Towards Real-Time Approximate Counting

Yash Pote[1] ⓡ **Kuldeep S. Meel**[2,3] ⓡ **Jiong Yang**[3*]

[1] National University of Singapore
[2] University of Toronto
[3] Georgia Institute of Technology

Abstract

Model counting is the task of counting the number of satisfying assignments of a Boolean formula. Since counting is intractable in general, most applications use (ε, δ)-approximations, where the output is within a $(1+\varepsilon)$-factor of the count with probability at least $1-\delta$. Many demanding applications make thousands of counting queries, and the state-of-the-art approximate counter, ApproxMC, makes hundreds of calls to SAT solvers to answer a single approximate counting query. The sheer number of SAT calls poses a significant challenge to the existing approaches.

In this work, we propose an approximation scheme, ApproxMC7 that is tailored to such demanding applications with low time limits. Compared to ApproxMC, ApproxMC7 makes $14\times$ fewer SAT calls while providing the same guarantees as ApproxMC in the constant-factor regime. In an evaluation over 2,247 instances, ApproxMC7 solved 271 more instances and achieved a $2\times$ speedup against ApproxMC.

1 Introduction

For a Boolean formula $\varphi : \{0,1\}^n \to \{0,1\}$, the problem of *model counting* is to compute the size of the set $\texttt{sol}(\varphi) = \{\sigma | \varphi(\sigma) = 1\}$. Model counting is fundamental to several domains, including inference on probabilistic graphical models (Chavira and Darwiche 2008; Agrawal, Pote, and Meel 2021), neural network verification (Bhattacharyya et al. 2020), computational biology (Sashittal and El-Kebir 2019), and cryptography (Beck, Zinkus, and Green 2020). In the seminal paper by Valiant (Valiant 1979), the problem of counting was shown to be #P-complete, with #P being the collection of counting problems whose decision counterparts are in NP. Subsequently, Toda demonstrated the complexity-theoretic hardness of this problem by establishing that a single invocation of a #P oracle can solve any problem within the polynomial hierarchy; formally, $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$ (Toda 1989).

Due to the intractability of exact model counting, there is significant interest among researchers and practitioners in the development of counting algorithms that provide probably approximate correct (PAC) guarantees. In PAC model counting, given parameters (ε, δ), one seeks to find a $(1+\varepsilon)$ multiplicative estimate of $|\texttt{sol}(\varphi)|$ with a probability of error at most δ. In 1983, Stockmeyer introduced a random hashing-based procedure that approximated the count with (ε, δ)-guarantees in time polynomial in $1/\varepsilon$, $\log(1/\delta)$, and n (the domain size), when given access to an NP oracle (Stockmeyer 1983). However, practical implementations of Stockmeyer's approach were not feasible at the time due to the lack of solvers capable of effectively handling NP problems (Gomes, Sabharwal, and Selman 2006). In 2013, leveraging the remarkable advancements in SAT solvers, Chakraborty et al. (Chakraborty, Meel, and Vardi 2013) extended Stockmeyer's framework to a scalable (ε, δ)-counting algorithm called ApproxMC. This development opened the door to practical implementations of hashing-based techniques for approximate counting. Since then, there has been a sustained interest in further optimizing these techniques (Ermon et al. 2013a,b; Chakraborty et al. 2014; Ivrii et al. 2016; Meel et al. 2016; Chakraborty et al. 2016; Soos and Meel 2019; Meel and Akshay 2020; Yang and Meel 2021; Yang, Chakraborty, and Meel 2022). The current state-of-the-art in this field is ApproxMC6 (Yang and Meel 2023), which is a variant of ApproxMC augmented with rounding techniques.

Driven by advances in theoretical and practical development, approximate model counting has been deployed in applications (Beck, Zinkus, and Green 2020) where hundreds of model count estimates are required en route to the final answer. This increase in the number of calls, albeit with a relaxed approximation accuracy, still poses a critical challenge to the existing approaches when time availability is constrained.

The approximation scheme, ApproxMC chooses an integer m and splits the solution space into 2^m many *roughly equal small* cells, and uses the number of solutions in a randomly selected cell scaled by 2^m to approximate the exact number of solutions. We use 2-universal hash functions to ensure a roughly *equal* split with low variance, and then we invoke a SAT solver to check the cell size. We consider a cell as small enough to estimate the model count when the SAT solver can only find a number of solutions at most a pre-computed threshold, thresh. We boost the confidence up to $1-\delta$ by repeating the above process independently and returning the

*The authors decided to forgo the old convention of alphabetical ordering of authors in favor of a randomized ordering, denoted by ⓡ.

median of estimates.

ApproxMC involves numerous calls to a SAT solver to check whether the selected cells contain thresh many solutions, where thresh is $\Theta(1/\varepsilon^2)$. For example, given $\varepsilon = 13$, we need up to 23 SAT calls to check whether a cell is small enough for model count estimation, and a total of 336 calls for an instance with $n \approx 13000$. SAT solving is NP-complete, and making hundreds of SAT calls is a major bottleneck to applying ApproxMC in scenarios with a limited time budget. Therefore, a reduction in the number of SAT calls would significantly improve the runtime performance of ApproxMC and extend its application to real-time use cases with short time limits.

The primary contribution of this paper is to propose a novel approximate counting scheme called ApproxMC7[1], realizing a significant reduction in the number of SAT calls in the large-ε regime. Compared to ApproxMC, ApproxMC7 only requires a *single* SAT call to check the cell size, reducing the number of SAT calls from hundreds to a few. For the above instance where ApproxMC makes 336 SAT calls, ApproxMC7 requires only 17 calls while providing the same (ε, δ)-guarantee. In a comprehensive evaluation over 2,247 instances, ApproxMC7 solved 271 more instances and achieved a $2\times$ speedup against ApproxMC. Further, we also study the problem of automating chosen ciphertext attacks, where hundreds of approximate counts are required, and we find that using ApproxMC7 as a subroutine instead of ApproxMC provides a $7\times$ speedup.

Organization. The rest of the paper is organized as follows. Section 2 introduces the notations and preliminaries. Section 3 presents the approximate counting scheme, ApproxMC7 with an accessible proof for the (ε, δ)-guarantee. Section 4 provides an empirical evaluation of ApproxMC7 against ApproxMC, and finally, we conclude in Section 5.

2 Notations and Background

Let $\varphi : \{0,1\}^n \to \{0,1\}$ be a function, where n denotes the number of variables. An assignment σ to the variables of φ is called a satisfying assignment or solution of φ if $\varphi(\sigma) = 1$. We denote the set of all satisfying assignments of φ by $\mathtt{sol}(\varphi) = \{\sigma | \varphi(\sigma) = 1\}$. The model counting problem is to compute the cardinality of the set of satisfying assignments for a given function φ, i.e., $|\mathtt{sol}(\varphi)|$. In the rest of the paper, we will use the short hand probably approximately correct (or PAC) counter to refer to an algorithm that takes as inputs a function φ, an approximation parameter $\varepsilon > 0$, and an error parameter $\delta \in (0,1)$, and returns an (ε, δ)-estimate c, such that

$$\Pr\left[\frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathtt{sol}(\varphi)|\right] \geq 1 - \delta \quad (1)$$

The algorithm proposed in this paper can also be applied to projected model counting, where we compute the cardinality of $\mathtt{sol}(\varphi)$ projected on a subset of variables. For simplicity, our algorithm is only described in the context of model counting, but we indeed consider projected counting benchmarks in empirical evaluation.

[1]The resulting tool ApproxMC7 is available open-source at https://github.com/meelgroup/approxmc

Let $n, m \in \mathbb{N}$ and $\mathcal{H}(n,m) \stackrel{\triangle}{=} \{h : \{0,1\}^n \to \{0,1\}^m\}$ be a family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$. We use $h \stackrel{R}{\leftarrow} \mathcal{H}(n,m)$ to denote the probability space obtained by choosing a function h uniformly at random from $\mathcal{H}(n,m)$. To measure the quality of a hash function, we are interested in the set of elements of $\mathtt{sol}(\varphi)$ mapped to λ by h and use $\mathsf{Cnt}_{\langle\varphi,m\rangle}$ to denote its cardinality. The expected value of Z is denoted $\mathsf{E}[Z]$ and its variance as $\sigma^2[Z]$.

Definition 1. *A family of hash functions $\mathcal{H}(n,m)$ is strongly 2-universal if for all distinct $x, y \in \{0,1\}^n$, and fixed $\lambda \in \{0,1\}^m$, we have that for $h \stackrel{R}{\leftarrow} \mathcal{H}(n,m)$, the following two conditions hold:*

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

where λ represents any specific possible output value from the hash function.

For $h \stackrel{R}{\leftarrow} \mathcal{H}(n,n)$ and $\forall m \in \{1,...,n\}$, the m^{th} prefix-slice of h, denoted $h^{(m)}$, is a map from $\{0,1\}^n$ to $\{0,1\}^m$, such that $h^{(m)}(y)[i] = h(y)[i]$, for all $y \in \{0,1\}^n$ and for all $i \in \{1,...,m\}$. Similarly, the m^{th} prefix-slice of $\lambda \in \{0,1\}^n$, denoted $\lambda^{(m)}$, is an element of $\{0,1\}^m$ such that $\lambda^{(m)}[i] = \lambda[i]$ for all $i \in \{1,...,m\}$.

The following proposition is used in the proof in Section 3. The proof can be found in (Yang and Meel 2023).

Proposition 1. *For every $1 \leq m \leq n$, the following holds:*

$$\mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,m\rangle}\right] = \frac{|\mathtt{sol}(\varphi)|}{2^m} \quad (2)$$
$$\sigma^2\left[\mathsf{Cnt}_{\langle\varphi,m\rangle}\right] \leq \mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,m\rangle}\right] \quad (3)$$

2.1 Approximate Model Counting

The state-of-the-art approximate model counting algorithm ApproxMC follows a hashing-based scheme, where the solution space is partitioned into *small roughly equal* cells by 2-universal hash functions. The solution space is consecutively partitioned until a randomly selected cell contains no more than thresh solutions. Finally, we scale the number of solutions in the cell by the number of cells to estimate the model count.

Algorithm 1 presents the pseudo-code of ApproxMC. During the initialization, thresh is calculated as a function of ε at Line 1 to ensure the final estimate satisfies the error tolerance in Equation 1. The number of repetitions t is pre-computed as a function of δ at Line 2 to guarantee the error probability lowers down to δ in Equation 1. The main loop returns an estimate in every iteration. Inside the loop, a random 2-universal hash function h is sampled at Line 5 and a random cell is selected at Line 6. Following that, a procedure called LogSATSearch (Chakraborty, Meel, and Vardi 2016) is invoked to search the appropriate prefix (m) of h to partition the solution space into 2^m *roughly equal small* cells. During the search, we check the cell size by invoking a SAT solver sequentially to enumerate solutions inside the cell until we

Algorithm 1: ApproxMC$(\varphi, \varepsilon, \delta)$

1: thresh $\leftarrow \left\lceil 9.84 \left(1 + \frac{\varepsilon}{1+\varepsilon}\right)\left(1 + \frac{1}{\varepsilon}\right)^2 \right\rceil$
2: $t \leftarrow \lceil 17 \log_2(3/\delta) \rceil$, $C \leftarrow$ emptyList, iter $\leftarrow 0$
3: **repeat**
4: iter $\leftarrow$ iter $+ 1$
5: Choose h at random from $\mathcal{H}(n, n)$
6: Choose λ at random from $\{0,1\}^n$
7: $m \leftarrow$ LogSATSearch$(\varphi, h, \lambda,$ thresh$)$
8: $\mathsf{Cnt}_{\langle\varphi,m\rangle} \leftarrow$ BoundedCount $\left(\varphi \wedge \left(h^{(m)}\right)^{-1}\left(\lambda^{(m)}\right), \text{thresh}\right)$
9: nSols $\leftarrow (2^m \times \mathsf{Cnt}_{\langle\varphi,m\rangle})$
10: AddToList$(C,$ nSols$)$
11: **until** (iter $\geq t$)
12: finalEstimate $\leftarrow$ FindMedian(C)
13: **return** finalEstimate

can not find more solutions or thresh solutions have been found. As stated in Lemma 1, the total number of SAT calls in ApproxMC depends linearly on the value of thresh.

Lemma 1. *The number of SAT calls for* ApproxMC$(\varphi, \varepsilon, \delta)$ *is* $\mathcal{O}\left(\text{thresh} \cdot \log(n) \cdot \log\left(\frac{1}{\delta}\right)\right)$.

Proof. From the calculation of t, we obtain the number of repetitions is $\mathcal{O}\left(\log\left(\frac{1}{\delta}\right)\right)$. Following LogSATSearch in (Chakraborty, Meel, and Vardi 2016), there are at most $\log(n)$ candidate cells, where we invoke up to thresh SAT calls to check whether the cell contains thresh solutions. □

After we find the appropriate partition and select a cell, the number of solutions in the cell, $\mathsf{Cnt}_{\langle\varphi,m\rangle}$, is counted at Line 8, and an estimate is calculated as the multiplication of the number of cells and $\mathsf{Cnt}_{\langle\varphi,m\rangle}$. Finally, we repeat the estimation loop t times and return the median of estimates as the approximate model count.

We remark that the numerous SAT calls occupy most of the runtime in ApproxMC, where the number of SAT calls depends linearly on the value of thresh as shown in Lemma 1. In this work, we address this issue by proposing a novel algorithm for approximate model counting with thresh equal to one, which leads to a significant reduction in the number of required SAT calls and delivers improved runtime performance.

2.2 Useful Lemmas

We define a function η to be used in Section 3,

$$\eta(t, \lceil t/2 \rceil, p) = \sum_{k=\lceil t/2 \rceil}^{t} \binom{t}{k} p^k (1-p)^{t-k} \tag{4}$$

Let Error_t denote the event that the median of t model-count estimates falls outside $\left[\frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon}, (1+\varepsilon)|\mathtt{sol}(\varphi)|\right]$. We restate the following lemmata from (Yang and Meel 2023), which are used in the proof in Section 3.

Algorithm 2: ApproxMC7$(\varphi, \varepsilon, \delta)$

1: **if** (φ is UNSAT) **then return** 0
2: $t \leftarrow$ ComputeIter(ε, δ)
3: $C \leftarrow$ emptyList, iter $\leftarrow 0$
4: **repeat**
5: iter $\leftarrow$ iter $+ 1$
6: nSols $\leftarrow$ ApproxMC7Core(φ, ε)
7: AddToList$(C,$ nSols$)$
8: **until** (iter $\geq t$)
9: finalEstimate $\leftarrow$ FindMedian(C)
10: **return** finalEstimate

Lemma 2 (Lem. 2 (Yang and Meel 2023))**.** *Assuming t is odd, we have:*

$$\begin{aligned}\Pr[\mathsf{Error}_t] = \eta\left(t, \lceil t/2 \rceil, \Pr\left[c < \frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon}\right]\right) \\ + \eta\left(t, \lceil t/2 \rceil, \Pr[c > (1+\varepsilon)|\mathtt{sol}(\varphi)|]\right)\end{aligned}$$

Lemma 3 (Lem. 3 (Yang and Meel 2023))**.** *Let* $p_{max} = \max\left\{\Pr\left[c < \frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon}\right], \Pr[c > (1+\varepsilon)|\mathtt{sol}(\varphi)|]\right\}$ *and* $p_{max} < 0.5$*, we have*

$$\Pr[\mathsf{Error}_t] \in \Theta\left(t^{-\frac{1}{2}}\left(2\sqrt{p_{max}(1-p_{max})}\right)^t\right)$$

Lemma 4 (Chernoff Bound (Chernoff 1952))**.** *Suppose $X_1, ..., X_n$ are independent Bernoulli random variables. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathsf{E}[X]$. Then for any $\theta \geq 0$,*

$$\Pr[X \geq (1+\theta)\mu] \leq e^{-\frac{\theta^2 \mu}{2+\theta}}$$

3 Real-Time Model Counting

This section introduces an efficient approximate model counting algorithm for $\varepsilon > 1$. We first present the pseudo-code of our algorithm, ApproxMC7 in Section 3.1. Section 3.2 then proves the (ε, δ)-guarantee of ApproxMC7. In contrast to the fifteen-page long proof of the latest ApproxMC (Yang and Meel 2023), ApproxMC7 admits a simple one-page proof. Section 3.3 illustrates the reduction in the number of SAT calls of ApproxMC7 vs. ApproxMC.

3.1 Algorithm

Algorithm 2 describes the pseudo-code of ApproxMC7. ApproxMC7 takes as input a function φ, and two parameters for approximation quality(ε) and confidence(δ), and returns an estimate of $|\mathtt{sol}(\varphi)|$ within tolerance ε and confidence at least $1 - \delta$. Using a single SAT call ApproxMC7 first checks whether the formula is UNSAT, i.e., $|\mathtt{sol}(\varphi)| = 0$, and if so, returns 0 immediately. If not, ApproxMC7 computes the number of iterations t (Algorithm 3). In every iteration, the subroutine ApproxMC7Core (Algorithm 4) is invoked to obtain an estimate, nSols. Finally, the median of all the estimates, finalEstimate is returned as the (ε, δ)-approximation of $|\mathtt{sol}(\varphi)|$.

Algorithm 3: ComputeIter(ε, δ)

1: $t \leftarrow 1, p_\varepsilon \leftarrow \left(\frac{1+\sqrt{1+2(1+\varepsilon)^2}}{2}\right)^{-1}$
2: **while** $(\eta(t, \lceil t/2 \rceil, p_\varepsilon) > \delta/2)$ **do**
3: $\quad t \leftarrow t + 2$;
4: **return** t

In Algorithm 3, ComputeIter calculates the number of iterations for the main loop of ApproxMC7 (Line 4-8 in Algorithm 1) to ensure that finalEstimate satisfies the (ε, δ)-guarantee. The proof of correctness is presented in Section 3.2.

In Algorithm 4, ApproxMC7Core computes an estimate of $|\mathtt{sol}(\varphi)|$. We use a galloping search to find the largest m for 2^m splits when the selected cell still has a solution. If the splits are sufficiently equal, every cell contains one solution in expectation. Therefore, we can use the number of cells, 2^m, to approximate $|\mathtt{sol}(\varphi)|$. The proof of correctness is presented in Section 3.2.

The algorithm first samples a 2-universal hash function h and a vector λ to split the solution space and select cells. As follows, we use the m-th prefix-slice of h, denoted $h^{(m)}$ to iteratively split the solution space into 2^m cells and use the prefix-slice of λ, i.e., $\lambda^{(m)}$ to select a cell. We use the galloping search to find the largest m so that the selected cell still has a solution. We use loIndex to track the largest m when the selected cell is known to have a solution while using hiIndex to record the smallest m when the cell is known to have no solutions. We initialize m to one.

The galloping search consists of two phases, a *doubling phase* and a *binary phase*. The doubling phase aims to find the first exponent, i, where 2^i is the hiIndex while 2^{i-1} is the loIndex, by keeping doubling m at Line 9. The binary phase then performs a binary search on the range $\left[2^{i-1}, 2^i\right]$ to find the final answer. The search terminates at loIndex $+ 1 =$ hiIndex when loIndex stores the largest m with a solution in the selected cell, while loIndex $+ 1$, i.e., hiIndex, is known to have no solutions in the cell. Finally, we return as the estimate of $|\mathtt{sol}(\varphi)|$, the number of cells 2^{loIndex} multiplied by a factor of $\sqrt{\frac{2\alpha}{\beta}}$. The usage of the factor improves the confidence of the estimate, and is formally justified in Section 3.2 along with the (ε, δ)-guarantee.

3.2 Analysis and the Proof of Correctness

Here, we prove the correctness of ApproxMC7, i.e., we show that ApproxMC7 returns an ε-approximation of the model count with probability at least $1 - \delta$ (in Theorem 1). This is then followed by (Lemma 6), the improved upper bound of the query complexity.

To show the correctness of ApproxMC7 we will focus on the core subroutine ApproxMC7Core in Lemma 5, which provides a ε-estimate of the model count with a constant probability δ_0 lower than $1/2$, and then we use the standard median trick to boost the likelihood of success from δ_0 to δ in Theorem 1.

Algorithm 4: ApproxMC7Core(φ, ε)

1: Choose h at random from $\mathcal{H}(n, n)$
2: Choose λ at random from $\{0, 1\}^n$
3: loIndex $\leftarrow 0$, hiIndex $\leftarrow n + 1, m \leftarrow 1$
4: $\beta \leftarrow \left(\frac{1+\sqrt{1+2(1+\varepsilon)^2}}{2}\right), \alpha \leftarrow \beta - 1$
5: **while** (loIndex $+ 1 <$ hiIndex) **do**
6: $\quad$ **if** $\left(\varphi \wedge \left(h^{(m)}\right)^{-1}\left(\lambda^{(m)}\right) \text{ is SAT}\right)$ **then**
7: $\quad\quad$ loIndex $\leftarrow m$
8: $\quad\quad$ **if** $(2 \cdot m <$ hiIndex$)$ **then**
9: $\quad\quad\quad m \leftarrow 2 \cdot m$
10: $\quad\quad$ **else**
11: $\quad\quad\quad m \leftarrow \lfloor (m +$ hiIndex$)/2 \rfloor$
12: $\quad$ **else**
13: $\quad\quad$ hiIndex $\leftarrow m$
14: $\quad\quad m \leftarrow \lfloor ($loIndex $+ m)/2 \rfloor$
15: **return** $\sqrt{\frac{2\alpha}{\beta}} \times 2^{\mathsf{loIndex}}$

Recall that Lemma 2 states the total error probability associated with under-approximation ($c < |\mathtt{sol}(\varphi)|/(1+\varepsilon)$) and over-approximation($c > (1+\varepsilon)|\mathtt{sol}(\varphi)|$). In Lemma 5, for the under-approximation, we simply use Markov's inequality to bound the error probability, and for the over-approximation, we use Cantelli's bound (Cantelli 1928). Thus, we avoid the complicated analysis for different termination events in the latest ApproxMC (Yang and Meel 2023), which required a fifteen-page long proof, and instead end up with a one-page proof.

Lemma 5. *If* ApproxMC7Core(φ, ε) *returns c, then for some* $\alpha_\varepsilon > 1$*, and* $\beta_\varepsilon > 2$ *we have:*

a. $\mathsf{Pr}\left[c < (1+\varepsilon)|\mathtt{sol}(\varphi)|\right] \geq 1 - \frac{1}{\beta_\varepsilon}$,

b. $\mathsf{Pr}\left[c > \frac{1}{1+\varepsilon}|\mathtt{sol}(\varphi)|\right] \geq 1 - \frac{1}{1+\alpha_\varepsilon}$,

(Proof of (a)). Let c denote the estimate of model count (Line 9) returned by ApproxMC7Core. Given ε, ApproxMC7 defines two constants α, β (Line 4)

$$\beta = \frac{1 + \sqrt{1 + 2(1+\varepsilon)^2}}{2} \quad \text{and} \quad \alpha = \beta - 1$$

Note that this choice of α, and β satisfies the following for $\varepsilon > 1$:

$$2\alpha\beta = (1+\varepsilon)^2 \quad \text{and} \quad \alpha > 1, \beta > 2 \tag{5}$$

Now let $i = \lceil \log(|\mathtt{sol}(\varphi)| \cdot \beta) \rceil$, then we have $\mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle}\right] \leq 1/\beta$. Using Markov's inequality, we get the following bound:

$$\mathsf{Pr}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle} \geq 1\right] \leq \mathsf{Pr}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle} \geq \mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle}\right]\beta\right] \leq 1/\beta \tag{6}$$

Therefore, with probability of at least $1 - \frac{1}{\beta}$, we have

$$\begin{aligned} c &\leq \sqrt{\frac{2\alpha}{\beta}} \cdot 2^{\lceil \log(|\mathtt{sol}(\varphi)|\cdot\beta) - 1 \rceil} < \sqrt{\frac{2\alpha}{\beta}} \cdot 2^{\log(|\mathtt{sol}(\varphi)|\cdot\beta)} \\ &= \sqrt{2\alpha\beta}|\mathtt{sol}(\varphi)| = (1+\varepsilon)|\mathtt{sol}(\varphi)| \end{aligned} \tag{7}$$

□

(Proof of (b)). Let $i = \lfloor \log(|\mathtt{sol}(\varphi)|/\alpha) \rfloor$, then we have $\mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle}\right] \geq \alpha$. Using the Cantelli's inequality, we obtain the following bound:

$$\begin{aligned}
&\mathsf{Pr}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle} \leq 0\right] \qquad (8)\\
&= \mathsf{Pr}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle} - \mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle}\right] \leq -\mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle}\right]\right]\\
&\leq \frac{\sigma^2\left[\mathsf{Cnt}_{\langle\varphi,m\rangle}\right]}{\sigma^2\left[\mathsf{Cnt}_{\langle\varphi,m\rangle}\right] + \mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle}\right]^2}\\
&\leq \frac{1}{1+\mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,i\rangle}\right]} \leq \frac{1}{1+\alpha} \qquad (9)
\end{aligned}$$

Therefore, with probability of at least $1 - \frac{1}{1+\alpha}$, we have

$$\begin{aligned}
c &\geq \sqrt{\frac{2\alpha}{\beta}} \cdot 2^{\lfloor \log(|\mathtt{sol}(\varphi)|/\alpha)\rfloor} \geq \sqrt{\frac{2\alpha}{\beta}} \cdot 2^{\log(|\mathtt{sol}(\varphi)|/\alpha)-1}\\
&= \sqrt{\frac{2\alpha}{\beta}} \cdot \frac{|\mathtt{sol}(\varphi)|}{2\alpha} = \frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon} \qquad (10)
\end{aligned}$$

The last equality follows from (5). □

To derive α and β, we follow (5) and Lemma 3 that requires $1/\beta < 1/2$ and $1/(1+\alpha) < 1/2$.

We make a claim that will be needed for the main proof.

Lemma 6. *Algorithm 3 terminates within $\mathcal{O}\left(\frac{\log\frac{1}{\delta}}{(\varepsilon-1)^2}\right)$ steps.*

Proof. Let t be the value returned by Algorithm 3. Suppose $X_1, ..., X_t$ denote iid Bernoulli random variables with probability p_ε of being equal to 1. Let $X = \sum_{i=1}^{t} X_i$, and then $\mathsf{E}[X] = t \cdot p_\varepsilon$, where $p_\varepsilon = \left(\frac{1+\sqrt{1+2(1+\varepsilon)^2}}{2}\right)^{-1}$.

$$\begin{aligned}
&\eta(t, \lceil t/2\rceil, p_\varepsilon) = \sum_{k=\lceil t/2\rceil}^{t} \binom{t}{k} p_\varepsilon^k (1-p_\varepsilon)^{t-k}\\
&= \mathrm{Pr}\left[X \geq \left\lceil \frac{t}{2} \right\rceil\right] \leq \mathrm{Pr}\left[X \geq \frac{t}{2}\right] = \mathrm{Pr}\left[X \geq \frac{1}{2p_\varepsilon} \cdot tp_\varepsilon\right]\\
&\leq \exp\left(-\frac{\left(\frac{1}{2p_\varepsilon} - 1\right)^2 p_\varepsilon t}{1+\frac{1}{2p_\varepsilon}}\right) = \exp\left(-\frac{(2p_\varepsilon-1)^2 t}{2(2p_\varepsilon+1)}\right)
\end{aligned}$$

where the first two equations follow the Equation 4 and the definition of X, respectively. The last inequality follows the Chernoff Bound in Lemma 4. Hence, let

$$\eta(t, \lceil t/2\rceil, p_\varepsilon) \leq \exp\left(-\frac{(2p_\varepsilon-1)^2 t}{2(2p_\varepsilon+1)}\right) \leq \frac{\delta}{2}$$

and we obtain $t \in \mathcal{O}\left(\frac{\log\frac{1}{\delta}}{\left(p_\varepsilon - \frac{1}{2}\right)^2}\right) = \mathcal{O}\left(\frac{\log\frac{1}{\delta}}{(\varepsilon-1)^2}\right)$. □

We finally state and prove the main theorem.

Theorem 1. *For any $\varepsilon > 1$, ApproxMC7(φ,ε,δ) takes in a formula φ, a tolerance parameter ε, and a confidence parameter δ.* ApproxMC7 *returns c such that*

$$\mathsf{Pr}\left[\frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathtt{sol}(\varphi)|\right] \geq 1-\delta$$

ApproxMC7 *makes $\mathcal{O}\left(\frac{\log n \log\frac{1}{\delta}}{(\varepsilon-1)^2}\right)$ many SAT calls.*

Proof. Combining (a) and (b) of Lemma 5, we have $\frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathtt{sol}(\varphi)|$, with an error probability of at most $\frac{1}{1+\alpha}$ for the lower bound and $\frac{1}{\beta}$ for the upper bound. Substituting ε into α, β reveals that the maximum error in the upper and lower bound is

$$\frac{1}{1+\alpha} = \frac{1}{\beta} = \left(\frac{1+\sqrt{1+2(1+\varepsilon)^2}}{2}\right)^{-1} < \frac{1}{2}$$

where the last inequality follows from the fact that $\varepsilon > 1$.

Let Error_t denote the event that the median of t estimates falls outside $\left[\frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon}, (1+\varepsilon)|\mathtt{sol}(\varphi)|\right]$. We have

$$\mathsf{Pr}\left[\mathsf{Error}_t\right] = 2 \cdot \eta(t, \lceil t/2\rceil, p_\varepsilon) \leq \delta$$
$$\text{for } p_\varepsilon = \left(\frac{1+\sqrt{1+2(1+\varepsilon)^2}}{2}\right)^{-1}$$

where the equation follows Lemma 2 and the inequality follows Algorithm 3.

We now analyze the number of SAT calls made by ApproxMC7. First, we note that ApproxMC7Core performs a doubling search over n elements and takes a SAT call per step, making $\log n$ SAT calls in total. Following Lemma 6, ApproxMC7Core is invoked $\mathcal{O}\left(\frac{\log\frac{1}{\delta}}{(\varepsilon-1)^2}\right)$ times in the main loop of ApproxMC7. Hence, ApproxMC7 makes $\mathcal{O}\left(\frac{\log n \log\frac{1}{\delta}}{(\varepsilon-1)^2}\right)$ many SAT calls. □

3.3 The Reduction in the Number of SAT Calls

We illustrate the ApproxMC7's reduction in the number of SAT calls by a heatmap. Figure 1 presents the heatmap for the ratio of the number of SAT calls of ApproxMC6 to ApproxMC7. The x-axis indicates the tolerance parameter (ε) for a $(1+\varepsilon)$-factor approximation. The y-axis represents the confidence parameter (δ) for a confidence of $1-\delta$. The value in a cell indicates the ratio of the number of SAT calls of ApproxMC6 to ApproxMC7. For example, when $\varepsilon = 13$ and $\delta = 0.001$, ApproxMC6 invokes $10\times$ SAT calls than ApproxMC7. Overall, ApproxMC7 needs fewer SAT calls than ApproxMC6, and the reduction improves up to 18-fold as ε increases, reducing the number of SAT calls by an order of magnitude.

4 Experimental Evaluation

To investigate whether reducing the number of SAT calls can improve runtime performance, we implemented a prototype of ApproxMC7 and compared it to the state-of-the-art approximate model counter, ApproxMC6 (Yang and Meel 2023). We evaluated the runtime performance of ApproxMC7 and ApproxMC6 over a comprehensive set of 2,247 instances (Yang, Pote, and Meel 2024) ranging from various applications such as quantitative verification of neural networks (Baluta et al. 2019), Model Counting Competitions 2020-2022 (Fichte, Hecher, and Hamiti 2021; Hecher and Fichte 2021, 2022), program synthesis (Alur et al. 2013), quantitative control improvisation (Gittis, Vin, and Fremont

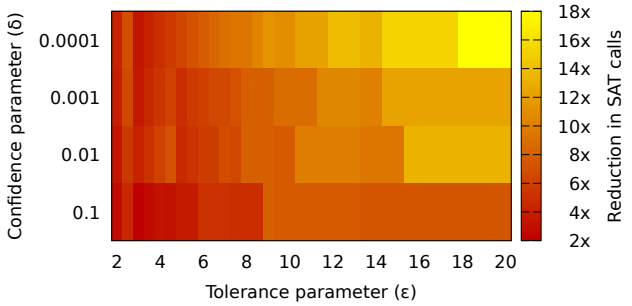

Figure 1: Heatmap for the ratio of the number of SAT calls of ApproxMC6 to ApproxMC7.

2022), quantification of software properties (Teuber and Weigl 2021), and adaptive chosen ciphertext attacks (Beck, Zinkus, and Green 2020). We omit trivial instances that ApproxMC6 could solve within one second from the benchmarks.

We conducted our experiments on a high-performance compute cluster, with each node consisting of AMD EPYC-Milan processor featuring 2×64 real cores and 512GB of RAM. We consider running a counter to estimate the model count of a function as a job. We ran each job on a single core with a 100-second time limit and 4GB memory. We evaluated the runtime performance using three metrics: the number of instances solved, the PAR-2 score[2], and speedup. The speedup measures the ratio of ApproxMC6 runtime to ApproxMC7 runtime, and we report the geometrical mean of speedup over instances solved by both counters. In our experiments, we set $\delta = 0.2$ and $\varepsilon = 13$ and used an efficient pre-processor Arjun (Soos and Meel 2022) to simplify the function.[3]

In particular, we aim to answer the following questions:

RQ1 Is ApproxMC7 faster than ApproxMC6?

RQ2 Does ApproxMC7's estimate satisfy the (ε, δ)-guarantee in practice?

RQ3 Does ApproxMC7 outperform ApproxMC6 in counting-intensive applications?

In summary, ApproxMC7 solved 271 more instances, reduced the PAR-2 score by 24, and achieved a $2\times$ speedup against ApproxMC6. The estimates from ApproxMC7 satisfy the (ε, δ)-guarantee with only 6% of cases (<20%) being outliers and an observed error of 1.59 (<13). In a counting-intensive application, ApproxMC7 outperformed ApproxMC6 with a $7\times$ speedup.

4.1 RQ1. Runtime Performance

We present a detailed comparison of runtime performance in Table 1. The first row indicates the number of instances solved by a counter within one hundred seconds. ApproxMC6 solved 801 of 2,247 instances whereas ApproxMC7 solved 1,072 instances, i.e., 271 more than ApproxMC6. The second row compares the PAR-2 score, where ApproxMC7 scores 115, an improvement of 24 over ApproxMC6. The last row highlights the speedup of ApproxMC7 over ApproxMC6. We report the geometrical mean speedup over the instances solved by both ApproxMC6 and ApproxMC7. Overall, ApproxMC7 delivered a $2.1\times$ speedup over ApproxMC6, with the maximum individual speedup of up to 70.

Table 2 illustrates the comparison on a randomly sampled subset of instances, where ApproxMC7 is generally faster and used $10\times$ fewer SAT calls than ApproxMC6. For the instances unsolved by ApproxMC6, we count the SAT calls by timeout.

	ApproxMC6	ApproxMC7
#Solved	801	**1072**
PAR-2 score	139	**115**
Speedup	—	2.1

Table 1: The number of solved instances, PAR-2 score, and speedup for ApproxMC7 vs. ApproxMC6 over 2,247 instances.

We present a CDF (cumulative distribution function) plot of counting times in Figure 2. The x-axis indicates the runtime in seconds, while the y-axis represents the number of solved instances so far. A point (x, y) indicates that y instances were solved within x seconds. For example, when $x = 40$, ApproxMC6 solved less than 600 instances within 40 seconds whereas ApproxMC7 solved more than 800 instances. As indicated by the gap between the two curves, ApproxMC7 has an early lead of over a hundred instances over ApproxMC6.

4.2 RQ2. Approximation Quality

This section evaluates the empirical approximation quality for ApproxMC7. We used the exact model counter Ganak (Sharma et al. 2019) to compute the exact count and compared it to the estimate of ApproxMC7. We collected results over 698 instances solved by both Ganak and ApproxMC7. Figure 3 compares the exact count and ApproxMC7's estimate over a subset of instances. The x-axis indicates the index of instances in ascending order of model count, while the y-axis shows the model count on a log scale. Following the (ε, δ)-guarantee in Equation 1 for $\varepsilon = 13$ and $\delta = 0.2$, the estimate from ApproxMC7 should fall into the range of $[|\texttt{sol}(\varphi)|/14, |\texttt{sol}(\varphi)| \cdot 14]$ with probability of 0.8, where $|\texttt{sol}(\varphi)|$ denotes the exact model count of a formula φ. In Figure 3, we use the curves $y = |\texttt{sol}(\varphi)|/14$ and $y = |\texttt{sol}(\varphi)| \cdot 14$ to represent the lower and upper bounds of the range, respectively.

In summary, ApproxMC7's estimates satisfied the (ε, δ)-guarantee. Figure 3 shows that the estimate falls into the range $[|\texttt{sol}(\varphi)|/14, |\texttt{sol}(\varphi)| \cdot 14]$ for most instances. Overall, 6% of the estimates (40 out of 698 instances) are outside of the range, which is smaller than the guaranteed probability

[2]The penalized average runtime (or PAR-2) score, extensively used in the SAT competition (Froleyks et al. 2021), is the sum of runtimes for all solved instances + twice the time limit (in our case, 200 seconds) for each unsolved instance.

[3]Arjun was not used with neural network benchmarks as we observed a significant slowdown with its use.

Instance	Runtime(s)		Number of SAT Calls	
	ApproxMC6	ApproxMC7	ApproxMC6	ApproxMC7
track3-195.mcc2020-pcnf	1.54	0.88	334	15
1-target-4-100-bnn-50-20-e-10-trojan-l-4	Unsolved	33.03	>128	11
mc2021-track1-144.cnf	Unsolved	3.9	>171	9
track1-148.mcc2020-cnf	1.91	0.82	331	16
track1-119.mcc2020-cnf	8.21	6.65	226	10
gsv2008.c-01-counterSharp-amh.dimacs	1.5	1.49	11	2
mc2022-track1-099.cnf	17.06	14.53	132	10
adv-3-bnn-100-50-e-5-robustness-p-2-id-20	77.48	28.62	83	5
bwd-loop7.c-01-counterSharp-ash.dimacs	2.08	1.79	11	2
adv-bnn-100-50-e-1-robustness-p-2-id-24	Unsolved	54.21	>32	3

Table 2: The runtime and number of SAT calls on sampled instances for ApproxMC7 vs. ApproxMC6.

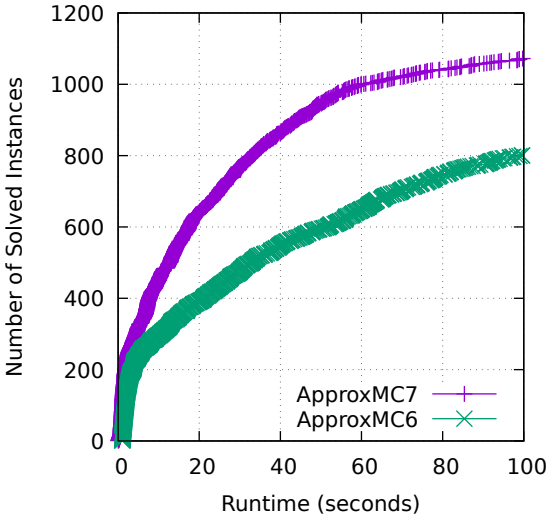

Figure 2: Comparison of counting times for ApproxMC7 and ApproxMC6.

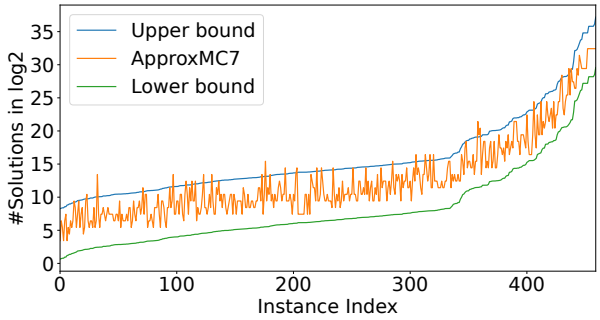

Figure 3: Comparison of approximate counts from ApproxMC7 with exact counts.

$\delta = 0.2$. The geometrical mean of the observed error is 1.59, which is calculated as the relative difference between the approximate and exact counts, subtracted by one, i.e., $\max\{c \cdot |\texttt{sol}(\varphi)|, |\texttt{sol}(\varphi)|/c\} - 1$. The empirical error of 1.59 is much smaller than the guaranteed error tolerance of $\varepsilon = 13$.

4.3 RQ3. Case Study: Automating Chosen Ciphertext Attacks

We conducted a case study on the runtime performance of automating chosen ciphertext attacks, where hundreds of approximate model counting calls are requested at one time. We asked the authors of (Beck, Zinkus, and Green 2020) for the CNF formulae generated in their approach and obtained 101 instances to evaluate the runtime performance of ApproxMC7. Figure 4 presents a scatter plot to compare the runtime between ApproxMC6 and ApproxMC7 for each formula. The x and y axes indicate the runtime in seconds for ApproxMC6 and ApproxMC7, respectively. The diagonal gray line highlights the points of equal runtime. A point below the gray line indicates an instance where ApproxMC6 used more time than ApproxMC7, and vice versa. Therefore, ApproxMC7 was faster than ApproxMC6 on all instances. Overall, ApproxMC7 achieved a 7× speedup against ApproxMC6.

5 Conclusion

In many applications of model counting, a quick and reliable estimate of the count is sufficient. State-of-the-art approximate counting frameworks focus on the development of close-to-exact algorithms, sacrificing runtime, and requiring excessive SAT solver invocations. In several practical use cases, execution time is important and only a coarse estimate is needed; our paper focuses on these scenarios. In the paper, we presented ApproxMC7, a model counter that shows significant improvements in the low-accuracy regime. The main takeaway from our paper is that with a tight analysis, a single SAT call is enough to provide a fairly good estimate of the count. Furthermore, the proof of correctness of ApproxMC7 is much simpler and hence is more easily adapted to other use cases. Our experimental evaluation revealed that ApproxMC7 is significantly faster than the state-of-the-art approximate counter ApproxMC while offering the same guarantees.

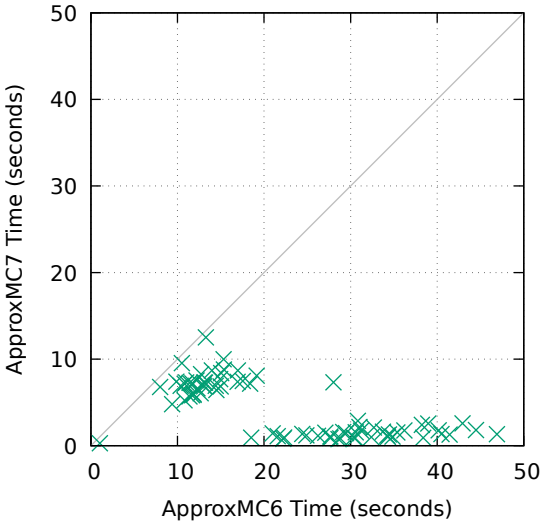

Figure 4: Instance-wise comparison in a case study of automating chosen ciphertext attacks.

Acknowledgments

This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004], Ministry of Education Singapore Tier 2 Grant [MOE-T2EP20121-0011], Ministry of Education Singapore Tier 1 Grant [R-252-000-B59-114], and Natural Sciences and Engineering Research Council of Canada (NSERC) [RGPIN-2024-05956]. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore https://www.nscc.sg.

References

Agrawal, D.; Pote, Y.; and Meel, K. S. 2021. Partition Function Estimation: A Quantitative Study. In *Proc. of IJCAI*.

Alur, R.; Bodik, R.; Juniwal, G.; Martin, M. M. K.; Raghothaman, M.; Seshia, S. A.; Singh, R.; Solar-Lezama, A.; Torlak, E.; and Udupa, A. 2013. Syntax-guided synthesis. In *Proc. of FMCAD*.

Baluta, T.; Shen, S.; Shinde, S.; Meel, K. S.; and Saxena, P. 2019. Quantitative Verification of Neural Networks and Its Security Applications. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19. New York, NY, USA: Association for Computing Machinery. ISBN 9781450367479.

Beck, G.; Zinkus, M.; and Green, M. 2020. Automating the Development of Chosen Ciphertext Attacks. In *Proc. of USENIX Security*.

Bhattacharyya, A.; Gayen, S.; Meel, K. S.; and Vinodchandran, N. V. 2020. Efficient Distance Approximation for Structured High-Dimensional Distributions via Learning. *ArXiv*, abs/2002.05378.

Cantelli, F. P. 1928. Sui confini della probabilita. In *Atti del Congresso Internazional del Matematici*.

Chakraborty, S.; Fremont, D. J.; Meel, K. S.; Seshia, S. A.; and Vardi, M. Y. 2014. Distribution-Aware Sampling and Weighted Model Counting for SAT. In *Proc. of AAAI*.

Chakraborty, S.; Meel, K. S.; Mistry, R.; and Vardi, M. Y. 2016. Approximate Probabilistic Inference via Word-Level Counting. In *Proc. of AAAI*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013. A scalable approximate model counter. In *Principles and Practice of Constraint Programming: 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings 19*, 200–216. Springer.

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

Chavira, M.; and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7): 772–799.

Chernoff, H. 1952. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations. *The Annals of Mathematical Statistics*.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013a. Embed and Project: Discrete Sampling with Universal Hashing. In *Proc. of NeurIPS*.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013b. Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization. In *Proc. of ICML*.

Fichte, J. K.; Hecher, M.; and Hamiti, F. 2021. The Model Counting Competition 2020. *ACM J. Exp. Algorithmics*.

Froleyks, N.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M. 2021. SAT Competition 2020. *Artificial Intelligence*, 301: 103572.

Gittis, A.; Vin, E.; and Fremont, D. J. 2022. Randomized Synthesis for Diversity and Cost Constraints with Control Improvisation. In *Proc. of CAV*.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Model Counting: A New Strategy for Obtaining Good Bounds. In *Proc. of AAAI*.

Hecher, M.; and Fichte, J. K. 2021. Model counting competition 2021.

Hecher, M.; and Fichte, J. K. 2022. Model counting competition 2022.

Ivrii, A.; Malik, S.; Meel, K. S.; and Vardi, M. Y. 2016. On Computing Minimal Independent Support and Its Applications to Sampling and Counting. *Constraints*.

Meel, K. S.; and Akshay, S. 2020. Sparse Hashing for Scalable Approximate Model Counting: Theory and Practice. In *Proc. of LICS*.

Meel, K. S.; Vardi, M. Y.; Chakraborty, S.; Fremont, D. J.; Seshia, S. A.; Fried, D.; Ivrii, A.; and Malik, S. 2016. Constrained Sampling and Counting: Universal Hashing meets SAT Solving. In *Proc. of Workshop on Beyond NP(BNP)*.

Sashittal, P.; and El-Kebir, M. 2019. SharpTNI: Counting and sampling parsimonious transmission networks under a weak bottleneck. *bioRxiv*.

Sharma, S.; Roy, S.; Soos, M.; and Meel, K. S. 2019. GANAK: A Scalable Probabilistic Exact Model Counter. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Soos, M.; and Meel, K. S. 2019. BIRD: Engineering an Efficient CNF-XOR SAT Solver and its Applications to Approximate Model Counting. In *Proc. of AAAI*.

Soos, M.; and Meel, K. S. 2022. Arjun: An Efficient Independent Support Computation Technique and its Applications to Counting and Sampling. In *Proc. of ICCAD*.

Stockmeyer, L. 1983. The Complexity of Approximate Counting. In *Proc. of STOC*.

Teuber, S.; and Weigl, A. 2021. Quantifying Software Reliability via Model-Counting. In *Proc. of QEST*.

Toda, S. 1989. On the computational power of PP and (+)P. In *Proc. of FOCS*.

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

Yang, J.; Chakraborty, S.; and Meel, K. S. 2022. Projected Model Counting: Beyond Independent Support. In *Proc. of ATVA*.

Yang, J.; and Meel, K. S. 2021. Engineering an Efficient PB-XOR Solver. In *Proc. of CP*.

Yang, J.; and Meel, K. S. 2023. Rounding Meets Approximate Model Counting. In *Proc. of CAV*.

Yang, J.; Pote, Y.; and Meel, K. S. 2024. Benchmark used for AAAI25 paper: Towards Real-Time Approximate Counting. https://doi.org/10.5281/zenodo.14533501.