Not All FPRASs are Equal: Demystifying FPRASs for DNF-Counting (Extended Abstract) *†

Kuldeep S. Meel[1], **Aditya A. Shrotri**[2] and **Moshe Y. Vardi**[2]
[1]National University of Singapore, Singapore
[2]Rice University, Houston USA
meel@comp.nus.edu.sg, {Aditya.Aniruddh.Shrotri, vardi}@rice.edu

Abstract

The problem of counting the number of solutions of a DNF formula, also called #DNF, is a fundamental problem in AI with wide-ranging applications. Owing to the intractability of the exact variant, efforts have focused on the design of approximate techniques. Consequently, several Fully Polynomial Randomized Approximation Schemes (FPRASs) based on Monte Carlo techniques have been proposed. Recently, it was discovered that hashing-based techniques too lend themselves to FPRASs for #DNF. Despite significant improvements, the complexity of the hashing-based FPRAS is still worse than that of the best Monte Carlo FPRAS by polylog factors. Two questions were left unanswered in previous works: Can the complexity of the hashing-based techniques be improved? How do these approaches compare empirically? In this paper, we first propose a new search procedure for the hashing-based FPRAS that removes the polylog factors from its time complexity. We then present the first empirical study of runtime behavior of different FPRASs for #DNF, which produces a nuanced picture. We observe that there is no single best algorithm for all formulas and that the algorithm with one of the worst time complexities solves the largest number of benchmarks.

1 Introduction

Constrained counting is a fundamental problem in artificial intelligence with a wide variety of applications ranging from network reliability [Dueñas-Osorio *et al.*, 2017], probabilistic inference [Bacchus *et al.*, 2003], probabilistic databases [Dalvi and Suciu, 2007], quantified information flow [Biondi *et al.*, 2018], and the like. Given a set of constraints F, the problem of constrained counting seeks to compute the total number of solutions to F. In this work, we focus on the variant of constrained counting where F is expressed in Disjunctive Normal Form (DNF), henceforth denoted as DNF-Counting or #DNF. This problem is important in practice, as applications such as query evaluation in probabilistic databases [Dalvi and Suciu, 2007] and failure-probability estimation of networks [Karger, 2001] reduce to it.

The problem of #DNF is known to be #P-complete [Valiant, 1979], where #P is the class of counting problems for decision problems in NP. Due to the intractability of exact #DNF, the approximate variant of #DNF has been studied extensively by both theoreticians and practitioners. Of particular interest is to obtain (ε, δ) approximation, such that the count returned by the approximation scheme is within $(1 + \varepsilon)$ factor of the exact count with confidence at least $1 - \delta$, where ε and δ are supplied by the user.

In their seminal paper, Karp and Luby [1983] proposed the first Fully Polynomial Randomized Approximation Scheme (FPRAS) for #DNF based on Monte Carlo sampling. We will henceforth use the term KL Counter to denote the FPRAS proposed by Karp et al. The time complexity of KL Counter is quadratic in the number of cubes (i.e., disjuncts) and linear in the number of the variables of the input formula F. Building on KL Counter, Karp, Luby and Madras [1989] proposed an improved FPRAS, henceforth denoted as KLM Counter, which has time complexity linear in the number of cubes. Vazirani [2013] proposed a variant of KL Counter (denoted Vazirani Counter) with same time complexity as KL Counter, but combined with an enhancement proposed in [Dagum *et al.*, 2000], it requires fewer Monte Carlo samples than KL Counter.

Recently, Chakraborty, Meel and Vardi [2016] showed that the hashing-based framework, which was originally proposed for approximate counting of CNF formulas, lends to an FPRAS scheme for #DNF as well. In particular, Chakraborty et al. proposed a hashing-based scheme called DNFApproxMC, whose time complexity was significantly worse than that of KLM Counter. Building on Chakraborty et al., Meel, Shrotri and Vardi [2017] proposed an improvement to DNFApproxMC, which we refer to as SymbolicDNFApproxMC. The time complexity of SymbolicDNFApproxMC is $\tilde{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$, which is within polylog factors of that of KLM Counter.

Two key questions however, are still unanswered: 1) Is it possible to remove the polylog factors in the complexity of SymbolicDNFApproxMC? 2) How do the various approaches perform empirically? The desire to make an inquiry into the

*Author names are ordered alphabetically by last name.

†This is an abridged version of a full paper in Constraints [Meel *et al.*, 2018]

runtime performance of different FPRAS is not just intellectual; it stems from the fruitful results such a study has produced in the development of theory and tools for approximate #CNF [Ermon *et al.*, 2013; Meel, 2018]. Despite the fact that some FPRAS have been around for over 30 years, a comprehensive experimental evaluation has not been performed for #DNF, to the best of our knowledge.

In this paper, we propose a new search technique for hashing-based algorithms that improves the complexity of SymbolicDNFApproxMC to $\mathcal{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$, which is the same as KLM Counter. We also present the first empirical study of runtime behavior of different FPRASs for #DNF. Similar to previous studies for SAT solvers, we experiment on classes of randomly generated DNF formulas covering a broad range of distribution parameters. Our results produce a nuanced picture. Firstly, we observe that there is no single best algorithm that outperforms all other algorithms for all classes of formulas and input parameters. Second, we observe that the algorithm with one of the worst time complexities, DNFApproxMC, solves the largest number of benchmarks. We believe that the above two results are significant as they demonstrate a gap between runtime performance and theoretical time complexity of approximate techniques for #DNF. Similar to studies of #CNF, this gap should serve as a guiding light for designing new #DNF algorithms, and for analyzing the structure of solution space of DNF formulas.

2 #DNF and Approximation Schemes

A formula F over boolean variables is in Disjunctive Normal Form (DNF) if it is a disjunction over conjunctions of literals. We use $\#F$ to denote the number of solutions or satisfying assignments of F Disjuncts in the formula are called *cubes*. We will use n and m to denote the number of variables and number of cubes in the input DNF formula, respectively. We use w to denote the minimum of width over all the cubes of the formula. A *fully polynomial randomized approximation scheme* (FPRAS) is a randomized algorithm that takes as input a formula F, a tolerance $\varepsilon \in (0,1)$ and confidence parameter $\delta \in (0,1)$ and outputs a random variable Y such that $\Pr[(1-\varepsilon)\#F \leq Y \leq (1+\varepsilon)\#F] \geq 1-\delta$ and the running time of the algorithm is polynomial in $|F|, 1/\varepsilon, \log(1/\delta)$.

There is intense interest in practical applications of #DNF and a number of algorithmic schemes have been designed towards that end. The strongest guarantees on worst-case running time are provided by FPRASs, yet there does not exist a comprehensive experimental evaluation comparing them. In this work, we perform the first such empirical study of runtime behavior of different FPRASs. Before delving into the experiments, we briefly discuss the two main paradigms – Monte Carlo sampling and hashing – for developing FPRASs for #DNF. We will also present a new search technique that improves the theoretical and practical running time of the hashing-based approach.

2.1 Monte Carlo Framework

Algorithms built on Monte Carlo framework are randomized algorithms whose output can be wrong with a certain (usually small) probability [Babai, 1979]. Typically, these algorithms rely on drawing independent random samples to obtain numerical results. We refer the reader to [Motwani and Raghavan, 2010] for further details. In the context of counting, the abstract Monte Carlo framework for finding cardinality of a set $\mathcal{A}$ in the universe $\mathcal{U}$ is shown in Algorithm 1.

Algorithm 1 Monte-Carlo-Count($\mathcal{A}, \mathcal{U}$)

1: $Y \leftarrow 0$
2: **repeat** $N \propto \frac{|\mathcal{U}|}{|\mathcal{A}|}$ times
3: Select an element $t \in \mathcal{U}$ uniformly at random
4: **if** $t \in \mathcal{A}$ **then**
5: $Y \leftarrow Y + \frac{1}{N}$
6: $Z \leftarrow Y \times |\mathcal{U}|$
7: **return** Z

In the context of this work, we have $\mathcal{A}$ is the set of solution of the input DNF formula F. We can employ Algorithm 1 by defining $\mathcal{U}$ to be the set of all assignments over n variables. Algorithm 1 is an FPRAS if the number of samples N, and the time taken by line 3 and 4 are polynomial in the size of input. However, this is not true in general for DNF formulas, as $\#F$ can be exponentially smaller than 2^n.

The key insight by Karp et al. is to transform the universe $\mathcal{U}$ into a different universe $\mathcal{U}'$ while preserving the number of solutions and ensuring that $\frac{|\mathcal{U}'|}{\#F}$ is polynomially bounded. Different transformations and enumeration procedures gave rise to 3 different FPRASs, namely the KL Counter, KLM Counter and the Vazirani Counter. The best worst-case complexity is achieved by KL Counter, which is $\mathcal{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$.

2.2 Hashing Framework

The number of solutions of Boolean formulas that arise in practice is typically extremely large, and it is infeasible to enumerate them all. The key idea behind hashing-based counting is to partition the solution space of a given formula into *roughly equal sized* cells such that the number of solutions in a cell is not too large. Counting the solutions in a cell by enumeration is now feasible, and we can estimate the total count by extrapolating the count of a randomly chosen cell by the total number of cells in the partition. The use of 2-universal hash functions [Carter and Wegman, 1977] for partitioning ensures that the solutions are roughly evenly divided amongst cells, which is necessary for obtaining theoretical guarantees on the final count.

This abstract hashing-based counting framework called ApproxMC is illustrated in Algorithm 2. The procedure takes as input a Boolean formula F, tolerance ε and confidence δ, and outputs an approximate count within (ε, δ) bounds of the true count of F. A low-confidence estimate of the true count is obtained in lines 5-12, which is boosted to the required confidence δ, by taking the median of $\mathsf{Iterations} = \mathcal{O}(\log(\frac{1}{\delta}))$ counts on line 13.

The crux of the framework is a search for the right number of hash constraints on line 6. The number of hash constraints p returned by the search procedure is guaranteed to be such that the number of solutions in the cell is not too large, yet the tolerance is as required. The loop on line 8 is used to

Algorithm 2 ApproxMC(F, ε, δ)

```
1: Threshold ← O(1/ε²);
2: Iterations ← O(log(1/δ));
3: CountList ← emptyList;
4: repeat Iterations times
5:     h ← SampleHashFunction();
6:     p ← Search(F, Threshold);
7:     count ← 0;
8:     while True do
9:         if EnumerateNextSol(F ∧ h^p) ≠ ⊥ then
10:            count = count + 1;
11:        else break;
12:    AddToList(CountList, count × 2^p);
13: Y ← FindMedian(CountList);
14: return Y;
```

exhaustively enumerate all the solutions in the cell $(F \wedge \mathsf{h}^{\mathsf{p}})$.

Concrete counting algorithms for a class of formulas can be obtained from the above framework by choosing an appropriate family of 2-universal hash functions along with the corresponding procedures SampleHashFunction (line 5), Search (line 6) and EnumerateNextSol (line 9). Chakraborty et al. [2016] obtained an FPRAS for DNF formulas with complexity $\mathcal{O}((\mathsf{mn}^3 + \mathsf{mn}^2/\varepsilon^2)\log \mathsf{n} \log(1/\delta))$, using Random XOR hash functions, Gaussian Elimination for EnumerateNextSol, and binary-search for Search. We denote the resulting algorithm as DNFApproxMC. In our experiments, we augmented DNFApproxMC with Row-Echelon Hash family (proposed in [Meel *et al.*, 2017]), which improves the complexity from cubic to quadratic in n leading to better performance on all benchmarks.

The algorithm SymbolicDNFApproxMC proposed in [Meel *et al.*, 2017] achieves better worst-case time complexity, made possible by three improvements over the original DNFApproxMC algorithm, namely Row Echelon hash functions, Symbolic Hashing and Stochastic Cell-Counting. The complexity of SymbolicDNFApproxMC is $\tilde{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$.

3 Reverse Search for Hashing-Based Algorithms

A close inspection of the SymbolicDNFApproxMC algorithm in [Meel *et al.*, 2017] reveals that the polylog factors in the complexity analysis stems from the use of binary search which redundantly explores the same part of the solution space. We proposed a new search technique called ReverseSearch, that removes the polylog factors (hidden in the $\tilde{O}$ notation) from the complexity of SymbolicDNFApproxMC to make it at par with the complexity achieved KLM Counter, and also improves its running time in practice.

Theorem 1. *The complexity of* SymbolicDNFApproxMC*, when invoked with* ReverseSearch *is* $\mathcal{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$

(See [Meel *et al.*, 2018] for full proof). Naturally, one wonders whether employing ReverseSearch leads to gains in performance in practice. We compared the running times of SymbolicDNFApproxMC with BinarySearch and with ReverseSearch over wide classes of randomly generated DNF formulas with $100,000$ variables, number of cubes ranging from $10,000$ to $800,000$ and cube-widths ranging from 3 to 43. Figure 1 shows a scatter-plot of the results. A point (in blue) in the plot corresponds to one DNF formula in our test set. Its y-coordinate represents the time taken by SymbolicDNFApproxMC using ReverseSearch, while its x-coordinate represents time taken using BinarySearch. It can be seen that SymbolicDNFApproxMC with ReverseSearch is roughly four or five times faster than with BinarySearch. Therefore in the empirical study we describe next, we use ReverseSearch in all experiments involving SymbolicDNFApproxMC. Henceforth, we denote SymbolicDNFApproxMC with ReverseSearch as just SymbolicDNFApproxMC. Note, however, that DNFApproxMC does not benefit from ReverseSearch (Fig. 2). In fact, a simple linear search works best since our implementation buffers solutions which obviates the need for other searches.

4 Empirical Evaluation

The objective of our experimental evaluation was to seek an answer for the following four key questions:

1. Runtime Variation: How does the running time of the algorithms vary across different benchmarks?
2. Benchmarks Solved: How many benchmarks can the algorithms solve overall?
3. Accuracy: How accurate are the returned counts?
4. $\varepsilon - \delta$ Scalability: How do the algorithms scale with the input tolerance and confidence?

To the best of our knowledge, there is no publicly-available standardized set of benchmarks for #DNF. This prevents adoption of algorithms in practice, which in turn affects benchmark availability. To break this vicious cycle, we conduct our study on random DNF formulas covering a broad range of parameters. We only present results on Runtime Variation and Benchmarks Solved; see [Meel *et al.*, 2018] for full details.

4.1 Runtime Variation

We present a graph of the running time vs. the number of cubes for $\mathsf{w} = 3, 23$ as well as for non-uniform cube-widths. This is shown in Figs. 3, 4, and 5 respectively. Each data point in the graphs represents the average running time of an algorithm over the 20 random formulas that were generated with the corresponding n, m and w. For $\mathsf{w} = 3$, we see that DNFApproxMC vastly outperforms other algorithms, while in Fig. 4 we see that the performance of Monte Carlo methods improves. For non-uniform widths, we see that the DNFApproxMC is again the best performer.

4.2 Benchmarks Solved

Fig. 6 shows the cactus plot of all the different algorithms.

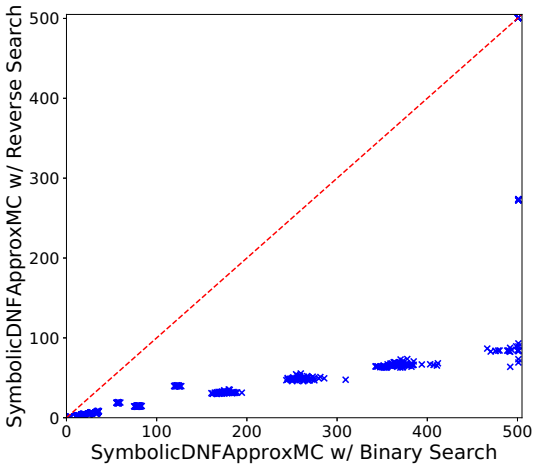

Figure 1: Running time of SymbolicDNFApproxMC: ReverseSearch vs. BinarySearch

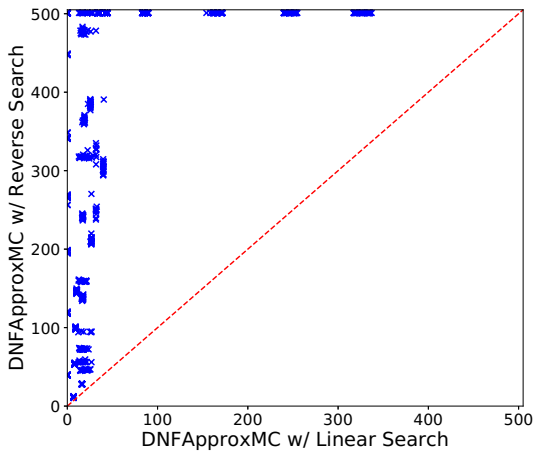

Figure 2: Running time of DNFApproxMC: ReverseSearch vs. LinearSearch

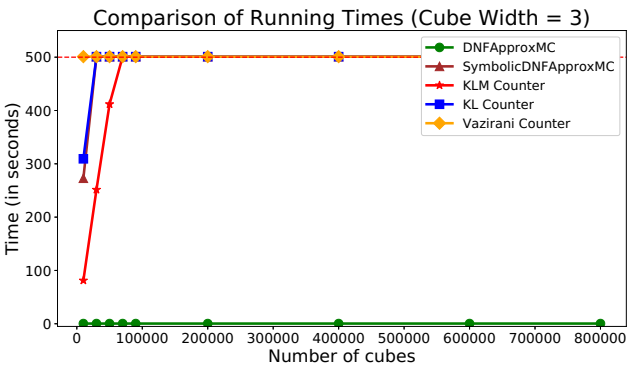

Figure 3: Runtime Variation: DNFApproxMC is the best performer. Rest timeout.

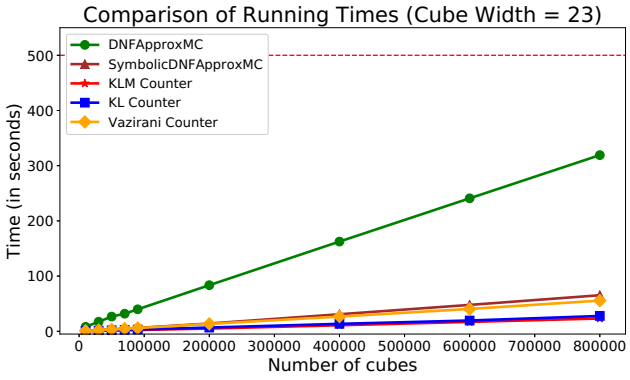

Figure 4: Runtime Variation: KLM Counter and KL Counter are the best performers

We present the number of benchmarks on x–axis and the total time taken on y–axis. A point (x, y) implies that x benchmarks took less than or equal to y seconds to solve. We see that DNFApproxMC completes all 1080 benchmarks in under 350 seconds which is well within the time limit of 500 seconds. Other algorithms time out on at least 100 benchmarks.

5 Discussion

The Runtime Variation experiments show that the performance of Monte Carlo FPRAS and SymbolicDNFApproxMC is adversely affected by cube width, while DNFApproxMC is consistent across all benchmarks, as evidenced by the Benchmarks Solved. The reason for this behavior, is that the former crucially depend on the density of solutions, which varies exponentially with cube-width for randomly generated formulas. On the other hand the running time of DNFApproxMC does not depend as heavily on the solution density which is also apparent in the experiment on formulas with non-uniform cube-widths (Fig. 5).

In the full version, we show that in terms of ε Scalability, DNFApproxMC performs better than other approaches due to the use of efficient data structures to buffer solutions, while the Monte Carlo algorithms scale substantially better with δ, which can be attributed to the overhead in repeating the core hashing sub-procedure for boosting confidence. Vazirani Counter achieves the best accuracy for a given (ε, δ); however this comes at a price as it solved the least number of benchmarks. Note that the maximum observed error was much lower than the input ε for all 5 algorithms.

In summary, KLM Counter and KL Counter are the algorithms of choice when solution density in the transformed space is known to be high. However, when there is no information about the formula or when solutions density is known to be low, DNFApproxMC is a safe bet.

6 Concluding Remarks

Designing counters for #DNF has been of practical and theoretical interest, owing to various applications in AI. Building on Chakraborty et al. [2016], Meel et al. [2017] proposed a hashing-based algorithm, SymbolicDNFApproxMC, whose time complexity was shown to be within polylog factors of the best known Monte Carlo schemes. Meel et al. left two key questions answered: (1) Are hashing techniques as powerful as Monte Carlo?, and (2) How do they compare empirically?

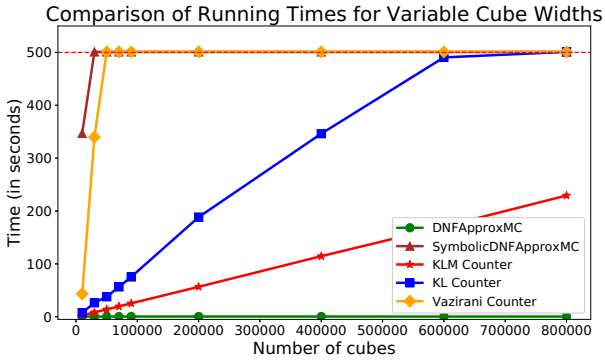

Figure 5: Runtime Variation: DNFApproxMC dominates other algorithms

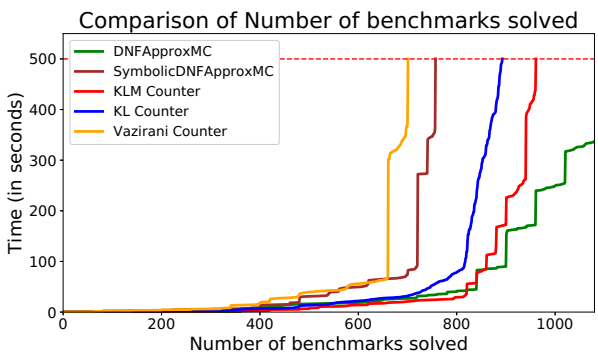

Figure 6: Benchmarks Solved: DNFApproxMC solved all benchmarks

We provide positive answers to these questions. In particular, we first introduced a new reverse-search technique that makes the time complexity of a hashing-based FPRAS at par with the state-of-the art Monte Carlo techniques. This leads to up to $4-5\times$ gains over the previous scheme proposed by Meel et al. [2017]. Moreover, reverse-search for hashing is not limited to #DNF, and maybe applied to #CNF as well.

We also provided the first empirical study of the various Monte Carlo and hashing-based FPRASs for #DNF. Our analysis leads to two important observations not apparent from the theoretical analysis alone: (1) There is no panacea; different algorithms are well suited for different formula types and input parameters; and (2) DNFApproxMC is robust across different classes of formulas, despite poor complexity.

Owing to comprehensive testing on a wide array of formula classes and input parameters, we believe that these observations will carry over to real-world benchmarks as well. These observations illustrate a gap between theory and practice of #DNF which we hope will kick-start further empirical investigations and serve as a blueprint for future work on #DNF.

Acknowledgements

Work supported in part by NSF grant IIS-1527668, NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering", and by NUS ODPRT Grant, R-252-000-685-133.

References

[Babai, 1979] L. Babai. Monte-carlo algorithms in graph isomorphism testing. *Université tde Montréal Technical Report, DMS*, pages 79–10, 1979.

[Bacchus *et al.*, 2003] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *Proc. of FOCS*, pages 340–351, 2003.

[Biondi *et al.*, 2018] F. Biondi, M. Enescu, A. Heuser, A. Legay, K. S. Meel, and J. Quilbeuf. Scalable approximation of quantitative information flow in programs. In *Proc. of VMCAI*, 1 2018.

[Carter and Wegman, 1977] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *Proc. of STOC*, pages 106–112. ACM, 1977.

[Chakraborty *et al.*, 2016] 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.

[Dagum *et al.*, 2000] P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on Computing*, 29(5):1484–1496, 2000.

[Dalvi and Suciu, 2007] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, 2007.

[Dueñas-Osorio *et al.*, 2017] L. Dueñas-Osorio, K. S. Meel, R. Paredes, and M. Y. Vardi. SAT-based connectivity reliability estimation for power transmission grids. Technical report, Rice University, 2017.

[Ermon *et al.*, 2013] 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*, pages 334–342, 2013.

[Karger, 2001] D. R. Karger. A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliability Problem. *SIAM Review*, 2001.

[Karp and Luby, 1983] R.M. Karp and M. Luby. Monte Carlo algorithms for enumeration and reliability problems. *Proc. of FOCS*, 1983.

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

[Meel *et al.*, 2017] K. S. Meel, A. A. Shrotri, and M. Y. Vardi. On hashing-based approaches to approximate DNF-counting. In *Proc. of FSTTCS*, 12 2017.

[Meel *et al.*, 2018] Kuldeep S Meel, Aditya A Shrotri, and Moshe Y Vardi. Not all fprass are equal: demystifying fprass for dnf-counting. *Constraints*, pages 1–23, 2018.

[Meel, 2018] Kuldeep S Meel. Constrained counting and sampling: Bridging the gap between theory and practice. *arXiv preprint arXiv:1806.02239*, 2018.

[Motwani and Raghavan, 2010] R. Motwani and P. Raghavan. *Randomized algorithms*. 2010.

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

[Vazirani, 2013] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.