

# Auditable Algorithms for Approximate Model Counting<sup>\*†</sup>

Kuldeep S. Meel<sup>1</sup>  Supratik Chakraborty<sup>2</sup>  S. Akshay<sup>2</sup>

<sup>1</sup> University of Toronto, Canada

<sup>2</sup> Indian Institute of Technology Bombay, Mumbai, India


## Abstract

Model counting, or counting the satisfying assignments of a Boolean formula, is a fundamental problem with diverse applications. Given  $\#P$ -hardness of the problem, developing algorithms for approximate counting is an important research area. Building on the practical success of SAT-solvers, the focus has recently shifted from theory to practical implementations of approximate counting algorithms. This has brought to focus new challenges, such as the design of auditable approximate counters that not only provide an approximation of the model count, but also a certificate that a verifier with limited computational power can use to check if the count is indeed within the promised bounds of approximation.

Towards generating certificates, we start by examining the best-known deterministic approximate counting algorithm due to Stockmeyer that uses polynomially many calls to a  $\Sigma_2^P$  oracle. We show that this can be audited via a  $\Sigma_2^P$  oracle with the query constructed over  $n^2 \log^2 n$  variables, where the original formula has  $n$  variables. Since  $n$  is often large, we ask if the count of variables in the certificate can be reduced – a crucial question for potential implementation. We show that this is indeed possible with a tradeoff in the counting algorithm’s complexity. Specifically, we develop new deterministic approximate counting algorithms that invoke a  $\Sigma_3^P$  oracle, but can be certified using a  $\Sigma_2^P$  oracle using certificates on far fewer variables: our final algorithm uses only  $n \log n$  variables. Our study demonstrates that one can simplify auditing significantly if we allow the counting algorithm to access a slightly more powerful oracle. This shows for the first time how audit complexity can be traded for complexity of approximate counting.

## 1 Introduction

Trust in computation done by third-party software is increasingly becoming a concern in today’s world. As an example,

<sup>\*</sup>The authors decided to forgo the old convention of alphabetical ordering of authors in favor of a randomized ordering, denoted by . The publicly verifiable record of the randomization is available at <https://www.aeaweb.org/journals/policies/random-author-order/search>

<sup>†</sup>The full version of the paper is available at <https://arxiv.org/abs/2312.12362>

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

consider a software implementing an algorithm for which we already have a rigorous proof of correctness. Even when the software is implemented by trusted and expert programmers, one cannot be sure that the implementation will never produce an erroneous result. Insidious bugs may go undetected for several reasons: for example, the implementation may be too complex to have been tested comprehensively, or it may use code from untrusted libraries that have bugs, or the programmer may have inadvertently introduced a bug. In such a scenario, we wish to ask: *How can we provide assurance that an untrusted implementation of an algorithm has produced a correct result for a given problem instance?*

One way to answer the above question is to formally model the semantics of every instruction in the implementation in a theorem proving system and then prove that the composition of these instructions always produces the correct result. This is a challenging and skill-intensive exercise for anything but the simplest of software, but once completed, it gives very high assurance about the correctness of results produced by the implementation *for all instances of the problem being solved*. A relatively less daunting alternative is to have the algorithm (and its implementation) output for each problem instance, not only the result but also a *certificate* of correctness of the result. The certificate must be independently checkable by an auditor, and a successful audit must yield a proof of correctness of the computed result *for the given problem instance*. We call such algorithms *auditable algorithms*. Note that a successful audit for one problem instance does not prove bug-freeness of the implementation; it only shows that the result computed for this specific problem instance is correct. A key requirement for effective use of auditable algorithms is access to an independent trusted certificate auditor. Fortunately, a certificate auditor is usually much simpler to design, and its implementation far easier to prove correct than the implementation of the original algorithm. Therefore, auditable algorithms provide a pragmatic solution to the problem of assuring correctness for untrusted implementations of algorithms. Not surprisingly, auditable algorithms have been studied earlier in the context of different problems (McConnell et al. 2011; Heule, Hunt, and Wetzler 2013; Wetzler, Heule, and Hunt 2014; Gocht et al. 2022; Ekici et al. 2017; Barthe et al.

2006; Necula and Lee 1998; Magron et al. 2015; Cheung and Moazzez 2016).

In this paper, we present the first detailed study of auditable algorithms for model counting with provable approximation guarantees— an important problem with diverse applications. Our study reveals a nuanced landscape where the complexity of an auditable algorithm can be “traded off” for suitable measures of complexity of certificate auditing. This opens up the possibility of treating certificate audit complexity as a first-class concern in designing auditable algorithms.

Before we delve into further details, a brief background on model counting is useful. Formally, given a system of constraints  $\varphi$ , model counting requires us to count the solutions (or models) of  $\varphi$ . This problem is known to be computationally hard (Valiant 1979), and consequently, significant effort has therefore been invested over the years in designing approximate model counters. Over the last decade, riding on spectacular advances in propositional satisfiability (SAT) solving techniques (see (Biere et al. 2021) for details), approximate model counting technology has matured to the point where we can count solutions of propositional constraints with several thousands of variables within a couple of hours on a standard laptop, while providing strong approximation guarantees (Soos, Gocht, and Meel 2020). This has led to a surge in the number of tools from different application domains that have started using approximate model counters as black-box engines (Klebanov 2014; Teuber and Weigl 2021; Beck, Zinkus, and Green 2020; Zinkus, Cao, and Green 2023) to solve various problems. With increasing such usage comes increasing demands of assurance on the correctness of counts computed by approximate counters. Auditable approximate counting offers a promising approach to solve this problem.



Approximate model counting algorithms come in two different flavours, viz. deterministic and randomized. While randomized counting algorithms with probably approximately correct (PAC) guarantees (Valiant 1984a,b) have been found to scale better in practice, it is not possible to provide a meaningful certificate of correctness for the result computed by a single run of a randomized algorithm (see (McConnell et al. 2011) for a nice exposition on this topic). Therefore, we focus on deterministic algorithms for approximate model counting. Specifically, we ask: (a) what would serve as a meaningful certificate for the best-known deterministic approximate counting algorithm (Stockmeyer 1983)? (b) what is the fine-grained complexity of auditing such a certificate? (c) can we re-design the approximate counting algorithm so as to make certificate auditing more efficient? and (d) is there a tradeoff in the complexity of an auditable deterministic approximate counting algorithm and the fine-grained complexity of certificate auditing? Our study shows that there is a nuanced interplay between the complexity of auditable approximate counting algorithms and that of auditing the certificates generated by such algorithms. Therefore, if efficiency of certificate auditing is treated as a first-class concern, we must re-think the design of auditable algorithms for approximate counting. Indeed, we present two new auditable approximate counting algorithms that significantly improve the efficiency of certificate

auditing at the cost of making the counting algorithms themselves a bit less efficient.

The primary contributions of the paper can be summarized as listed below. In the following, we wish to count models of a propositional formula on  $n$  variables, upto a constant approximation factor.

1. We first show that the best-known deterministic approximate counting algorithm (from (Stockmeyer 1983)) can be converted to an auditable algorithm that generates a certificate over  $\mathcal{O}(n^2 \log^2 n)$  variables. The resulting counting algorithm uses  $\mathcal{O}(n \log n)$  invocations of a  $\Sigma_2^P$  oracle, as in Stockmeyer’s original algorithm. An auditor, however, needs only an invocation of a  $\Sigma_2^P$  oracle on formulas constructed over  $\mathcal{O}(n^2 \log^2 n)$  variables.
2. Next, we develop an deterministic approximate counting algorithm for which the certificate uses only  $\mathcal{O}(n^2)$  variables. This reduction is achieved by allowing the counting algorithm access to a more powerful  $\Sigma_3^P$  (instead of a  $\Sigma_2^P$ ) oracle, that is invoked only  $\mathcal{O}(n)$  times. Certificate auditing in this case requires a single invocation of a  $\Sigma_2^P$  oracle on a formula constructed over  $\mathcal{O}(n^2)$  variables.
3. Finally, we present another deterministic approximate counting algorithm that requires a certificate over only  $\mathcal{O}(n \log n)$  variables. This makes certificate auditing significantly more efficient compared to the cases above, when viewed through the lens of fine-grained complexity. This improvement is obtained by allowing the counting algorithm to invoke a  $\Sigma_3^P$  oracle  $\mathcal{O}(n \log n)$  times. Certificate auditing now requires one invocation of a  $\Sigma_2^P$  oracle on formulas over  $\mathcal{O}(n \log n)$  variables.

Our results show that the fine-grained complexity of certificate checking can be “traded off” significantly against the power of oracles accessible to a deterministic approximate counting algorithm. This opens up the possibility of designing new counting algorithms that are cognizant of certificate checking complexity. Our proofs involve reasoning about specially constructed formulas with quantifier alternations, with incomplete information about parts of the formula. This is quite challenging in general, and one of our technical novelties lies in our application of the probabilistic method in the proofs.

The remainder of the paper is organized as follows. We present some preliminaries and notation in Section 2, and formalize the notion of an audit for approximate counting algorithms. In Section 3, we discuss how Stockmeyer’s algorithm can be turned into an auditable algorithm, and also present the corresponding audit algorithm. Sections 4 and 5 discuss the design of two new auditable algorithms for approximate counting with successively small certificate sizes, and the complexity of auditing these certificates. Finally, we conclude with a discussion of our findings in Section 6. Proof details, omitted due to space constraints, can be found in a full version of the paper available at (Meel  Chakraborty  Akshay 2023).

## 2 Preliminaries

Let  $F$  be a Boolean formula in conjunctive normal form (CNF), and let  $\text{Vars}(F)$ , also called its *support*, be the set of

variables appearing in  $F$ . An assignment  $\sigma$  of truth values to the variables in  $\text{Vars}(F)$  is called a *satisfying assignment* or *witness* of  $F$  if it makes  $F$  evaluate to true. We denote the set of all witnesses of  $F$  by  $\text{sol}(F)$  and its count by  $|\text{sol}(F)|$ . Throughout the paper, we use  $n$  to denote  $|\text{Vars}(F)|$ . We also use bold-faced letters to represent vectors of Boolean variables, when there is no confusion.

We write  $\Pr[\mathcal{Z} : \Omega]$  to denote the probability of outcome  $\mathcal{Z}$  when sampling from a probability space  $\Omega$ . For brevity, we omit  $\Omega$  when it is clear from the context. The expected value of  $\mathcal{Z}$  is denoted  $\mathbb{E}[\mathcal{Z}]$  and its variance is denoted  $\sigma^2[\mathcal{Z}]$ . The quantity  $\frac{\sigma^2[\mathcal{Z}]}{\mathbb{E}[\mathcal{Z}]}$  is called the dispersion index of the random variable  $\mathcal{Z}$ . Given a distribution  $\mathcal{D}$ , we use  $\mathcal{Z} \sim \mathcal{D}$  to denote that  $\mathcal{Z}$  is sampled from the distribution  $\mathcal{D}$ .

We introduce some notation from complexity theory for later use. Let  $\mathcal{C}$  be a decision complexity class. An *oracle* for  $\mathcal{C}$  is a (possibly hypothetical) computational device that when queried with any problem in  $\mathcal{C}$ , answers it correctly in one time unit. If  $\mathcal{C}_1$  and  $\mathcal{C}_2$  are two complexity classes, we use  $\mathcal{C}_1^{\mathcal{C}_2}$  to denote the class of problems solvable by an algorithm for a problem in  $\mathcal{C}_1$  with access to an oracle for  $\mathcal{C}_2$ . The complexity classes P, NP and CoNP are well-known. The classes  $\Sigma_2^P$  and  $\Sigma_3^P$  are defined as  $\text{NP}^{\text{NP}}$  and  $\text{NP}^{\Sigma_2^P}$  respectively.

**Deterministic Approximate Counting (DAC):** Given a Boolean formula  $F$  in CNF and an approximation factor  $k (> 1)$ , *deterministic approximate counting* (or DAC for short), requires us to compute an estimate  $\text{CntEst} (\geq 0)$  using a deterministic algorithm such that  $\frac{|\text{sol}(F)|}{k} \leq \text{CntEst} \leq |\text{sol}(F)| \cdot k$ . It has been shown in (Stockmeyer 1983) that DAC is in  $\text{P}^{\Sigma_2^P}$ , the crucial idea in the proof being use of pairwise independent hash functions to partition the space of all assignments. Given a DAC algorithm for any  $k > 1$ , we can apply it on independent copies of  $F$  (i.e., copies of  $F$ , each with a fresh set of variables) conjoined together to obtain  $|\text{sol}(F)|$  to within any approximation factor  $> 1$ . Therefore, we will often be concerned with DAC algorithms for some convenient  $k$ , viz. 8 or 16.

Practical implementations of DAC algorithms must of course use SAT/QBF solvers as proxies for oracles like the  $\Sigma_2^P$  oracle in (Stockmeyer 1983). Therefore, it is important in practice to look at the finer structure of oracle invocations, especially the support sizes and exact quantifier prefix of the formulas fed to the oracles.

**$k$ -wise independent hash family:** For  $n, m \in \mathbb{N}^{\geq 0}$ , let  $\mathcal{H}(n, m)$  denote a family of hash functions from  $\{0, 1\}^n$  to  $\{0, 1\}^m$ . We use  $h \stackrel{R}{\leftarrow} \mathcal{H}(n, m)$  to denote the probability space obtained by choosing  $h$  uniformly at random from  $\mathcal{H}(n, m)$ .

**Definition 2.1.** For  $k \geq 2$ , we say that  $\mathcal{H}(n, m)$  is a family of  $k$ -wise independent hash functions if for every distinct  $\mathbf{y}_1, \dots, \mathbf{y}_k \in \{0, 1\}^n$  and for every (possibly repeated)  $\alpha_1, \dots, \alpha_k \in \{0, 1\}^m$ ,  $\Pr[h(\mathbf{y}_1) = \alpha_1 \wedge \dots \wedge h(\mathbf{y}_k) = \alpha_k] = \left(\frac{1}{2^m}\right)^k$ , where the probability space is  $h \stackrel{R}{\leftarrow} \mathcal{H}(n, m)$ . Such a family is also sometimes denoted

$\mathcal{H}(n, m, k)$  to highlight the degree of independence.

Significantly, every  $h \in \mathcal{H}(n, m, k)$  can be specified by a  $k$ -tuple of coefficients  $(a_1, a_2, \dots, a_k)$  from the finite field  $GF(2^{\max(n, m)})$  and vice versa (Bellare, Goldreich, and Prank 2000). Hence,  $\mathcal{O}(k \cdot \max(m, n))$  bits suffice to represent  $h$ . We leverage this observation to simplify notation and write  $\exists h$  as syntactic sugar for existentially quantifying over the  $\mathcal{O}(k \cdot \max(m, n))$  Boolean variables that represent  $h$ .

For every formula  $F$  on  $n$  variables, and for every  $\alpha \in \{0, 1\}^m$ , define  $\text{Cell}_{\langle F, h, \alpha \rangle}$  to be the (sub-)set of solutions of  $F$  that are mapped to  $\alpha$  by  $h$ . Formally,  $\text{Cell}_{\langle F, h, \alpha \rangle} = \{\sigma \mid \sigma \in \{0, 1\}^n, \sigma \models F \text{ and } h(\sigma) = \alpha\}$ . For convenience, we use  $\text{Cnt}_{\langle F, h, \alpha \rangle}$  to denote the cardinality of  $\text{Cell}_{\langle F, h, \alpha \rangle}$ .

The following result proves useful in our analysis later.

**Proposition 2.2.** (Bellare and Rompel 1994) Let  $t \geq 4$  be an even integer. Let  $Z_1, Z_2, \dots, Z_k$  be  $t$ -wise independent random variables taking values in  $[0, 1]$ . Let  $Z = Z_1 + Z_2 + \dots + Z_k$ ,  $\mu = \mathbb{E}[Z]$ . Then  $\Pr[|Z - \mu| \geq \varepsilon\mu] \leq 8 \cdot \left(\frac{t\mu + t^2}{\varepsilon^2\mu^2}\right)^{t/2}$  for all  $\varepsilon > 0$ .

### 2.1 Audit Complexity

Let  $A$  be an auditable algorithm that solves (or claims to solve) DAC with a known constant-factor approximation, say  $k$ . Let  $\mathcal{K}$  denote the certificate generated by  $A$  for an input formula  $F$  over  $n$  variables. We expect  $\mathcal{K}$  to be a string of Boolean values of size that is a function of  $n$ . We can now define the notion of an auditor for  $A$  as follows.

**Definition 2.3.** An auditor for  $A$  is an algorithm  $B$  that takes as inputs the formula  $F$ , the estimate  $\text{CntEst}$  returned by  $A$ , and the certificate  $\mathcal{K}$ . Algorithm  $B$  uses an oracle of complexity class  $\mathcal{C}$  to certify that  $\text{CntEst}$  lies within the claimed  $k$ -factor approximation of  $|\text{sol}(F)|$ . The audit complexity of algorithm  $A$  is defined to be the tuple  $(\mathcal{C}, r, t)$ , where the auditor can certify the answer computed by  $A$  by making  $t$  calls to an oracle of complexity class  $\mathcal{C}$ , where each call/query is over a formula with at most  $r$  variables.

In our context, motivated by Stockmeyer’s result (Stockmeyer 1983), we restrict  $\mathcal{C}$  to be  $\Sigma_2^P$ , and further fix the number of calls  $t$  to be 1. Hence, we omit  $\mathcal{C}$  and  $t$ , and simply refer to  $r$  as the audit complexity of the algorithm  $A$ .

### 3 Stockmeyer’s Algorithm and Its Audit

In this section, we consider a celebrated DAC algorithm due to Stockmeyer (Stockmeyer 1983), and analyze its audit complexity. Towards this end, we first consider a QBF formula that plays a significant role in Stockmeyer’s algorithm. For a given Boolean formula  $F$  on  $n$  variables, and for  $1 \leq m \leq n$ , consider

$$\varphi_{\text{stock}}^F(m) := \exists h_1 \dots \exists h_m, \forall \mathbf{z}_1, \forall \mathbf{z}_2 \bigvee_{i=1}^m (F(\mathbf{z}_1) \wedge (h_i(\mathbf{z}_1) = h_i(\mathbf{z}_2)) \rightarrow \neg F(\mathbf{z}_2))$$

In the above formula, the  $h_i$ ’s are hash functions from  $\mathcal{H}(n, m, 2)$ . The formula says that there exist  $m$  such hash functions such that for any two assignments  $\mathbf{z}_1, \mathbf{z}_2 \in$

$\{0, 1\}^n$ , if  $z_1$  is a solution for  $F$  and some hash function  $h_i$  maps  $z_1$  and  $z_2$  to the same cell, then  $z_2$  cannot be a solution. In other words, for each solution some hash function puts it in a cell where it is the only solution. Recall that  $\exists h_i$  is shorthand for existentially quantifying over coefficients of  $h$ .

Now, the following claim and lemma provide necessary and sufficient conditions for  $\varphi_{stock}^F(m)$  evaluating to True.

**Claim 3.1** (Necessary). *If  $\varphi_{stock}^F(m) = 1$ , then  $|sol(F)| \leq m \cdot 2^m$ .*

The following lemma capturing sufficient conditions is inherent in Sipser’s work (Sipser 1983); to obtain the precise constant for our analysis, we provide an alternate simpler argument (proofs can be found in the full version at (Meel & Chakraborty & Akshay 2023)).

**Lemma 3.2** (Sufficient). *If  $|sol(F)| \leq 2^{m-2}$ , then  $\varphi_{stock}^F(m) = 1$*

We can now write Stockmeyer’s algorithm as follows:

---

#### Algorithm 1: Stock( $F$ )

---

```

1:  $v \leftarrow 0$ ;  $hashsgn_{stock} \leftarrow \emptyset$ ,  $CntEst \leftarrow 0$ ;
2:  $F' \leftarrow MakeCopies(F, \log n)$  /  $F'$  has  $n \log n$  variables
3: for  $m=1$  to  $n \log n$  do
4:   (ret, assign)  $\leftarrow$  2QBFCheck( $\varphi_{stock}^{F'}(m)$ );
5:   if ret == 1 then
6:      $v = m$ ;
7:      $hashsgn_{stock} = assign$ ;
8:   break
9:  $CntEst = 2^{\lfloor \frac{v}{\log n} \rfloor}$ ;
10: return ( $v$ ,  $CntEst$ ,  $hashsgn_{stock}$ )
    
```

---

Equipped with necessary and sufficient conditions for the satisfiability of  $|sol(F)| \leq 2^{m-2}$ , we now describe Stockmeyer’s algorithm, as presented in Algorithm 1. We first invoke  $MakeCopies(F, \log n)$  subroutine, which makes  $\log(n)$  many copies of  $F$ , each with a fresh set of variables and conjuncts them. Now, the algorithm runs over  $m$  from 1 to number of variables of  $F'$ , i.e.,  $n \log n$  and checks if  $\varphi_{stock}^{F'}(m)$  evaluates to 1. Each such call is a 2QBFCheck as  $\varphi_{stock}^{F'}$  is a  $\Sigma_2^P$  formula.

Now, if  $\varphi_{stock}^{F'}(i) = 1$ , then  $\varphi_{stock}^{F'}(i+1) = 1$  as well, by the definition of the formula, so the above algorithm essentially searches for the first point (smallest value of  $m$ ) where  $\varphi_{stock}^{F'}(i) = 1$  and returns the estimate of number of solutions at that point as  $2^{\lfloor \frac{v}{\log n} \rfloor}$ . Thus, when algorithm breaks at line 8, we have  $\varphi_{stock}^{F'}(v) = 1$  and  $\varphi_{stock}^{F'}(v-1) = 0$ .

**Theorem 3.3** (Stockmeyer). *Algorithm 1 makes  $O(n \log n)$  queries to  $\Sigma_2^P$  oracle and solves DAC.*

### 3.1 Auditing Stockmeyer’s Algorithm

Now to check the result/count given by the above algorithm, what can we do? In other words, what could be considered as a *certificate* for the answer? If we inspect the above algorithm, we can see that the crux is to identify the smallest value of  $m$  for which 2QBFCheck returns True.

Thus, at  $v$ , we need to check that the True answer is correct, i.e., the  $v^{th}$  call to  $\Sigma_2^P$  oracle. This is easy since we can use the certificate returned by the call, i.e., the set of hash functions  $h_1, \dots, h_v$ , i.e.,  $hashsgn_{stock}$ . Given this certificate (indeed a solver returns this) we can pass it to the verifier and hence, a query to NP oracle suffices.

However, this does not suffice. We also need to check that at  $v-1$ , the False answer is correct, i.e., at  $v-1$ , we need to check that  $\varphi_{stock}^{F'}(v-1) = 0$ , which can be answered by a query to  $\Sigma_2^P$  oracle. More precisely, for  $m = v-1$ , we need to check that  $\neg \varphi_{stock}^{F'}(v-1) = 1$ , i.e.,  $\forall h_1 \dots \forall h_m, \exists z_1, \exists z_2 \bigwedge_{i=1}^m (F(z_1) \wedge (h_i(z_1) = h_i(z_2)) \wedge F(z_2))$ .

---

#### Algorithm 2: StockAudit( $F, Stock(F)$ )

---

```

1: poscheck  $\leftarrow$  0; negcheck  $\leftarrow$  0;
2: poscheck  $\leftarrow$  CoNPCheck( $\varphi_{stock}^{F'}(v)[h_1 \dots, h_v \leftarrow$ 
    $hashsgn_{stock}]$ );
3: negcheck  $\leftarrow$  2QBFCheck( $\neg \varphi_{stock}^{F'}(v-1)$ );
4: if poscheck  $\wedge$  negcheck == 1 then
5:   return Verified;
    
```

---

Thus, the complexity is dominated by the “False” check at  $v-1$ . In fact combining the two checks as done in line 4, we just require a single call to 2-QBF solver. Thus the *audit-complexity*, or the number of variables on which this call is made is  $n^2 \log^2(n) + 2n \log n$ . Thus we obtain:

**Corollary 3.3.1.** *The audit complexity of Algorithm 1 is  $O(n^2 \log^2 n)$ .*

## 4 Trading Algorithm vs Audit Complexity

Observe that the dominant factor of  $n^2 \log n$  in the audit complexity of Stockmeyer’s algorithm was the need to certify “False” returned by 2QBFCheck. Therefore, a plausible approach to improving the audit complexity would be to develop an algorithm whose audit does not have to rely on certifying any of the “False” values returned by the underlying QBF checker. To this end, we construct a new formula  $\varphi$  whose validity would imply both an lower and upper bound (without requires a separate check for False as in the previous case). At a high-level, our approach is to partition the solution space into cells and then assert that every cell has neither too many solutions nor too few solutions.

**Not Too Many Solutions** For a given input Boolean formula  $F$  over  $n$  variables, consider the formula:

$\exists h \in \mathcal{H}(n, m, n) \forall \alpha \in \{0, 1\}^m \mathbf{y}_1, \mathbf{y}_2 \dots \mathbf{y}_{u+1} \in \{0, 1\}^n$   
 ConstructNotMany( $F, h, m, \alpha, \mathbf{y}_1, \mathbf{y}_2 \dots \mathbf{y}_{u+1}$ )

where,  $ConstructNotMany(F, h, m, \alpha, \mathbf{y}_1, \mathbf{y}_2 \dots \mathbf{y}_{u+1}) :=$

$$\left( \bigwedge_{i=1}^{u+1} (F(\mathbf{y}_i) \wedge h(\mathbf{y}_i) = \alpha) \rightarrow \bigvee_{i=1}^u (\mathbf{y}_{u+1} = \mathbf{y}_i) \right)$$

The above formula encodes that there is a  $n$ -wise independent hash function such that for each  $m$ -sized cell  $\alpha$ , and every set of  $u+1$  solutions (each solution being an assignment to  $n$  variables), if all of them are mapped to the same

cell, then two of them must be identical. So there are at most  $u$  solutions in each cell.

**At Least Few Solutions** Next, let us consider the formula

$$\exists h \in \mathcal{H}(n, m, n) \forall \alpha \in \{0, 1\}^m \exists z_1, z_2, \dots, z_\ell \in \{0, 1\}^n$$

ConstructAtLeastFew( $F, h, m, \alpha, z_1, z_2, \dots, z_\ell$ )

where, ConstructAtLeastFew( $F, h, m, \alpha, z_1, z_2, \dots, z_\ell$ ) :=

$$\left( \bigwedge_{i=1}^{\ell} (F(z_i) \wedge h(z_i) = \alpha \wedge \bigwedge_{i=1, j=i+1}^{\ell} (z_i \neq z_j)) \right)$$

This formula says that there is a hash function such that for each cell  $\alpha$ , there are  $\ell$  solutions which all map to the cell  $\alpha$ . Hence, if they are all distinct, in each cell the number of solutions is at least  $\ell$ . Thus the number of solutions is bounded below by  $\ell$  times number of cells.

Combining these, for any Boolean formula  $F$ , parameters  $\ell, u$ , we obtain a single formula asserting existence of a hash function such that each cell has a number of solutions that is simultaneously bounded from above by  $u$  and below by  $\ell$ .

$$\begin{aligned} \varphi_{Cells}^{(F, \ell, u)}(m) := & \exists h \forall \alpha, \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{u+1} \exists z_1, z_2, \dots, z_\ell \\ & (\text{ConstructNotMany}(F, h, m, \alpha, \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{u+1}) \\ & \wedge \text{ConstructAtLeastFew}(F, h, m, \alpha, z_1, z_2, \dots, z_\ell)) \end{aligned}$$

Note that the formula has a  $\exists \forall \exists$  alternation, i.e., it is a  $\Sigma_3^P$  formula. Then,

**Proposition 4.1.** *If  $\varphi_{Cells}^{(F, \ell, u)}(m) = 1$ , then  $\ell \cdot 2^m \leq |\text{sol}(F)| \leq u \cdot 2^m$ .*

Assuming that it is possible to show that such a hash function  $h$  with polynomial time evaluation and polynomial size description indeed exists (we will do so below), we now design an algorithm for approximate model counting. The algorithm simply goes over every value of  $m$  from 1 to  $n$  and checks whether  $\varphi_{Cells}^{(F, \ell, u)}(m)$  is True. We present the pseudocode of the algorithm in Algorithm 3.

---

Algorithm 3: EqualCellsCounter( $F$ )

---

```

1:  $u \leftarrow 16384n$ ;
2:  $\ell \leftarrow 1024n$ ;
3:  $\text{hashassgn}_{cells} \leftarrow 0$ ;  $\text{CntEst} \leftarrow 0$ ;
4: for  $m = 1$  to  $n$  do
5:   (ret, assign)  $\leftarrow$  3QBFCheck( $\varphi_{Cells}^{(F, \ell, u)}(m)$ );
6:   if ret==1 then
7:      $\text{hashassgn}_{cells} = \text{assign}$ ;
8:   break
9:  $\text{CntEst} = \ell \cdot 2^m$ ;
10: return (CntEst,  $\text{hashassgn}_{cells}$ )
```

---

**Theorem 4.2.** *Algorithm 3 solves DAC and makes  $O(n)$  calls to a  $\Sigma_3^P$  oracle.*

*Proof.* First observe that by definition of  $\varphi_{Cells}^{(F, \ell, u)}(m)$ , and Proposition 4.1, when the check in line 6 in Algorithm 3 passes, we must have  $\ell \cdot 2^m \leq |\text{sol}(F)| \leq u \cdot 2^m$ .

Since  $\frac{u}{\ell} = 16$ , we obtain a 16-factor approximation, i.e.,  $\frac{|\text{sol}(F)|}{16} \leq \text{CntEst} \leq |\text{sol}(F)|$ . Further, observe that we make at most  $n$  calls to  $\Sigma_3^P$  oracle.

It remains to show that there indeed exists  $h$  s.t., with  $\ell$  and  $u$  as above, for some  $m \in [n]$ , the check in line 6 will pass. To this end, we will utilize the probabilistic method.

Let us fix a cell  $\alpha$ . Let  $m = \lfloor \log |\text{sol}(F)| \rfloor - 12 - \log n$ . For  $\mathbf{y} \in \{0, 1\}^n$ , define the indicator variable  $\gamma_{\mathbf{y}, \alpha}$  such that  $\gamma_{\mathbf{y}, \alpha} = \begin{cases} 1 & h(\mathbf{y}) = \alpha \\ 0 & \text{otherwise} \end{cases}$ . Therefore,  $\mu_m =$

$$\sum_{\mathbf{y} \in \text{sol}(F)} \mathbb{E}[\gamma_{\mathbf{y}, \alpha}] = \frac{|\text{sol}(F)|}{2^m}.$$

Substituting value of  $m$ , we have  $\mu_m \in [4096n, 8192n]$ . Now consider  $h \xleftarrow{R} \mathcal{H}(n, m, n)$  i.e., chosen randomly from a  $n$ -wise independent hash family. Then  $\{\gamma_{\mathbf{y}, \alpha}\}$  are  $n$ -wise independent,

$$\begin{aligned} & \Pr[\text{Cnt}_{\langle F, h, \alpha \rangle} < 1024n \text{ or } \text{Cnt}_{\langle F, h, \alpha \rangle} > 16384n] \\ & \leq \Pr \left[ \left| \text{Cnt}_{\langle F, h, \alpha \rangle} - \mu_m \right| \geq \frac{1}{2} \mu_m \right] \\ & \leq 8 \cdot \left( \frac{4(n\mu_m + n^2)}{\mu_m^2} \right)^{n/2} \\ & \leq 8 \cdot \left( \frac{4 * 4097}{4096 * 4096} \right)^{n/2} \leq 8 * 31^{-n} \end{aligned}$$

Thus,  $\Pr[\exists \alpha, \text{Cnt}_{\langle F, h, \alpha \rangle} < 1024n \text{ or } \text{Cnt}_{\langle F, h, \alpha \rangle} > 16384n] \leq 8 \cdot \left(\frac{2}{31}\right)^n < 1$ , for  $n \geq 1$ , which implies  $\Pr[\forall \alpha, \text{Cnt}_{\langle F, h, \alpha \rangle} > 1024n \text{ and } \text{Cnt}_{\langle F, h, \alpha \rangle} \leq 16384n] > 0$ . Since the probability space is defined over a random choice of  $h$ , there must exist an  $h$  such that we have  $\exists h \forall \alpha, \text{Cnt}_{\langle F, h, \alpha \rangle} > 1024n \text{ and } \text{Cnt}_{\langle F, h, \alpha \rangle} \leq 16384n$ . Furthermore, observe that  $\ell = 1024n$  and  $u = 16384n$ , we have that  $\varphi_{Cells}^{(F, \ell, u)}(m)$  is True.  $\square$

#### 4.1 Auditing EqualCellsCounter( $F$ )

Unlike for Stockmeyer's algorithm, for Algorithm 3, a single call to a  $\Sigma_2^P$  oracle is sufficient for verification and this call is intrinsically different from the earlier algorithm.

---

Algorithm 4: EqualCellsAudit( $F$ , EqualCellsCounter( $F$ ))

---

```

1: ck  $\leftarrow$  0;  $m = \lfloor \log(\frac{\text{CntEst}}{n}) \rfloor - 10$ ;  $\ell \leftarrow 1024n$ ;  $u \leftarrow 16384n$ ;
2: ck  $\leftarrow$  2QBFCheck( $\varphi_{Cells}^{(F, \ell, u)}(m)[h \leftarrow \text{hashassgn}_{cells}]$ );
3: if ck == 1 then
4:   return Verified;
```

---

That is, we substitute in the formula  $\varphi_{Cells}^{(F, \ell, u)}$  the value returned by Algorithm 3 and the hash functions  $\text{hashassgn}_{cells}$  passed as witness and check if the resulting oracle call gives True. Now, examining the audit complexity, we can see that the number of variables on which the query in Algorithm 4 is bounded by  $n(2n + 2)$ :  $n$  variables for  $\alpha$  (since  $m \leq n$ ),  $n^2 + n$  for  $u + 1$  many  $\mathbf{y}_i$ 's and another  $n^2$  for  $\ell$  many  $\mathbf{z}_i$ 's, since  $u$  and  $\ell$  can at most be  $n$ . Thus, we get

**Corollary 4.2.1.** *The audit complexity of Algorithm 3 is  $O(n^2)$ .*

## 5 Improving the Audit Complexity

We now focus on further improving the audit complexity. We start by observing that the factor  $n^2$  in the audit complexity of Algorithm 3 was due to the size of cell being  $O(n)$  wherein every solution requires  $n$  variables to represent, and therefore, contributing to the factor of  $n^2$ . To this end, we ask if we can have cells of constant size; achieving this goal has potential to reduce the audit complexity to  $O(n)$ .

However, as we will see, having cells of size 1 suffices only for an  $n$ -factor approximation of the count. To achieve constant factor approximation, we rely on the standard technique of making multiple copies (as used in Section 3), which will finally achieve an audit complexity of  $O(n \log n)$ .

Taking a step back, recall that Stockmeyer’s approach too relied on cells of size 1; after all,  $\varphi_{stock}^F(m)$  asserted that for every solution  $z$ , there is a hash function  $h$  that maps  $z$  to a cell where it is the only solution. But the higher audit complexity of Stockmeyer’s algorithm was due to the need to certify the “False” answer. So, in order to achieve best of both worlds: i.e., have cells of size 1 and better audit complexity, we construct a new formula  $\varphi_{holes}^F(m)$  that operates on  $F$  and evaluates to True for sufficiently many values of  $m$  for which  $\varphi_{stock}^F(m)$  would evaluate to False, in the hope that certifying the “True” answer of  $\varphi_{holes}^F(m)$  would be easier: it indeed turns out to be true, as we will see later.

$$\varphi_{holes}^F(m) := \exists h_1, h_2, \dots, h_m, h_{m+1} \\ \forall \alpha \exists z \left( \bigvee_i F(z) \wedge h_i(z) = \alpha \right)$$

Essentially, the above formula is True if and only if there are  $m + 1$  hash functions such that for every cell  $\alpha$ , there is a solution that is mapped to the cell by some hash function.

We now turn to the design of an approximate counter, called AFCounter, with a much improved audit complexity. Given an input formula  $F$ , we will first construct  $F'$  by making  $\log n$  copies. We can design an approximate counting algorithm that queries  $\varphi_{stock}^{F'}(m)$  as well as  $\varphi_{holes}^{F'}(m)$  for all values of  $m$  and generate a certificate that consists of (1) smallest value of  $m$  for which  $\varphi_{stock}^{F'}(m)$  evaluates to True, denoted  $c_{high}$ , and (2) the largest value of  $m$  for which  $\varphi_{holes}^{F'}(m)$  evaluates to True, denoted  $c_{low}$ . A surprising aspect of our algorithm design is that we are able to compute the estimate of CntEst without  $c_{low}$ , i.e., purely based on the  $c_{high}$ , but the certificate returned by AFCounter uses  $c_{low}$  and associated set of hash functions.

**Theorem 5.1.** *Algorithm 5 solves DAC and makes  $O(n \log n)$  calls to  $\Sigma_3^P$  oracle.*

*Proof.* Observe that AFCounter returns the same output as that of Algorithm 1, therefore, the correctness follows from the correctness of Algorithm 1. Furthermore, observe that AFCounter makes at most  $n \log n$  calls to  $\Sigma_2^P$  oracle calls and at most  $n \log n$  calls to  $\Sigma_3^P$  oracle calls.  $\square$

But even though the correctness of the algorithm followed easily from earlier results, we will now see that proving the

---

### Algorithm 5: AFCounter( $F$ )

---

```

1:  $c_{low} \leftarrow 0; c_{high} \leftarrow n'; n' \leftarrow n \cdot \log n; \text{CntEst} \leftarrow 0;$ 
2:  $F' \leftarrow \text{MakeCopies}(F, \log n)$ 
3:  $\text{hashassgn}_{\text{stock}} \leftarrow \emptyset; \text{hashassgn}_{\text{holes}} \leftarrow \emptyset$ 
4: for  $m = 1$  to  $n'$  do
5:    $(\text{assign}, \text{ret}) \leftarrow 3\text{QBFCheck}(\varphi_{holes}^{F'}(m))$ 
6:   if  $\text{ret} == 0$  then
7:      $c_{low} = m - 1$ 
8:     break
9:    $\text{hashassgn}_{\text{holes}} \leftarrow \text{assign}$ 
10: for  $m' = 1$  to  $n'$  do
11:    $(\text{assign}, \text{ret}) \leftarrow 2\text{QBFCheck}(\varphi_{stock}^{F'}(m'))$ 
12:   if  $\text{ret} == 1$  then
13:      $c_{high} = m'$ 
14:      $\text{hashassgn}_{\text{stock}} \leftarrow \text{assign}$ 
15:     break
16:  $\text{CntEst} \leftarrow 2^{\frac{c_{high}}{\log n}}$ 
17: return
    ( $\text{CntEst}, c_{low}, c_{high}, \text{hashassgn}_{\text{stock}}, \text{hashassgn}_{\text{holes}}$ )

```

---

witness, i.e., the audit requires more work. To this end, we will show a relationship between  $c_{high}$  and  $c_{low}$ .

We first provide sufficient condition for  $\varphi_{holes}^F(m) = 1$ .

**Lemma 5.2** (Sufficient). *If  $|\text{sol}(F)| \geq 2^{m+3}$ , then  $\varphi_{holes}^F(m) = 1$ .*

*Proof.* Let us fix a cell  $\alpha$  and  $h_i \xleftarrow{R} H(n, m, 2)$ . For  $\mathbf{y} \in \{0, 1\}^n$ , define the indicator variable  $\gamma_{\mathbf{y}, \alpha, i}$  such that  $\gamma_{\mathbf{y}, \alpha, i} = 1$  if  $h_i(\mathbf{y}) = \alpha$  and 0 otherwise. Let  $\text{Cnt}_{\langle F, h_i, \alpha \rangle} = \sum_{\mathbf{y} \in \text{sol}(F)} \gamma_{\mathbf{y}, \alpha, i}$ . Thus  $\mu_m = \sum_{\mathbf{y} \in \text{sol}(F)} \mathbb{E}[\gamma_{\mathbf{y}, \alpha, i}] = \frac{|\text{sol}(F)|}{2^m}$ .

By assumption we have  $|\text{sol}(F)| \geq 2^{m+3}$ , which implies that  $\mu_m \geq 8$ . Further, since  $h_i$  is chosen from a 2-wise independent hash family, we infer that  $\{\gamma_{\mathbf{y}, \alpha, i}\}$  are 2-wise independent, and can conclude that  $\sigma^2[\text{Cnt}_{\langle F, h_i, \alpha \rangle}] \leq \mu_m$ .

Using Payley-Zygmund inequality, we have

$$\begin{aligned} \Pr[\text{Cnt}_{\langle F, h_i, \alpha \rangle} < 1] &\leq \Pr[\text{Cnt}_{\langle F, h_i, \alpha \rangle} < \frac{1}{8} \cdot \mu_m] \\ &\leq 1 - \left(1 - \frac{1}{8}\right)^2 \frac{\mu_m^2}{\sigma^2[\text{Cnt}_{\langle F, h_i, \alpha \rangle}] + \mu_m^2} \\ &\leq 1 - \left(1 - \frac{1}{8}\right)^2 \frac{\mu_m}{1 + \mu_m} \end{aligned}$$

As  $h_1, h_2, \dots, h_{m+1}$  are chosen independently, we have

$$\begin{aligned} &\Pr \left[ \bigcap_{i=1}^{m+1} \{\text{Cnt}_{\langle F, h_i, \alpha \rangle} < 1\} \right] \\ &= (\Pr [\{\text{Cnt}_{\langle F, h_i, \alpha \rangle} < 1\}])^{m+1} \leq \frac{1}{2^{m+1}} \end{aligned}$$

Finally, we apply union bound to bound  $\Pr [\exists \alpha, \bigcap_{i=1}^{m+1} \{\text{Cnt}_{\langle F, h_i, \alpha \rangle} < 1\}] \leq \frac{2^m}{2^{m+1}} = \frac{1}{2}$ .

Therefore,  $\Pr [\forall \alpha, \bigcup_{i=1}^{m+1} \{\text{Cnt}_{\langle F, h_i, \alpha \rangle} \geq 1\}] \geq \frac{1}{2}$ .

Observe the probability space is defined over the random choice of  $h_1, h_2, \dots, h_{m+1}$ , and therefore, since the probability of randomly chosen  $h$  ensuring  $\forall \alpha, \bigcup_{i=1}^{m+1} \{\text{Cnt}_{\langle F, h_i, \alpha \rangle} \geq 1\}$  is non-zero, therefore, there must exist an  $h_1, h_2, \dots, h_m$ . Formally, when  $|sol(F)| > 2^{m+3}$ , we have  $\exists h_1, h_2, \dots, h_{m+1} \forall \alpha \exists z (\bigvee_i F(z) \wedge h_i(z) = \alpha)$ .  $\square$

From this we can infer the following fact.

**Lemma 5.3.**  $c_{high} - c_{low} \leq 7$ .

*Proof.* Let  $m^* = \lfloor \log |sol(F')| \rfloor$ . Then, from Lemma 3.2, we have  $c_{high} \leq m^* + 3$ . Further, from Lemma 5.2, we obtain  $c_{low} \geq m^* - 4$ . Thus, we have  $c_{high} - c_{low} \leq 7$ .  $\square$

### 5.1 Audit Complexity of Algorithm 5

Now, we focus on certifying Algorithm 5. Observe that,  $c_{high}$  is the smallest value of  $m$  for which  $\varphi_{stock}^{F'}(m)$  evaluates to True and  $c_{low}$  is the largest value of  $m$  for which  $\varphi_{holes}^{F'}(m)$  evaluates to True. Thus, a straightforward approach would be to certify that  $c_{high}$  is indeed smallest value of  $m$  for which  $\varphi_{stock}^{F'}(m)$  but that is precisely what we did in Algorithm 2, which led to the audit complexity of  $O(n^2 \log^2 n)$ . How do we overcome the barrier?

The key surprising observation is that we do not need to certify that  $c_{high}$  is indeed the smallest value: instead we will just certify that  $\varphi_{stock}^{F'}(c_{high})$  evaluates to True and  $\varphi_{holes}^{F'}(c_{low})$  evaluates to True, and then we check  $c_{high} - c_{low} \leq 7$ , and these three facts allow the auditor to certify the estimate returned by Algorithm 5. Note that we were able to side step certifying  $c_{high}$  being the smallest value because we had access to  $c_{low}$ , which was not the case when we were trying to certify Algorithm 1.

---

Algorithm 6: CountAuditor( $F'$ , AFCounter( $F'$ ))

---

- 1: poscheck  $\leftarrow$  0; negcheck  $\leftarrow$  0;
  - 2: poscheck  $\leftarrow$  CoNPCheck( $\varphi_{stock}^{F'}(c_{high})[h_1 \dots, h_{c_{high}} \leftarrow \text{hashassign}_{stock}]$ );
  - 3: negcheck  $\leftarrow$  2QBFCheck( $\varphi_{holes}^{F'}(c_{low})[h_1 \dots, h_{c_{low}} \leftarrow \text{hashassign}_{holes}]$ );
  - 4: **if** (poscheck  $\wedge$  negcheck == 1)  $\wedge$  ( $c_{high} - c_{low} \leq 7$ ) **then return** Verified;
- 

To show correctness of CountAuditor, we also need the following, which gives a lower bound on no. of solutions:

**Lemma 5.4 (Necessary).** *If  $\varphi_{holes}^{F'}(m) = 1$ , then  $|sol(F')| \geq \frac{2^m}{m+1}$*

*Proof.* Let us construct a many-to-one mapping  $g$  from  $\{0, 1\}^m$  to  $sol(F')$  such that every edge from  $\alpha \in \{0, 1\}^m$  to  $z \in sol(F')$  is labeled with a non-empty set of hash functions  $\{h_i\}$  such that  $h_i(z) = \alpha$ . Such a mapping must exist when  $\varphi_{holes}^{F'}(m) = 1$ , since for every cell  $\alpha$ , we have  $z \in sol(F')$  such that  $h_i(z) = \alpha$  for some  $h_i$ . Furthermore, observe that there can be at most  $m + 1$  different  $\alpha$  that map to same  $z$  as every hash function maps  $z$  to only one unique

$\alpha$ . Therefore, the range of  $g$ , which is a subset of  $sol(F')$ , must be at least  $\frac{2^m}{m+1}$ .  $\square$

Finally, combining all these, we obtain:

**Theorem 5.5.** *The audit complexity of Algorithm 5 is  $O(n \log n)$ .*

*Proof.* We will first show that Algorithm 6 certifies the estimate returned by Algorithm 5, and then focus on measuring the audit complexity. From the True answers to poscheck and negcheck, the auditor can conclude using Lemma 5.4 and Claim 3.1 that

$$\frac{2^{c_{low}}}{c_{low} + 1} \leq |sol(F')| \leq c_{high} \cdot 2^{c_{high}} \quad (1)$$

From the rhs of Eq 1, and since  $c_{high} \leq n \log n \leq n^2$ , we have  $\frac{|sol(F')|}{n^2} \leq \frac{|sol(F')|}{c_{high}} \leq 2^{c_{high}}$

Now, from line 4 of the audit, we have  $c_{high} - c_{low} \leq 7$ . We combine this with lhs of Eq 1, and observe that  $c_{low} \leq n \log n$ , and hence  $c_{low} + 1 \leq n^2$  for  $n \geq 12$ ,  $n^2 > 128$ , and  $2^{c_{high}} \leq n^2 \cdot 2^{c_{low}} \leq n^2 \cdot (c_{low} + 1) \cdot |sol(F')| \leq n^4 \cdot |sol(F')|$ . Noting that  $|sol(F')| = |sol(F)|^{\log n}$ , we have

$$\left(\frac{|sol(F)|}{4}\right)^{\log n} \leq (2^{\frac{c_{high}}{\log n}})^{\log n} \leq (16 \cdot |sol(F)|)^{\log n}$$

i.e.,  $\frac{|sol(F)|}{4} \leq \text{CntEst} \leq 16 \cdot |sol(F)|$

which is a 16-factor approximation. As before, the calls in line 2 and line 3 can be combined and answered using a single  $\Sigma_2^F$  oracle query. Finally, the number of variables on which this query is made is  $O(|\alpha| + |z|)$ , thus of size  $O(n') = O(n \log n)$ .  $\square$

## 6 Discussion and Perspectives

The design of algorithms has traditionally focused on optimizing resources like time, space, or communication without necessarily considering how hard it is to independently certify correctness of results. Yet, the need for certified or auditable results is becoming increasingly important in a world where untrusted components are unavoidable in the implementation of any non-trivial software.

In this paper, we presented a principled approach towards addressing this problem by showing that appropriate measures of audit complexity can be treated as a first-class computational resource to be optimized at the design stage of an algorithm itself, rather than as an afterthought. For deterministic approximate counting, this approach yields completely new algorithms that are sub-optimal with respect to time/space resources, and yet permit significantly simpler certification in practice. We believe the same “trade off” applies to other hard combinatorial problems as well, and one needs to navigate the algorithm design space carefully to strike the right balance between audit complexity and time/space/communication complexity. We hope our work opens the door for further such investigations. As concrete steps for future work, we wish to explore the audit complexity versus time/space complexity tradeoff in probabilistic approximate counting.

## 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, and Ministry of Education Singapore Tier 1 Grant [R-252-000-B59-114].

## References

- Barthe, G.; Grégoire, B.; Kunz, C.; and Rezk, T. 2006. Certificate Translation for Optimizing Compilers. In *Static Analysis Symposium*, 301–317.
- Beck, G.; Zinkus, M.; and Green, M. 2020. Automating the development of chosen ciphertext attacks. In *29th USENIX Security Symposium (USENIX Security 20)*, 1821–1837.
- Bellare, M.; Goldreich, O.; and Petrank, E. 2000. Uniform Generation of NP-Witnesses Using an NP-Oracle. *Information and Computation*, 163(2): 510–526.
- Bellare, M.; and Rompel, J. 1994. Randomness-Efficient Oblivious Sampling. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, 276–287. IEEE Computer Society.
- Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. ISBN 978-1-64368-160-3.
- Cheung, K. K. H.; and Moazzez, B. 2016. Certificates of Optimality for Mixed Integer Linear Programming Using Generalized Subadditive Generator Functions. *Advances in Operations Research*, 2016: 5017369:1–5017369:11.
- Ekici, B.; Mebsout, A.; Tinelli, C.; Keller, C.; Katz, G.; Reynolds, A.; and Barrett, C. W. 2017. SMTCoq: A Plug-In for Integrating SMT Solvers into Coq. In *Proc. of CAV*, 126–133.
- Gocht, S.; Martins, R.; Nordström, J.; and Oertel, A. 2022. Certified CNF Translations for Pseudo-Boolean Solving. In *Proc. of SAT*, 16:1–16:25.
- Heule, M. J.; Hunt, W. A.; and Wetzler, N. 2013. Trimming while checking clausal proofs. In *Formal Methods in Computer-Aided Design*, 181–188.
- Klebanov, V. 2014. Precise quantitative information flow analysis—a symbolic approach. *Theoretical Computer Science*, 538: 124–139. Quantitative Aspects of Programming Languages and Systems (2011-12).
- Magron, V.; Allamigeon, X.; Gaubert, S.; and Werner, B. 2015. Formal Proofs for Nonlinear Optimization. *Journal of Formalized Reasoning*, 8(1): 1–24.
- McConnell, R. M.; Mehlhorn, K.; Näher, S.; and Schweitzer, P. 2011. Certifying algorithms. *Comput. Sci. Rev.*, 5(2): 119–161.
- Meel, K. S.  Chakraborty, S.  Akshay, S. 2023. Auditable Algorithms for Approximate Model Counting. *arXiv preprint arXiv:2312.12362*.
- Necula, G. C.; and Lee, P. 1998. The Design and Implementation of a Certifying Compiler. In *Proc. of PLDI*, 333–344.
- Sipser, M. 1983. A Complexity Theoretic Approach to Randomness. In *Proc. of STOC*, 330–335. ACM.
- Soos, M.; Gocht, S.; and Meel, K. S. 2020. Tinted, Detached, and Lazy CNF-XOR Solving and Its Applications to Counting and Sampling. In *Proc. of CAV*, volume 12224, 463–484. Springer.
- Stockmeyer, L. 1983. The complexity of approximate counting. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, 118–126.
- Teuber, S.; and Weigl, A. 2021. Quantifying software reliability via model-counting. In *International Conference on Quantitative Evaluation of Systems*, 59–79. Springer.
- Valiant, L. G. 1979. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8: 189–201.
- Valiant, L. G. 1984a. A Theory of the Learnable. In DeMillo, R. A., ed., *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, 436–445. ACM.
- Valiant, L. G. 1984b. A Theory of the Learnable. *Communications of the ACM*, 27(11): 1134–1142.
- Wetzler, N.; Heule, M. J. H.; and Hunt, W. A. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In *Theory and Applications of Satisfiability Testing (SAT)*, 422–429.
- Zinkus, M.; Cao, Y.; and Green, M. 2023. McFIL: Model Counting Functionality-Inherent Leakage. *arXiv preprint arXiv:2306.05633*.