Behind the Scenes of Building a Scalable Approximate Model Counter (Draft Version)

Supratik Chakraborty[1], Kuldeep S. Meel[2], and Moshe Y. Vardi[3]

[1] Department of Computer Science and Engineering, I.I.T. Bombay, India
[2] Department of Computer Science, National University of Singapore, Singapore
[3] Department of Computer Science, Rice University

1 Introduction and preliminaries

Counting and uniformly sampling solutions of a system of constraints are fundamental problems in Computer Science with applications spanning diverse areas. Example applications include functional verification of circuits and programs, partition function estimation, probabilistic inference, network reliability estimation, quantifying information leakage in programs and the like. Exact counting and sampling are known to be computationally hard and are widely believed to lie beyond the polynomial hierarchy. Significant effort has therefore been invested over the last four decades in designing and analyzing approximate algorithms for solving these problems.

For clarity of exposition, let us quickly review some terminology before delving deeper. We will be primarily concerned with propositional constraints, represented by a formula φ. We call the set of all variables in φ the *support* of φ, and denote it by $\mathsf{Sup}(\varphi)$. A satisfying assignment or model of φ is an assignment of truth values to all variables in $\mathsf{Sup}(\varphi)$ such that φ evaluates to 1. We use $\mathsf{Sol}(\varphi)$ to denote the set of all models of φ. The *model counting* problem asks: *Given φ, find* $|\mathsf{Sol}(\varphi)|$. Several approximate variants of this problem have been studied in the literature. The predominant one among them concerns the following question: *Given φ, a tolerance measure ε (> 0) and a confidence measure δ ($0 < \delta \leq 1$), find a random number c (≥ 0) such that* $\mathsf{Pr}\left[|\mathsf{Sol}(\varphi)| \cdot (1+\varepsilon)^{-1} \leq c \leq |\mathsf{Sol}(\varphi)| \cdot (1+\varepsilon)\right] \geq 1 - \delta$. This is also known as *probably approximately correct* (or PAC) model counting. The problem of *uniform sampling* is intimately related to that of model counting. In uniform sampling, we ask: *Given φ, return a random assigment π to variables in $\mathsf{Sup}(\varphi)$ such that $\mathsf{Pr}[\pi$ is returned$] = c$ (> 0) if $\pi \in \mathsf{Sol}(\varphi)$ and $\mathsf{Pr}[\pi$ is returned$] = 0$ otherwise, where c is a positive real number that is independent of π.* In the approximate variant of uniform sampling, also called *almost uniform sampling*, we are given a tolerance ε (> 0), and we require that $c \cdot (1+\varepsilon)^{-1} \leq \mathsf{Pr}[\pi$ is output$] \leq c \cdot (1+\varepsilon)$ *if* $\pi \in \mathsf{Sol}(\varphi)$ *and* $\mathsf{Pr}[\pi$ is output$] = 0$ *otherwise.*

2 Understanding the problem

In the winter of 2011, we started surveying the literature on approximate algorithms for sampling and counting, and came up with a rather curious observation. Existing techniques (as of 2011-2012) largely fell in one of two categories: (a) those that provided strong approximation guarantees but did not scale well in practice, and (b) those that scaled well in practice but provided weak or no provable approximation guarantees. An example of a technique in the first category is Stockmeyer's randomized approximation scheme [24] for model counting using 2-universal hash functions, while examples of the second category include Monte Carlo estimators where the number of Monte Carlo steps is truncated heuristically before the theory-mandated mixing of states happen [25]. As we surveyed the literature, it soon became clear that marrying practical scalability with strong approximation guarantees had remained an elusive goal despite a few decades' worth of extremely impressive research. This motivated us to delve deeper to try to understand what was amiss in bridging this gap between theory and practice.

In a seminal paper published in 1986, Jerrum, Valiant, and Vazirani showed that approximate counting and almost-uniform sampling are polynomially interreducible [19]. From a theoretical perspective, the two problems can therefore be viewed as different sides of the same coin. In the same paper, Jerrum, Valiant and Vazirani also argued that Stockmeyer's ground-breaking work of 1983 [24] had all the ingredients to yield a randomized approximation scheme for model counting that makes polynomially many calls to an NP-oracle. Given that modern propositional satisfiability (henceforth called SAT) solvers routinely check satisfiability of formulas with hundreds of thousands to millions of variables, this may lead one to falsely believe that it is not too difficult to obtain a practically efficient model counter with strong approximation guarantees. On the face of it, all that needed to be done was to implement Stockmeyer's algorithm with a state-of-the-art SAT solver replacing the NP-oracle. Once this is done, Jerrum, Valiant and Vazirani's reduction could perhaps even be used to obtain a practically efficient almost uniform sampler. Unfortunately, nobody had been able to accomplish this feat succesfully until then, hinting that things were not as simple as they appeared. A little experimentation made us realize that while the theory is beautiful, the devil lies in the details of this theory is translated to tools that can be run on benchmarks. The plot was thickening and we decided to drill down further.

3 The gap between theory and practice

Our literature survey showed that the theoretical discourse on approximate counting and sampling had largely revolved around its complexity *relative to an* NP*-oracle*. Specifically, it was known that these problems were in $\mathsf{BPP}^{\mathsf{NP}}$ or could be polynomially reduced to this complexity class, where the polynomial is in the size of φ, $1/\varepsilon^2$ and $\log_2(1/\delta)$. Unfortunately, this said very little about the

running times of these algorithms in practice. For oracle-based algorithms to be useful in practice, it was necessary to go beyond the theoretical models and consider the *overall* running time. This meant that we needed to take into account additional considerations that had hitherto been de-emphasized in the theoretical discourse. Specifically, it was clear that the basic BPP algorithm should have a very low-degree polynomial running time. In addition, four decades of meticulously documented SAT research had shown that there was wide variation in SAT solving times across classes of problem instances and also across the space of heuristics used. Therefore, if we wanted to design an approximate model counter or sampler with a SAT solver in place of an NP-oracle, it was necessary to ensure that (i) we didn't blow up the sizes of SAT problem instances fed as inputs to the oracle, and (ii) the oracle queries were such that an extant SAT solver could solve them reasonably efficiently in practice.

Armed with this insight, we went back to the drawing board and asked ourselves the following question: *Can we design an approximate counting/sampling algorithm that makes use of a modern* SAT *solver as a black-box (instead of an* NP*-oracle), and makes judicious use of the strengths and limitations of state-of-the-art* SAT *solving techniques?* This called for a change in the way our algorithm had to be designed. Specifically, we could no longer grant ourselves the freedom to invoke a SAT solver polynomially many times for a high-degree polynomial, or even with an input formula that had a very large support. Every invocation of a SAT solver had a cost that could not be ignored, and our algorithm had to optimize this cost as much as possible. We were also required to see if we could exploit empirically observed strengths of modern SAT solvers in the design of our algorithm. For example, incremental SAT solving is known to yield improved performance vis-a-vis stand-alone SAT solving. Similarly, the performance of SAT solvers that accept as inputs CNF+XOR clauses are known to deteriorate rapidly as the size of XOR clauses increases. Was there a way to exploit these empirically observed properties in the design of our algorithm so that it ran efficiently in practice? At the same time, we could not afford to rest with just optimizing the peformance of our algorithm. We also needed to ensure that every design choice we made didn't end up breaking any formal approximation guarantees offered by our algorithm. Clearly, this was going to be a very tight rope-walk, but certainly well worth trying. The technical challenges and potential impact got us really fired up.

We went back and re-examined Stockmeyer's model counting algorithm and realized that in order to obtain an approximate count within a user-specified tolerance ε of the true model count, we would need to make $\Theta\left(\frac{1}{\log_2(1+\varepsilon)}\right)$ independent copies of the original system of constraints and conjoin them before feeding them to a SAT solver. This had important consequences, since it entailed a blow-up in the support of the formula, which in turn entailed an increase in the number and size of XOR clauses in the formula fed as input to the SAT solver. We also looked at techniques that weren't directly derived from Stockmeyer's approach. Since counting and sampling are two faces of the same coin, we looked up relevant work on approximate sampling as well. We found that in

2000, Bellare, Goldreich and Petrank [5] had proposed an algorithm for almost uniform generation of models of propositional formulas using hash functions of high degree of universality and an NP-oracle. Although the theory is absolutely beautiful, our best implementation efforts in 2012 revealed that the approach scaled only to formulas with support of around 20 variables. The big hurdle that came in the way of scalability in this case was the need to invert a hash function with a high degree of universality. Encoding this requirement as a propositional formula resulted in complex constraints that choked state-of-the-art SAT solvers. The need to incorporate insights from modern SAT solvers in the design of the algorithm was staring at us in the face.

4 An attempt to bridge the gap

By the end of 2012, we knew that if our project was to succeed, we had to work with hash functions of low degree of universality, be overly cautious about the formulas being fed as inputs to the SAT solver, and ensure that nothing we did to accommodate the practical limitations of SAT solvers invalidated our proofs of approximation guarantees. Around this time, we also studied the work of Gomes, Sabharwal and Selman [17], who had proposed a parameterized approximate counting algorithm that used XOR constraints as 2-universal hash functions. This was a bold attempt to address the same concerns that we were trying to address. However, we quickly realized that using an algorithm whose performance crucially depends on user-provided parameters wasn't going to result in a practically useful model counter or sampler. Indeed, automatically choosing the right combination of parameters in Gomes, Sabharwal and Selman's algorithm turned out to be highly non-trivial in general, and this came in the way of scaling up the algorithm's performance. We needed to come up with an algorithm that was fully automatic – all parameters required for the correct and efficient operation of the algorithm should either be pre-determined and hard-coded in the algorithm, or computed automatically on-the-fly.

As we kept working on various alternative algorithms, it became clear that using XOR constraints as 2-universal (in fact, they are 3-universal as well) hash functions would be crucial in our approach. The literature contained sufficient warnings that CDCL SAT solvers performed badly when an input formula had XOR clauses encoded in CNF. Indeed, this is what we observed in our experiments as well. To scale up to large problem instances, we desperately needed a SAT solver that could benefit from the rich legacy of CDCL SAT solvers working on CNF formulas, and also on Gaussian elimination applied to XOR clauses. Around this time, we came across CryptoMiniSat, a CDCL SAT solver for CNF and XOR clauses, developed by Soos, Nohl and Castelluccia [23]. Although Soos et had developed CryptoMiniSat for cryptographic applications, it fit our requirements of a SAT solver for CNF+XOR clauses beautifully. This saved us at least a year's (perhaps more) work of integrating Gaussian reasoning on XOR clauses inside a state-of-the-art CDCL SAT solver for CNF formulas.

By the end of 2012, a simple idea was emerging for our CryptoMiniSat-based approximate counter and sampler The gist of our idea was as follows. Given a propositional formula φ, let $h_i(\mathsf{Sup}(\varphi))$ denote the conjunction of i independent identically distributed random XOR clauses over the variables in $\mathsf{Sup}(\varphi)$. As in [17], each XOR clause is obtained by randomly selecting a subset of $\mathsf{Sup}(\varphi)$, taking the XOR of the selected variables, and equating the resulting formula to either 1 or 0 chosen at random. Each such XOR clause maps an assignment of $\mathsf{Sup}(\varphi)$ to either 1 or 0. Hence, h_i can also be viewed as a hash function mapping $\{1,0\}^{|\mathsf{Sup}(\varphi)|}$ to $\{1,0\}^i$. Since each XOR clause in h_i is chosen randomly and independently, h_i randomly partitions the solution space of φ into 2^i cells, and the solutions of $\varphi \wedge h_i(\mathsf{Sup}(\varphi))$ comprise one such cell. We were able to show that the size of an arbitrary chosen cell induced by the above partitioning is not only an unbiased estimator of $|\mathsf{Sol}(\varphi)|/2^i$, but also crucially has small variance. By choosing i appropriately, the cells can indeed be made "small" and similar enough. Once this is achieved, the problem of almost uniform sampling and approximate model counting then reduces to uniform sampling and exact counting in an arbitrarily chosen "small" cell. Our theoretical analysis showed that the threshold for deciding whether a cell is "small" enough depended on $1/\varepsilon^2$, and could be statically pre-computed. This analysis crucially used universality properties of XOR based hashing. In the case of approximate model counting, the above method suffices to give an approximate count within the specified tolerance of the actual count, but with a fixed confidence. To amplify the confidence to the user-specified level, standard probability amplification arguments can be used by repeating the entire procedure sufficiently many times. The count of such repetitions was also pre-computed and shown to depend on $\log_2(1/\delta)$.

The only parameter that could not be statically pre-computed in the above approach was the right value of i to be used in $h_i(\mathsf{Sup}(\varphi))$. Our theoretical analysis showed that this value ought to be $\log_2 |\mathsf{Sol}(\varphi)|$ in expectation. However, since we didn't know $|\mathsf{Sol}(\varphi)|$ to begin with, this couldn't be pre-computed. Instead, we decided to find the right value of i iteratively. This involved iteratively increasing i from 1 to $|\mathsf{Sup}(\varphi)|$ and checking at each step whether the count of solutions of $\varphi \wedge h_i(\mathsf{Sup}(\varphi))$ was below the pre-computed threshold, say thresh, for a cell to be considered "small". Note that this required making no more than thresh invocations of CryptoMiniSat for each value of i, since we could abort the check and increment i once thresh solutions of $\varphi \wedge h_i(\mathsf{Sup}(\varphi))$ had been found. Thus, our method resulted in $\mathsf{thresh} \cdot \log_2 |\mathsf{Sol}(\varphi)|$ invocations of CryptoMiniSat in expectation, where each invocation involved the formula $\varphi \wedge h_i(\mathsf{Sup}(\varphi))$. This circumvented the twin problems of blowing up the support of the input formula and invoking CryptoMiniSat prohibitively many times. Needless to say, the worst-case behaviour of our algorithm was going to be bad, since a call to CryptoMiniSat could take exponential time in the worst-case. However, since the observed performance of CryptoMiniSat was far better, we expected our approximate counter and solver to work well in practice.

By late 2012, we had implemented a sampler named UniGen, using the above ideas, and had worked out a proof of it almost uniform guarantee. UniGen sig-

nificantly outperformed the best available approximate samplers at that time in terms of the experimentally observed uniformity of samples generated, and did not take prohibitively large time to generate a few samples. In fact, with the highly engineered CryptoMiniSat as our backend engine, we were able to generate samples from the space of solutions of fairly large formulas that had hitherto been beyond the reach of almost uniform samplers with strong approximation guarantees. A paper on UniGen was published in CAV 2013 [8], and was well received by the community. In parallel, we were also working on an approximate model counter using similar ideas. We chose not to use Jerrum et al's reduction of approximate counting to almost uniform sampling or vice versa in designing either the counter or sampler. This was an informed choice motivated by the observation that we were using a real SAT solver (instead of an NP-oracle) in our implementation, and Jerrum et al's reduction would have effectively increased the number of invocations of this solver prohibitively. Our work on the first scalable, fully automatic (i.e. no dependence on user-provided parameters) approximate model counter, called ApproxMC, with provable approximation guarantees was published in CP 2013 [9]. Since we did not use Jerrum et al's reduction, we had to work out the theoretical analyses of UniGen and ApproxMC separately. Not surprisingly, there were commonalities between the analyses of UniGen and ApproxMC. Surprisingly however, there were some key differences as well. In particular, our analysis of UniGen required using 3-universality properties of XOR based hash functions, while the analysis of ApproxMC required only 2-universality properties of the same hash functions. This discrepancy has continued to puzzle us over the years, and it is only recently that we are beginning to understand that it may be possible to re-design UniGen so that only 2-universality properties of XOR-based hash functions are needed to provide the same approximation guarantees.

5 Work post 2013

The year 2013 also saw another paper [16], authored by Ermon et al, that described an independent effort to design a scalable approximate model counter with provable guarantees in a weighted setting. Interestingly, this algorithm reduced to Stockmeyer's algorithm in case a constant weight function was used. While there were commonalities in the technical concerns addressed in our work and that of [16], the techniques used were different.

Since 2013, there has been a fair body of work that has taken the work of [9] forward. Three significant developments are worth mentioning here.

1. ***Use of indepedent supports in XOR-based hashing:*** An *independent support* of a Boolean formula φ is a subset $\mathcal{I}$ of $\mathsf{Sup}(\varphi)$ such that whenever two satisfying assignments π_1 and π_2 of φ agree on $\mathcal{I}$, then $\pi_1 = \pi_2$. Often, the size of an independent support of a formula can be 1-2 orders of magnitude smaller than the size of its support. Therefore, if XOR clauses are constructed using only variables in the independent support of φ, we are likely to have significantly smaller XOR clauses. This is known to improve

the performance of solvers like CryptoMiniSat. In [10], this idea was used to significantly speed up ApproxMC without sacrificing any approximation guarantee. In [18], an algorithmic framework, called MIS, was also proposed to compute an independent support of a given CNF formula automatically, by reducing this problem to the Group-oriented Minimal Unsatisfiable Subformula problem [20, 21].

Since the size of XOR clauses has a significant bearing on the performance of SAT solvers that reason about combinations of CNF and XOR clauses, there has been significant interest in recent times in reducing the size of XOR clauses using sophisticated families of hash functions [26, 3, 2, 1]. The problem is particularly challenging since reducing the size of XOR clauses almost always weakens the approximation guarantees that one can derive for XOR-hashing based approximate counters. This is an extremely active area of research, and any breakthrough here can trigger the next quantum jump in performance of approximate model counters. However, the final word on this is yet to be said.

2. ***Use of dependent XOR clauses in XOR-based hashing:*** While the hash function $h_i(\mathsf{Sup}(\varphi))$ described above was constructed by choosing i XOR constraints independently at random, we have subsequently shown in [11] that there are significant benefits that can be reaped by carefully introducing dependence among the random XOR constraints. Specifically, we introduced the *prefix-family* of dependent XOR clause based hash functions in [11], and showed that this leads to significant reduction in running time of ApproxMC. This improvement can be attributed to two different reasons. First, the prefix family of hash functions makes it possible to use a binary search to determine the right value of i in $h_i(\mathsf{Sup}(\varphi))$. In contrast, with independent hash functions, a linear search was needed to find the right value of i. Second, the prefix family of hash functions makes it possible to use incremental SAT solving when searching for the right value of i. It is well-known that incremental SAT solving can lead to significant improvements in run-time when a sequence of SAT solver calls on related problem instances must be made. ApproxMC directly benefits from this by using the prefix family of hash functions. The flip side of using the prefix family of hash functions is that earlier proofs of approximation guarantees that assumed independence of the clauses were no longer applicable. Therefore we had to re-work the proofs from scratch for the prefix family of hash functions. This required a significant re-think of the analysis, and eventually, we were able to establish exactly the same approximation guarantees that independent XOR clause-based hashing provided.q
3. ***Tighter integration of Gaussian elimination and CDCL reasoning:*** Recently, the capabilities of CryptoMiniSat have been significantly enhanced by very innovative engineering by Soos and Meel [22]. Specifically, Soos and Meel have proposed a scheme of blasting of XOR clauses into CNF clauses, in-processing the resulting ensemble of CNF clauses, recovering (potentially new) XOR clauses from the in-processed CNF clauses, combining CDCL on CNF clauses and Gaussian elimination on the recovered XOR clauses, and

then destroying all recovered XOR clauses before repeating the above steps in each re-start of the CDCL solver. Since checking satisfiability of formulas with mixed CNF and XOR clauses is the single most time consuming step in ApproxMC and UniGen, it is not surprising that this has led to dramatic improvements in the running time of the latest versions of ApproxMC and UniGen.

The combination of the three ideas discussed above, along with other engineering optimizations, has helped push the frontiers of practical approximate model counting with strong guarantees to uncharted territory. In their latest paper [22], Soos and Meel report that the latest version of ApproxMC is able to solve problems with more half a million variables. Thanks to the close connection between approximate model counting and almost uniform sampling, almost all the developments discussed above have positively impacted the performance of UniGen as well. In fact, in the context of almost uniform sampling, we developed a truly parallel version of UniGen and discussed it in [12].

Building on the succes of ApproxMC, we also extended the core idea to the setting of weighted model counting in [6]. In weighted model counting, every assignment of variables in $\mathsf{Sup}(\varphi)$ is assigned a non-negative weight by a weight function, and our goal is to compute the sum of weights of all solutions of a given formula φ. The approach used in [6] required the use of a satisfiability solver that accepts a mix of pseudo-Boolean constraints and XOR constraints as inputs. While the theory works out beautifully, the primary bottleneck in practical deployment turned out to be the availability of practically efficient pseudo-Boolean+XOR solvers. More recently, it has been shown by de Colnet and Meel [14] that weighted model counting can be naturally reduced to either an optimization problem, as has been used in [16], or to the problem of counting models with a specified minimum weight. In an independent line of work, we also showed in [13] that weighted model counting can be reduced all the way to (unweighted) model counting for a specific class of weight functions called *literal weight functions*. This requires encoding the weights of literals using separate gadgets that involve additional variables. While this approach has shown promise in the setting of exact weighted model counting, the introduction of additional variables increases the sizes of XOR clauses when ApproxMC is run on the transformed problem instances. Experiments on a range of benchmarks show that this doesn't always yield better performance vis-a-vis other methods discussed above for solving weighted model counting directly. Finally, in [7], we also extended our work on approximate counting to counting models of quantifier-free SMT formulas in the theory of bit-vectors. This required generalizing XOR-based hash functions to linear modular hash functions with prime moduli to preserve the same approximation guarantees as provided by XOR-based hash functions. The backend engine also had to be correspondingly replaced by an SMT solver like Z3 [15] or CVC4 [4]. Our initial experiments, reported in [7], showed that while this succeeded in solving a decent number of benchmarks, the lack of a tight integration between Gaussian elimination and core SMT reasoning (viz. DPLL(T)) eventually came in the way of scaling up to

large problem instances. We are hopeful that once this integration reaches the level of sophistication as it has in CryptoMiniSat, approximate model counting for bit-vector SMT formulas will also scale to large industry-scale examples.

6 Concluding remarks

It has been a long journey from where we started in the winter of 2011 to where we are today with respect to approximate model counters and almost uniform samplers. The ability to scale to industry-scale problems and also provide rigorous approximation guarantees has encouraged researchers working from diverse domains to use our tools as backend engines in their work. We believe this will enrich the area and will spur further improvements in approximate model counting and sampling. While the bridge between theory and practice has been partly bridged, there is still a lot more that needs to be done. Fortunately, this is an active area of research today, and we hope that further breakthroughs will happen soon enough so that we can break the million variable barrier and go far beyond.
[An expanded version of our original CP 2013 paper, complete with all proofs, can be found at https://arxiv.org/pdf/1306.5726.pdf.]

References

1. Dimitris Achlioptas, Zayd Hammoudeh, and Panos Theodoropoulos. Fast and flexible probabilistic model counting. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 148–164. Springer, 2018.
2. Dimitris Achlioptas and Panos Theodoropoulos. Probabilistic model counting with short xors. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 3–19. Springer, 2017.
3. Megasthenis Asteris and Alexandros G Dimakis. Ldpc codes for discrete integration. Technical report, Technical report, UT Austin, 2016.
4. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovi'c, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, July 2011. Snowbird, Utah.
5. M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of NP-witnesses using an NP-oracle. *Information and Computation*, 163(2):510–526, 2000.
6. S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proc. of AAAI*, pages 1722–1730, 2014.
7. S. Chakraborty, K. S. Meel, R. Mistry, and M. Y. Vardi. Approximate probabilistic inference via word-level counting. In *Proc. of AAAI*, 2016.
8. S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable and nearly uniform generator of SAT witnesses. In *Proc. of CAV*, pages 608–623, 2013.
9. S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable approximate model counter. In *Proc. of CP*, pages 200–216, 2013.

10. S. Chakraborty, K. S. Meel, and M. Y. Vardi. Balancing scalability and uniformity in SAT witness generator. In *Proc. of DAC*, pages 1–6, 2014.
11. 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.
12. Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. On parallel scalable uniform sat witness generation. In *Proc. of TACAS*, pages 304–319, 2015.
13. Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. From weighted to unweighted model counting. In *Proceedings of AAAI*, pages 689–695, 2015.
14. Alexis de Colnet and Kuldeep S. Meel. Dual hashing-based algorithms for discrete integration. In *Proc. of CP*, 10 2019.
15. Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proc. of TACAS*, pages 337–340. Springer, 2008.
16. Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proc. of ICML*, pages 334–342, 2013.
17. C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *Proc. of AAAI*, volume 21, pages 54–61, 2006.
18. Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, pages 1–18, 2015.
19. M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43(2-3):169–188, 1986.
20. Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.
21. Alexander Nadel. Boosting minimal unsatisfiable core extraction. In *Proc. 10th Int. Conf. on Formal Methods in Computer-Aided Design*, pages 221–229, 2010.
22. Mate Soos and Kuldeep S Meel. BIRD: Engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)(1 2019)*, 2019.
23. Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Proc. of SAT*, pages 244–257, 2009.
24. L. Stockmeyer. The complexity of approximate counting. In *Proc. of STOC*, pages 118–126, 1983.
25. W. Wei and B. Selman. A new approach to model counting. In *Proc. of SAT*, pages 2293–2299. Springer, 2005.
26. S. Zhao, S. Chaturapruek, A. Sabharwal, and S. Ermon. Closing the gap between short and long xors for model counting. In *Proc. of AAAI*, 2016.