmas 4, 5, we have $\Pr[B] \leq 1/63 + 1/11 + 1/21 + 1/5 \leq 0.36$ □

Theorem 7. *Let $\text{SymbolicDNFAproxMC}(\phi, \varepsilon, \delta)$ 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 $\text{SymbolicDNFAproxMC}$ boosts the probability of correctness of the count returned by $\text{SymbolicDNFAproxMCCore}$ to $1 - \delta$ by using median of $t = O(\log(1/\delta))$ calls.

Theorem 8. *$\text{SymbolicDNFAproxMC}$ runs in $\tilde{O}(mn \log(1/\delta)/\varepsilon^2)$ time.**

Proof. $\text{SymbolicDNFAproxMC}$ makes $\mathcal{O}(\log 1/\delta)$ calls to $\text{SymbolicDNFAproxMCCore}$. $\text{SymbolicDNFAproxMCCore}$ samples a hash matrix $\hat{\mathbf{D}}$ in $\mathcal{O}(n(\log m + \log(1/\varepsilon^2)))$ time and makes one call each to LogSATSearch and BSAT . LogSATSearch search in turn makes upto $\mathcal{O}(\log \log m)$ calls to BSAT . BSAT first reduces the matrix $\hat{\mathbf{D}}$ to \mathbf{D} in $\mathcal{O}(n(\log m + \log(1/\varepsilon^2))^2)$ time by calling Extract . Next, the check in line 9 of BSAT ensures that at most $m \times \text{hiThresh}$ calls are made to the check in line 5 of CheckSAT during one execution of BSAT . Every assignment is enumerated in $\mathcal{O}(n)$ and the check in line 5 of CheckSAT is performed in $\mathcal{O}(n)$ time. Therefore, the time complexity of

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

BSAT is $O(n \times m \times \text{hiThresh})$. Hence, the time complexity of each invocation of SymbolicDNFAproxMCCore is $\tilde{O}(mn/\varepsilon^2)$, which implies that the time complexity of SymbolicDNFAproxMC is $\tilde{O}(mn \log(1/\delta)/\varepsilon^2)$. \square

Chapter 6

Empirical Evaluation

In order to test the performance of ApproxMC on DNF formulas in practice, we built prototype implementations of ApproxMC as well as the Monte Carlo algorithms described in chapter 2. In this chapter we report on our experiments comparing these algorithms for #DNF.

6.1 Previous Experimental Work

DNF Counting finds application primarily in the areas of Probabilistic Databases and Network Reliability [2, 18, 13, 9]. Probabilistic Databases research, in particular, has actively focused on building and testing approximate DNF counters for answering probabilistic queries [25, 32]. Answering probabilistic queries actually reduces to the related problem of Weighted DNF Counting, where each literal is assigned a weight between 0 and 1 that is interpreted as the probability of the corresponding variable being true or false. Finding the probability of a tuple in the result of a probabilistic conjunctive query is equivalent to calculating the weighted count of a DNF formula[2]. We refer the reader to [33] for more on the connection between DNF counting and probabilistic query evaluation.

To the best of our knowledge there are no standard sets of benchmark formulas that are used to test algorithms for #DNF in the probabilistic databases community. Instead, a random data generator called TPC-H [34] which is an industry standard for

benchmarking databases in general, is used to populate a prototype database[25]. By running a probabilistic query in the database, the DNF formula is generated implicitly within the query engine and the desired counting algorithm is run on it.

While comparing their new algorithms to those by Karp et al., all previous works compared only against the Vazirani Counter with DKLR-MC. This algorithm was chosen as the baseline over the other variants, because the variance of the related estimator is the least, and DKLR-MC provably requires close to optimal number of samples. However, the running time of the algorithm depends both on the number of samples generated as well as the time it takes to generate one sample, and generating a sample using the Vazirani Counter is much more expensive than generating one using the KLM Counter. In fact, the worst case complexity of the Vazirani Counter is $\mathcal{O}(m^2n \log(2/\delta)/\varepsilon^2)$ while that of the KLM Counter is $\mathcal{O}(mn \log(2/\delta)/\varepsilon^2)$. Therefore, discarding all other Monte Carlo algorithms in favor of the Vazirani Counter with DKLR-MC may be ill-advised.

6.2 Algorithm Suite

We chose 2 hashing based variants of ApproxMC and 4 Monte Carlo algorithms for empirical evaluation.

Hashing based

There can be seven versions of ApproxMC for #DNF in theory depending on which of the three add-ons among Row Echelon Hash functions, Symbolic Hashing and Stochastic Cell Counting are selected. Of these, we use two versions for experimental evaluation:

1. the final algorithm described in this work which has all 3 add-ons which we will

refer to as `SymbolicDNFAproxMC` in this chapter.

2. the original `ApproxMC` for DNF [15] with the Row Echelon Hash function addition which we will refer to as `DNFAproxMC`.

The reason for these particular choices is that using Row Echelon Hash is unconditionally faster than plain XOR hash in both time and space since Gaussian Elimination is an $\mathcal{O}(n^3)$ operation. Furthermore, Stochastic Cell Counting can be applied only in the context of the transformed space of Symbolic Hashing, thereby leaving `SymbolicDNFAproxMC` and `DNFAproxMC` as the only valid choices.

Monte Carlo based

We chose 4 algorithms from the Karp et al. suite for experimental evaluation

1. KL Counter with DKLR-MC
2. KLM Counter
3. Vazirani Counter with DKLR-MC
4. Naive Counter (see section 2.3 of Chapter 2) with DKLR-MC

We use DKLR-MC instead of V-MC for the first 3 algorithms, since we assume to have no knowledge of the solution density of our benchmarks while running the algorithm, even though the solution density is taken into account while generating the benchmarks. DKLR-MC can thus be expected to perform better than V-MC because of no tight bound on the number of samples required. Furthermore, as mentioned in Chapter 2, the KLM Counter cannot be used with DKLR-MC framework.

6.3 Implementation Details

We implemented all algorithms in C++ and are available at https://gitlab.com/Shrotri/DNF_Counting. For the hashing-based algorithms, we used a library called M4RI that supports fast matrix operations over GF_2 [35]. M4RI uses SIMD instructions available in most modern processors to speed up computation, although it can work without these instructions as well. The performance of both SymbolicDNFAproxMC and DNFAproxMC, however, crucially depends on M4RI and availability of SIMD instructions. For DNFAproxMC, we made some improvements over the original algorithm in [15]. Besides using Row Echelon hash functions we also buffer and reuse solutions generated in a call to BSAT which sped up the algorithm further.

For the Monte Carlo algorithms, we adapted the implementation of DKLR-MC in MayBMS [36] to the unweighted case. We also modified their implementation to use only two sets of samples as outlined in the original paper instead of three, which further improved the running time. The KLM Counter algorithm was implemented from scratch.

6.4 Set Up

We ran the experiments on an Intel Core i7 6th Gen 2.6 GHz processor with 16GB RAM. We used GCC version 5.4 on 64 bit Ubuntu 16.04.

6.5 Benchmarks

One possible way to obtain benchmarks for unweighted $\#DNF$ would have been to extract DNF formulas from the query engines of databases populated by TPC-H

data and to simply disregard the weights associated with the literals. However, the width of cubes for formulas obtained this way is exactly the number of joins in the associated query, which is typically very small (2-15) and can be treated as a constant. Therefore, the count of such formulas is very close to 2^n and the Naive Counter with V-MC would take only a constant number of samples to estimate it. Moreover an epsilon approximation for a count extremely close to 2^n is meaningless. Lastly, future applications of unweighted DNF counting may have formulas with large width cubes. Hence a better approach would assume no information on the structure of benchmark formulas.

To obtain a comprehensive picture of the relative performance of the six candidate algorithms, we aimed to generate random formulas from the 'formula terrain' defined by the parameters n , m and w which is the width for all cubes. The number of samples needed for the 4 Monte Carlo algorithms also depends on the density of solutions in the solution space. Ergo while generating benchmarks we also take into account the parameters $\mu = \frac{|R_\phi|}{2^n}$ and $\mu' = \frac{|R'_\phi|}{U'}$ which represent the density of solutions in the original and transformed space respectively. Thus we consider the space of all formulas to be defined by the parameters n , m , w , μ and μ' . Note that these parameters are not independent; μ' is determined by the remaining parameters. However, considering them separately simplifies the presentation of results.

One way of generating random DNF formulas is to randomly sample w variables out of n , without replacement for each of the m cubes and to negate each one with probability 0.5. However, in our experiments we found that formulas generated this way invariably have μ and μ' extremely close to their respective upper bounds. Therefore, another process for random benchmark generation is required, which can provide benchmarks with the densities closer to the their lower bound.

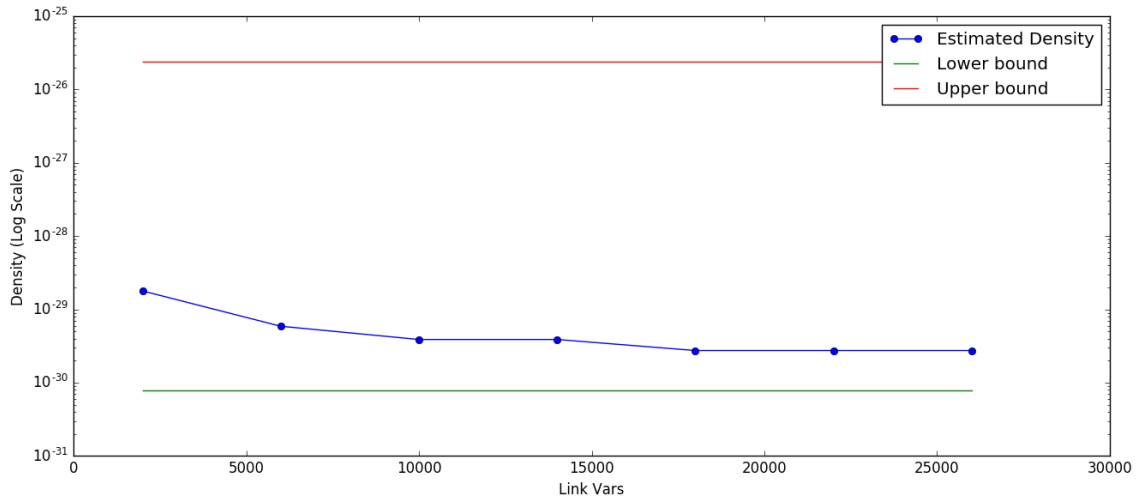


Figure 6.1 : Solution density μ (estimated) vs #link variables for $n = m = 30000$, $w = 100$

While $2^{n-w} \leq |R_\phi| \leq m * 2^{n-w}$, controlling $|R_\phi|$ within this range without using a degenerate formula with repeated cubes is non-trivial. We achieved coarse control over the $|R_\phi|$ and by extension μ and μ' , that was sufficient to demonstrate our experimental results by generating monotone formulas using the following high level idea:

1. the first cube is generated by randomly picking w variables out of possible n .
2. A literal in the cube generated in step 1 is selected at random. Then η new cubes are generated from the cube by swapping the chosen literal with η other literals one at a time.
3. Steps 1 and 2 are repeated by using a cube picked at random from the ones generated so far, until the desired number of cubes m is reached.

The number of 'link variables' η controls the densities of the formula obtained.

If η is close to m , then almost all cubes differ from each other in at most one literal resulting in a large intersection of their solution spaces. Consequently, $|R_\phi|$ which is the union of the solution spaces of all cubes is small which makes μ' close to the lower bound. On the other hand, if η is close to 1 then because of the random choice of literal to be swapped, most cubes have few literals in common leading to a large union and μ' close to the upper bound. η is thus a proxy for solution density. We demonstrate the effectiveness of this method in figure 6.1, which shows the solution density μ of benchmarks generated using the above process versus the number of link variables, while keeping all other parameters fixed. Link variables thus enable the generation of random benchmarks with solution densities closer to the lower bound.

6.6 Experimental Results

Our experiments show that worst case theoretical analysis does not provide a complete picture of which algorithm is the best for approximating $\#DNF$. Even algorithms like KL Counter and Vazirani Counter which have the same worst case complexity performed very differently in our experiments as did `SymbolicDNFAproxMC` and KLM Counter. Preliminary exploratory experiments had showed us that the formula terrain as defined in the previous sections, is very heterogeneous; different algorithms perform well in different parts of the terrain and predicting the regions where a particular algorithm does well is non-trivial. Instead of attempting the unmanageable and uninformative task of benchmarking the entire terrain, we choose to provide snapshots of interesting features instead:

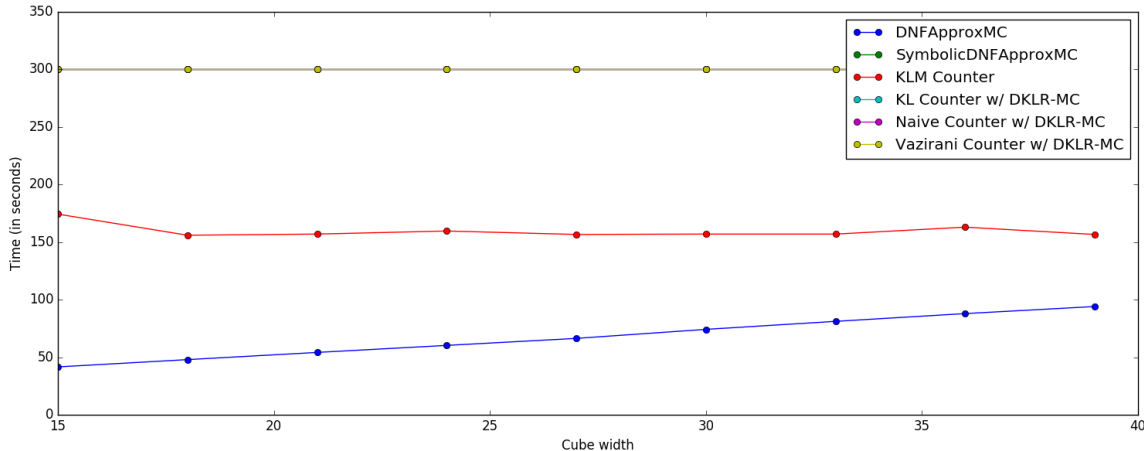


Figure 6.2 : DNFApproxMC is the best performer when $n = m = 30000$, $15 \leq w \leq 40$, $\eta = 20000$, $\varepsilon = 0.8$, $\delta = 0.36$, 300s timeout. (All algorithms except DNFApproxMC and KLM Counter timed out)

6.6.1 DNFApproxMC outperforms SymbolicDNFApproxMC on the benchmarks we tested

This is evident from figure 6.2. DNFApproxMC performs better than SymbolicDNFApproxMC despite the worst case complexity of SymbolicDNFApproxMC being better than that of DNFApproxMC by a factor of $\mathcal{O}(m)$. This can be attributed to the fact that `hiThresh` for SymbolicDNFApproxMC is twice that for DNFApproxMC and also the additional step of translating \mathbf{z} to (σ, ϕ^{C_i}) , though inexpensive in theory, takes non-negligible time in practice. However there is a caveat: the running time of DNFApproxMC scales linearly with the width of cubes while SymbolicDNFApproxMC is unaffected. Conceivably SymbolicDNFApproxMC may outperform DNFApproxMC for very large cube widths, though all benchmarks we tried to generate to demonstrate this timed out.

6.6.2 No algorithm consistently outperforms all other algorithms

This is evident from figures 6.2,6.4,6.3 and 6.5. Each data point in the plots corresponds to the average time taken by an algorithm on 3 formulas generated randomly with the parameters specified in the plots, using the link variable process described above. We found that the times for each of the three formulas corresponding to a data point were almost the same, for all algorithms. Each of DNFAproxMC, KLM Counter, Naive Counter with DKLR-MC, and KL Counterwith DKLR-MC were best performers in at least one part of the formula terrain. KL Counter is the best performer when number of link variables η is small (μ' is close to the upper bound) and ε is small. KLM Counter performs well when η is large (μ' is close to the lower bound) and w is large. *NaiveCounter* fares well when w is very small, while DNFAproxMC does well for intermediate values of w . We do not claim to have completely charted the entire space of formulas. Nevertheless, we were unable to find instances where Vazirani Counterwith DKLR-MC was the best performer. This warrants more comprehensive experimentation in the weighted case as well, as Vazirani Counterwith DKLR-MC may not be the best baseline as assumed in some prior work [25].

6.6.3 DNFAproxMC outperforms all other algorithms on certain benchmarks

This can be seen in figure 6.2. DNFAproxMC is a best performer for benchmark formulas with low density of solutions in the transformed space and when the cube width is not too large. This result cements the claim of hashing-based techniques as a powerful approach for counting boolean formulas, not just in theory but also in practice.

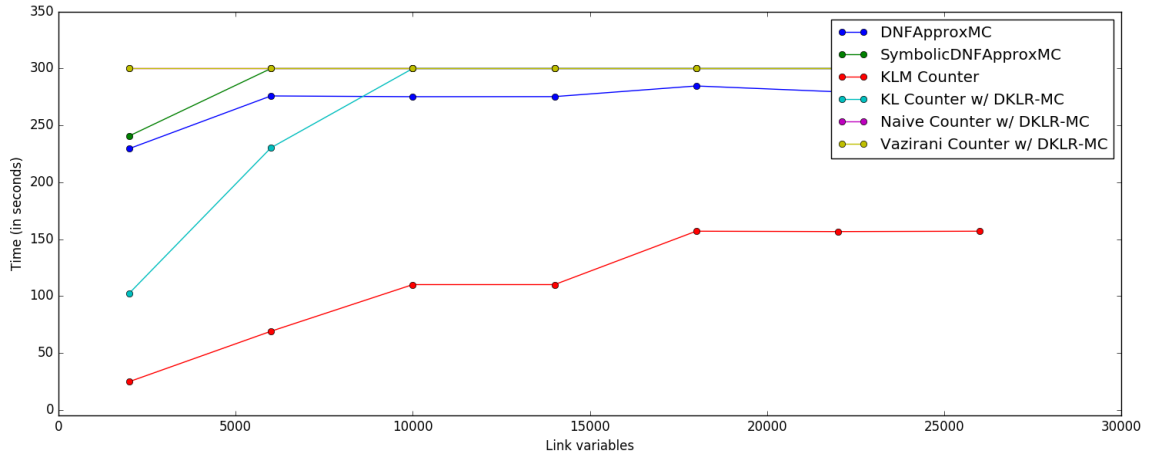


Figure 6.3 : KLM Counter is the best performer when $n = m = 30000, w = 100, 2000 \leq \eta \leq 26000, \varepsilon = 0.8, \delta = 0.36, 300s$ timeout.

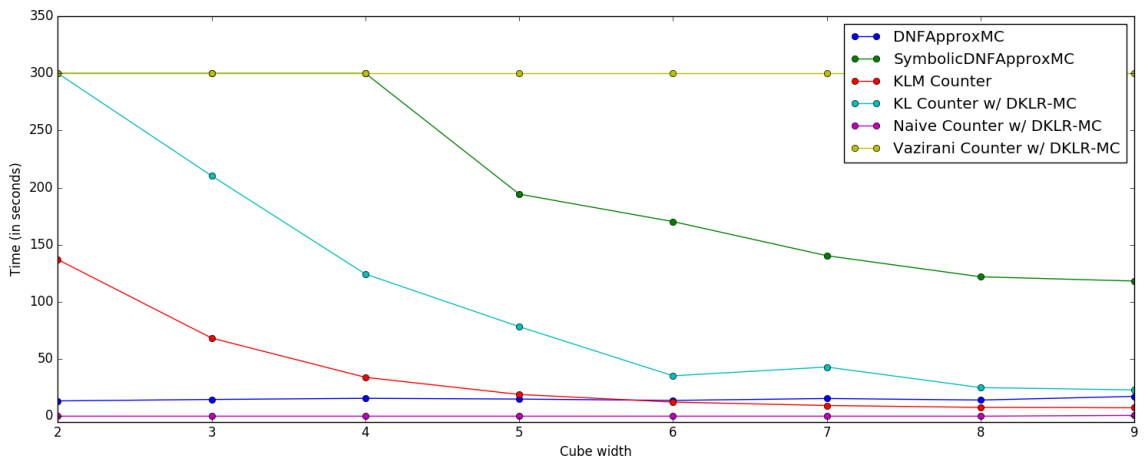


Figure 6.4 : Naive Counter w/ DKLR-MC is the best performer when $n = m = 30000, 2 \leq w \leq 9, \eta = 400, \varepsilon = 0.8, \delta = 0.36, 300s$ timeout.

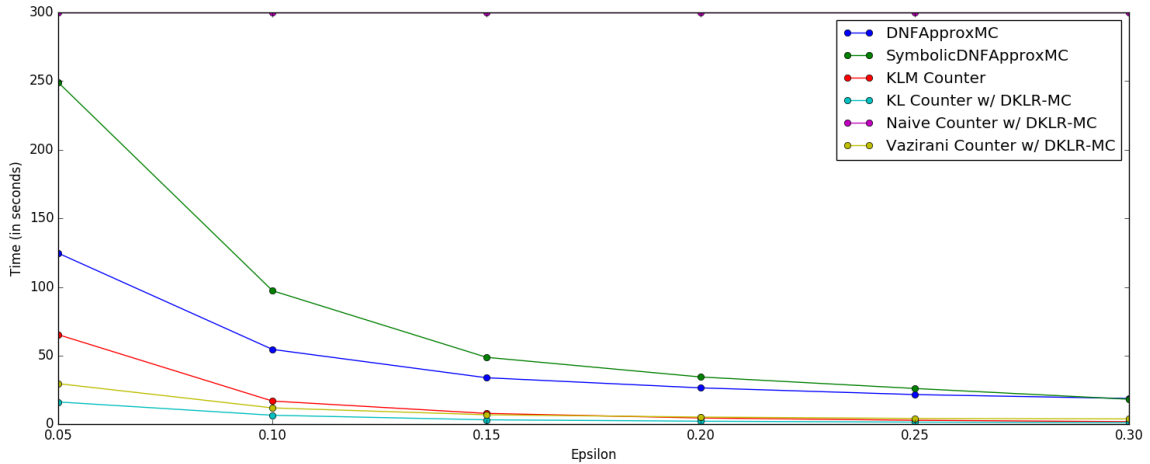


Figure 6.5 : KL Counter w/ DKLR-MC is the best performer when $n = m = 30000$, $w = 100$, $\eta = 1$, $\varepsilon \leq 0.3$, $\delta = 0.36$, 300s timeout.

6.7 Other techniques

A more meaningful estimate can be obtained for DNF formulas with small cubes and counts close to 2^n , by counting the negative space i.e. the complementary CNF formula instead. Note that a count of the negative space with multiplicative error bounds only makes sense when the negative space is equal or smaller than the original space. Given the success of ApproxMC for CNF formulas, it was an interesting question to see how ApproxMC fared against the six candidate algorithms above. Therefore, we used ApproxMC2 for CNF to count the complement of DNF formulas with large counts. ApproxMC2, however, timed out on all benchmarks with more than 10000 variables.

Another method for approximate $\#DNF$ is to first convert the DNF formula to CNF using Tseitin encoding. The resultant CNF formula can be counted using ApproxMC2. However this method too failed to scale beyond small formulas.

Summary We compared `SymbolicDNFAproxMC` and `DNFAproxMC` to the state-of-the-art Monte Carlo algorithms. Our results show that `DNFAproxMC` outperforms all other algorithms on certain benchmarks. Hashing is thus an important tool not only for `#CNF` but also `#DNF`. Significantly, no algorithm was the undisputed winner, which leads us to recommend using an algorithmic portfolio for approximate `#DNF` of which `DNFAproxMC` is an important member. We leave the problem of quickly finding the best algorithm to approximately count a given formula using heuristics as future work. Until a fast heuristic for selecting the best algorithm for the job is found, a multi-algorithm approach may be used where all algorithms are run in parallel and the result from the first one to terminate is returned.

Chapter 7

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 [15] 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. Our experiments demonstrate that there is currently no single best algorithm for approximate DNF counting; nevertheless hashing-techniques are important members of the algorithmic portfolio.

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 [37, 38, 39]. An interesting direction for

future work is to test H_{REX} family with CNF formulas. On the experimental side, it would be interesting to develop fast heuristics for selecting the best algorithm for a given formula.

Bibliography

- [1] F. Bacchus, S. Dalmao, and T. Pitassi, “Algorithms and complexity results for #SAT and Bayesian inference,” in *Proc. of FOCS*, pp. 340–351, 2003.
- [2] N. Dalvi and D. Suciu, “Efficient query evaluation on probabilistic databases,” *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 16, no. 4, pp. 523–544, 2007.
- [3] C. Domshlak and J. Hoffmann, “Probabilistic planning via heuristic forward search and weighted model counting,” *Journal of Artificial Intelligence Research*, vol. 30, no. 1, pp. 565–620, 2007.
- [4] T. Sang, P. Beame, and H. Kautz, “Performing bayesian inference by weighted model counting,” in *Prof. of AAAI*, pp. 475–481, 2005.
- [5] L. Valiant, “The complexity of enumeration and reliability problems,” *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.
- [6] J. D. Park and A. Darwiche, “Complexity results and approximation strategies for map explanations,” *Journal of Artificial Intelligence Research*, pp. 101–133, 2006.
- [7] D. Roth, “On the hardness of approximate reasoning,” *Artificial Intelligence*, vol. 82, no. 1, pp. 273–302, 1996.

- [8] M. Jerrum, L. Valiant, and V. Vazirani, “Random generation of combinatorial structures from a uniform distribution,” *Theoretical Computer Science*, vol. 43, no. 2-3, pp. 169–188, 1986.
- [9] R. Karp, M. Luby, and N. Madras, “Monte-Carlo approximation algorithms for enumeration problems,” *Journal of Algorithms*, vol. 10, no. 3, pp. 429–448, 1989.
- [10] 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.
- [11] L. Stockmeyer, “The complexity of approximate counting,” in *Proc. of STOC*, pp. 118–126, 1983.
- [12] P. Dagum, R. Karp, M. Luby, and S. Ross, “An optimal algorithm for monte carlo estimation,” *SIAM Journal on computing*, vol. 29, no. 5, pp. 1484–1496, 2000.
- [13] R. Karp and M. Luby, “Monte-carlo algorithms for enumeration and reliability problems,” *Proc. of FOCS*, 1983.
- [14] S. Chakraborty, K. S. Meel, and M. Y. Vardi, “A scalable approximate model counter,” in *Proc. of CP*, pp. 200–216, 2013.
- [15] 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.
- [16] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman, “Taming the curse of dimensionality: Discrete integration by hashing and optimization,” in *Proc. of ICML*, pp. 334–342, 2013.

- [17] C. P. Gomes, A. Sabharwal, and B. Selman, “Model counting: A new strategy for obtaining good bounds,” in *Proc. of AAAI*, vol. 21, pp. 54–61, 2006.
- [18] L. Duenas-Osorio, K. S. Meel, R. Paredes, and M. Y. Vardi, “Counting-based reliability estimation for power-transmission grids,” in *AAAI*, pp. 4488–4494, 2017.
- [19] 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>.
- [20] K. S. Meel, A. A. Shrotri, and M. Y. Vardi, “On hashing-based approaches to approximate dnf-counting,” in *Proceedings of IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, dec 2017.
- [21] S. Toda, “On the computational power of PP and (+)P,” in *Proc. of FOCS*, pp. 514–519, IEEE, 1989.
- [22] L. Babai, “Monte-carlo algorithms in graph isomorphism testing,” *Université tde Montréal Technical Report, DMS*, pp. 79–10, 1979.
- [23] R. Motwani and P. Raghavan, *Randomized algorithms*. Chapman & Hall/CRC, 2010.
- [24] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.
- [25] D. Olteanu, J. Huang, and C. Koch, “Approximate confidence computation in probabilistic databases,” in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pp. 145–156, IEEE, 2010.

- [26] J. L. Carter and M. N. Wegman, “Universal classes of hash functions,” in *Proceedings of the ninth annual ACM symposium on Theory of computing*, pp. 106–112, ACM, 1977.
- [27] G. Strang, *Introduction to linear algebra*, vol. 3. Wellesley-Cambridge Press Wellesley, MA, 1993.
- [28] F. Gray, “Pulse code communication,” Mar. 17 1953. US Patent 2,632,058.
- [29] D. E. Knuth, “Generating all n-tuples,” *The Art of Computer Programming*, vol. 4, 2004.
- [30] V. Gogate and R. Dechter, “Approximate counting by sampling the backtrack-free search space,” in *Proc. of the AAAI*, vol. 22, p. 198, 2007.
- [31] G. Markowsky, J. Lawrence Carter, and M. Wegman, “Analysis of a universal class of hash functions,” *Mathematical Foundations of Computer Science 1978*, pp. 345–354, 1978.
- [32] W. Gatterbauer and D. Suciu, “Oblivious bounds on the probability of boolean functions,” *ACM Transactions on Database Systems (TODS)*, vol. 39, no. 1, p. 5, 2014.
- [33] D. Suciu, D. Olteanu, C. Ré, and C. Koch, “Probabilistic databases,” *Synthesis Lectures on Data Management*, vol. 3, no. 2, pp. 1–180, 2011.
- [34] “TPC Benchmark H.” <http://www.tpc.org/>.
- [35] M. Albrecht and G. Bard, *The M4RI Library – Version 20121224*. The M4RI Team, 2012.

- [36] J. Huang, L. Antova, C. Koch, and D. Olteanu, “Maybms: a probabilistic database management system,” in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 1071–1074, ACM, 2009.
- [37] A. Ivrii, S. Malik, K. S. Meel, and M. Y. Vardi, “On computing minimal independent support and its applications to sampling and counting,” *Constraints*, pp. 1–18, 2015.
- [38] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman, “Low-density parity constraints for hashing-based discrete integration,” in *Proc. of ICML*, pp. 271–279, 2014.
- [39] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman, “Short XORs for Model Counting; From Theory to Practice,” in *SAT*, pp. 100–106, 2007.