A New Probabilistic Algorithm for Approximate Model Counting and Extensions for Numeric Domains [⋆]

Cunjing Ge[1,3], Feifei Ma[1,2,3](✉), Tian Liu[4], Jian Zhang[1,3], and Xutong Ma[3,5]

[1] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences
{gecj, maff, zj}@ios.ac.cn
[2] Laboratory of Parallel Software and Computational Science,
Institute of Software, Chinese Academy of Sciences
[3] University of Chinese Academy of Sciences
[4] School of Electronics Engineering and Computer Science, Peking University
[5] Technology Center of Software Engineering,
Institute of Software, Chinese Academy of Sciences

Abstract. Constrained counting is important in domains ranging from artificial intelligence to software analysis. There are already a few approaches for counting models over various types of constraints. Recently, hashing-based approaches achieve success but still rely on solution enumeration. In the IJCAR version, a probabilistic approximate model counter is proposed, which is also a hashing-based universal framework, but with only satisfiability queries. A dynamic stopping criteria, for the algorithm, is presented, which has not been studied yet in previous works of hashing-based approaches. Although the algorithm lacks theoretical guarantee, it works well in practice. In this paper, we further extend our approach to SMT(LIA) formulas with a new encoding technique.

1 Introduction

Constrained counting, the problem of counting the number of solutions for a set of constraints, is important in theoretical computer science and artificial intelligence. It-s interesting applications in several fields include program analysis [25, 18, 19, 15, 27, 16], probabilistic inference [28, 11], planning [13] and privacy/confidentiality verification [17]. Constrained counting for propositional formulas is also called model counting, to which probabilistic inference is easily reducible. However, model counting is a canonical #P-complete problem, even for polynomial-time solvable problems like 2-SAT [33], thus it presents fascinating challenges for both theoreticians and practitioners.

There are already a few approaches for counting solutions over propositional logic formulas and SMT(BV) formulas. Recently, hashing-based approximate counting achieves both strong theoretical guarantees and good scalability [26]. The use of universal

[⋆] This work has been supported by the National 973 Program of China under Grant 2014CB340701, Key Research Program of Frontier Sciences, Chinese Academy of Sciences (CAS) under Grant QYZDJ-SSW-JSC036, and the National Natural Science Foundation of China under Grant 61100064. Feifei Ma is also supported by the Youth Innovation Promotion Association, CAS.

hash functions in counting problems began in [30, 32], but the resulting algorithm scaled poorly in practice. A scalable approximate counter `ApproxMC` in [9] scales to large problem instances, while preserving rigorous approximation guarantees. `ApproxMC` has been extended to finite-domain discrete integration, with applications to probabilistic inference [14, 6, 3]. It was improved by designing efficient universal hash functions [22, 7] and reducing the use of NP-oracle calls from linear to logarithmic [10].

The basic idea in `ApproxMC` is to estimate the model count by randomly and iteratively cutting the whole space down to a small "enough" cell, using hash functions, and sampling it. The total model count is estimated by a multiplication of the number of solutions in this cell and the ratio of the whole space to the small cell. To determine the size of the small cell, which is essentially a small-scale model counting problem with the model counts bounded by some thresholds, a model enumeration in the cell is adopted. In previous works, the enumeration query was handled by transforming it into a series of satisfiability queries, which is much more time-consuming than a single satisfiability query. An algorithm called `MBound` [20] only invokes satisfiability query once for each cut. Its model count is determined with high precision by the number of cuts down to the boundary of being unsatisfiable. However, this property is not strong enough to give rigorous guarantees, and `MBound` only returns an approximation of upper or lower bound of the model count.

In the IJCAR version, a hashing-based approximate counting algorithm, with only satisfiability query, is proposed. Dynamic stopping criterion for the algorithm to terminate, once meeting the criterion of accuracy, is presented, which has not been proposed yet in previous works of hashing-based approaches. Theoretical insights over the efficiency of a prevalent heuristic strategy called leap-frogging are also provided. Prototype tools for propositional logic formulas, SMT(BV) formulas are implemented. An extensive evaluation on a suite of benchmarks demonstrates that (i) the approach significantly outperforms the state-of-the-art approximate model counters, including a counter designed for SMT(BV) formulas, and (ii) the dynamic stopping criterion is promising.

In this paper, we further present a new technique to encode XOR-based hash function over numeric domains. In addition, our approach is a general framework. Thus we managed to extend it to numeric domains, e.g., SMT(LIA) formulas. A prototype tool for SMT(LIA) formulas is also implemented.

The rest of this paper is organized as follows. Preliminary material is in Section 2, related works in Section 3, the algorithm in Section 4, extensions for SMT(LIA) formulas in Section 5, analysis in Section 6, experimental results in Section 7, and finally, concluding remarks in Section 8.

2 Preliminaries

Let $F(x)$ denote a propositional logic formula on n variables $x = (x_1, \ldots, x_n)$. Let S and S_F denote the whole space (the space of assignments) and the solution space of F, respectively. Let $\#F$ denote the cardinality of S_F, i.e. the number of solutions of F.

(ϵ, δ)-bound To count $\#F$, an (ϵ, δ) approximation algorithm, $\epsilon > 0$ and $\delta > 0$, is an algorithm which on every input formula F, outputs a number $\tilde{Y}$ such that $\Pr[(1 +$

$\epsilon)^{-1}\#F \leq \tilde{Y} \leq (1+\epsilon)\#F] \geq 1-\delta$. Such an algorithm is called a (ϵ, δ)-counter and the bound is called a (ϵ, δ)-bound [23].

Hash Function Let $\mathcal{H}_F$ be a family of XOR-based bit-level hash functions on the variables of a formula F. Each hash function $H \in \mathcal{H}_F$ is of the form $H(x) = a_0 \bigoplus_{i=1}^{n} a_i x_i$, where $a_0, \ldots, a_n$ are Boolean constants. In the hashing procedure `Hashing(F)`, a function $H \in \mathcal{H}_F$ is generated by independently and randomly choosing a_is from a uniform distribution. Thus for an assignment α, it holds that $\Pr_{H \in \mathcal{H}_F}(H(\alpha) = true) = \frac{1}{2}$. Given a formula F, let F_i denote a hashed formula $F \wedge H_1 \wedge \cdots \wedge H_i$, where $H_1, \ldots, H_i$ are independently generated by the hashing procedure.

Satisfiability Query Let `Solving(F)` denote the satisfiability query of a formula F. With a target formula F as input, the satisfiability of F is returned by `Solving(F)`.

Enumeration Query Let `Counting(F, p)` denote the bounded solution enumeration query. With a constraint formula F and a threshold p ($p \geq 2$) as inputs, a number s is returned such that $s = \min(p-1, \#F)$. Specifically, 0 is returned for unsatisfiable F, or $p = 1$ which is meaningless.

SMT(BV) Formula SMT(BV) formulas are quantifier-free and fixed-size that combine propositional logic formulas with constraints of bit-vector theory. For example, $\neg(x + y = 0) \vee (x = y << 1)$, where x and y are bit-vector variables, $<<$ is the shift-left operator. It can be regarded as a propositional logic formula $\neg A \vee B$ that combines bit-vector constraints $A \equiv (x + y = 0)$ and $B \equiv (x = y << 1)$. To apply hash functions to an SMT(BV) formula, a bit-vector is bit-blasted to a set of Boolean variables.

SMT(LIA) Formula Similar to SMT(BV) formulas, SMT(LIA) formulas combine propositional logic formulas with constraints of linear arithmetic theory (i.e., linear inequalities). For example, $\neg(x + y \geq 0) \vee (x < y - 1)$, where x and y are integer variables. It can be regarded as a propositional logic formula $\neg A \vee B$ that combines linear inequalities $A \equiv (x + y \geq 0)$ and $B \equiv (x < y - 1)$. To apply hash functions to an SMT(LIA) formula, we introduce a new encoding technique in Section 5.

3 Related Works

[2] showed that almost uniform sampling from propositional constraints, a closely related problem to constrained counting, is solvable in probabilistic polynomial time with an NP oracle. Building on this, [9] proposed the first scalable approximate model counting algorithm `ApproxMC` for propositional formulas. `ApproxMC` is based on a family of 2-universal bit-level hash functions that compute XOR of randomly chosen propositional variables. In the current work, this family of hash functions is adopted, which was shown to be 3-independent in [21], and is revealed to potentially possess better properties than expected by the experimental results and the theoretical analysis in the current work.

The sketch framework of `ApproxMC` [9, 12] is listed as Algorithm 1. Its inputs are a formula F and two accuracy parameters T and $pivot$, where T determines the number of times `ApproxMCCore` is invoked, and $pivot$ determines the threshold of the enumeration query. The function `ApproxMCCore` starts from the formula F_0, iteratively calls `Counting` and `Hashing` over each F_i, to cut the space (cell) of all models of F_0 using random hash functions, until the count of F_i is no larger than $pivot$, then

Algorithm 1

```
function ApproxMC($F$, $T$, $pivot$)
    for $1$ to $T$ do
        $c \leftarrow$ ApproxMCCore($F$, $pivot$)
        if ($c \neq 0$) then AddToList($C$, $c$)
    end for
    return FindMedian($C$)
end function
function ApproxMCCore($F$, $pivot$)
    $F_0 \leftarrow F$
    for $i \leftarrow 0$ to $\infty$ do
        $s \leftarrow$ Counting($F_i$, $pivot + 1$)
        if ($0 \leq s \leq pivot$) then return $2^i s$
        $H_{i+1} \leftarrow$ Hashing($F$)
        $F_{i+1} \leftarrow F_i \wedge H_{i+1}$
    end for
end function
```

breaks out of the loop and adds the approximation $2^i s$ into list C. The main procedure `ApproxMC` repeatedly invokes `ApproxMCCore` and collects the returned values, at last returning the median number of list C. The general algorithm in [7] is similar to Algorithm 1, but cuts the cell with dynamically determined proportion instead of the constant $\frac{1}{2}$, due to the word-level hash functions. [10] improves `ApproxMCCore` via binary search to reduce the number of enumeration queries from linear to logarithmic. This binary search improvement is orthogonal to our approach.

A recent work [1] considered a special family of shorter XOR-constraints to improve the efficiency of SAT solving while preserving rigourous guarantee. This improvement of hash functions is also orthogonal to our approach as we use hash functions and SAT solving as black boxes. However, it is unknown whether there exist similar theoretical results like [1].

4 Algorithm

In this section, a new hashing-based algorithm for approximate model counting, with only satisfiability queries, will be proposed, building on an assumption of a probabilistic approximate correlation between the model count and the probability of the hashed formula being unsatisfiable.

Let $F_d = F \wedge H_1 \wedge \cdots \wedge H_d$ be a hashed formula resulted by iteratively hashing d times independently over a formula F. F_d is unsatisfiable if and only if no solution of F satisfies F_d, thus $\Pr_{F_d}(F_d\ is\ unsat) = \Pr_{F_d}(F_d(\alpha) = false, \alpha \in S_F)$. Assume we have

$$\Pr_{F_d}(F_d\ is\ unsat) \approx (1 - 2^{-d})^{\#F}. \tag{1}$$

Then based on Equation (1), an approximation of $\#F$ is achieved by taking logarithm on the value of $\Pr_{F_d}(F_d\ is\ unsat)$, which is estimated in turn by sampling F_d.

This is the general idea of our approach. The pseudo-code is presented in Algorithm 2. The inputs are the target formula F and a constant T which determines the number of times `GetDepth` invoked. `GetDepth` calls `Solving` and `Hashing` repeatedly until an unsatisfiable formula F_{depth} is encountered, and returns the $depth$. Every time `GetDepth` returns a $depth$, the value of $C[i]$ is increased, for all $i < depth$. At line 9, the algorithm picks a number d such that $C[d]$ is close to $T/2$, since the error estimation fails when $C[d]/T$ is close to 0 or 1. The final result is returned by the formula $\log_{1-(1/2)^d} \frac{counter}{T}$ at line 11.

Algorithm 2 Satisfiability Testing based Approximate Counter (STAC)

```
 1: function STAC($F$, $T$)
 2:     initialize $C[i]$s with zeros
 3:     for $t \leftarrow 1$ to $T$ do
 4:         $depth \leftarrow$ GetDepth($F$)
 5:         for $i \leftarrow 0$ to $depth - 1$ do
 6:             $C[i] \leftarrow C[i] + 1$
 7:         end for
 8:     end for
 9:     pick a number $d$ such that $C[d]$ is closest to $T/2$
10:     $counter \leftarrow T - C[d]$
11:     return $\log_{1-2^{-d}} \frac{counter}{T}$   /* return 0 when $d = 0$ */
12: end function
13: function GETDEPTH($F$)
14:     $F_0 \leftarrow F$
15:     for $i \leftarrow 0$ to $\infty$ do
16:         $b \leftarrow$ Solving($F_i$)
17:         if ($b$ is false) then return $i$
18:         $H_{i+1} \leftarrow$ Hashing($F_i$)
19:         $F_{i+1} \leftarrow F_i \wedge H_{i+1}$
20:     end for
21: end function
```

Note that our approach is based on Equation (1) which is only an assumption. In Section 6, we provide theoretical analysis, including the bound of the approximation and the correctness of algorithm, based on the hypothesis. Then in Section 7, experimental results on an extensive set of benchmarks show that the approximation given by our approach fits the bound well. It indicates that Equation (1) is probably true as it is a reasonable explanation to the positive results.

Dynamic Stopping Criterion The essence of Algorithm 2 is a randomized sampler over a binomial distribution. The number of samples is determined by the value of T, which is pre-computed for a given (ϵ, δ)-bound, and we loosen the value of T to meet the guarantee in theoretical analysis. However, it usually does not loop T times in practice. A variation with dynamic stopping criterion is presented in Algorithm 3.

Line 2 to 7 is the same as Algorithm 2, still setting T as a stopping rule and terminating whenever $t = T$. Line 8 to 16 is the key part of this variation, calculating

Algorithm 3 STAC with Dynamic Stopping Criterion

```
 1: function STAC_DSC($F$, $T$, $\epsilon$, $\delta$)
 2:     initialize $C[i]$s with zeros
 3:     for $t \leftarrow 1$ to $T$ do
 4:         $depth \leftarrow$ GetDepth($F$)
 5:         for $i \leftarrow 0$ to $depth - 1$ do
 6:             $C[i] \leftarrow C[i] + 1$
 7:         end for
 8:         for each $d$ that $C[d] > 0$ do
 9:             $q \leftarrow \frac{t-C[d]}{t}$
10:             $M \leftarrow \log_{1-2^{-d}} q$
11:             $U \leftarrow \log_{1-2^{-d}}(q - z_{1-\delta}\sqrt{\frac{q(1-q)}{t}})$
12:             $L \leftarrow \log_{1-2^{-d}}(q + z_{1-\delta}\sqrt{\frac{q(1-q)}{t}})$
13:             if $U < (1+\epsilon)M$ and $L > (1+\epsilon)^{-1}M$ then
14:                 return $M$
15:             end if
16:         end for
17:     end for
18: end function
```

the binomial proportion confidence interval $[L, U]$ of an intermediate result M for each cycle. A commonly used formula $q \pm z_{1-\delta}\sqrt{\frac{q(1-q)}{t}}$ [4, 34] is adopted, which is justified by the central limit theorem to compute the $1 - \delta$ confidence interval. However, it becomes invalid for small sample size or proportion close to 0 or 1. In practice, we also considered some improvements, e.g., Wilson score interval [35]. The exact count $\#F$ lies in the interval $[L, U]$ with probability $1 - \delta$. Combining the inequalities presented in line 13, the interval $[(1 + \epsilon)^{-1}M, (1 + \epsilon)M]$ is the (ϵ, δ)-bound (if the assumption of Formula 1 holds). So the algorithm terminates when the condition in line 13 comes true. The time complexity of Algorithm 3 is still the same as the original algorithm, though it usually terminates earlier.

Satisfiability And Enumeration Query The bounded counting can be done by negating solution and calling SAT oracle repeatedly, which is employed by ApproxMC. In practice, enumerating solutions in this way is not very efficient. In evaluation section, experimental results show that the average number of SAT calls of `ApproxMC` is usually 20 to 30 times to `STAC`. It may also cause problems while extending to other kinds of formulas. For example, for linear integer arithmetic formula, inserting solution negation clauses will exponentially increase the number of calls of LIP solver.

Leap-frogging Strategy Recall that `GetDepth` is invoked T times with the same arguments, and the loop of line 15 to 20 in the pseudo-code of `GetDepth` in Algorithm 2 is time consuming for large i. A heuristic called leap-frogging to overcome this bottleneck was proposed in [8, 9]. Their experiments indicate that this strategy is extremely efficient in practice. The average depth $\bar{d}$ of each invocation of `GetDepth` is recorded.

In all subsequent invocations, the loop starts by initializing i to $\bar{d} - k \cdot \mathrm{offset}$, where $k \geq 1$. Note that if F_i is unsatisfiable, the algorithm repeatedly decreases i by increasing k and check the satisfiability of the new F_i, until a proper initialization i is found for satisfiable F_i. In practice, the constant offset is usually set to 5. Theorem 3 in Section 6 shows that the $depth$ computed by `GetDepth` lies in an interval $[d, d+7]$ with probability over 90%. So it suffices to invoke `Solving` in constant time since the second iteration.

5 Extensions to SMT(LIA) Formulas

Recall that we employ a family of XOR-based bit-level hash functions on the variables of a formula F. Each such hash function $H \in \mathcal{H}_F$ is of the form $H(\boldsymbol{b}) = a_0 \bigoplus_{i=1}^{n} a_i b_i$, where $a_0, \ldots, a_n$ are Boolean constants. In the hashing procedure, a function $H \in \mathcal{H}_F$ is generated by independently and randomly choosing a_is from a uniform distribution. Thus for an assignment $\boldsymbol{\alpha}$, it holds that $\Pr_{H \in \mathcal{H}_F}\big(H(\boldsymbol{\alpha}) = true\big) = \frac{1}{2}$. To extend the hash functions to numeric domains, we still have to maintain such probability property in order to fit our approach.

If we represent a Boolean variable via integers, i.e., 0 is false, 1 is true, intuitively, the XOR hash function $a_0 \bigoplus_{i=1}^{n} a_i b_i$ can be represented by $(\alpha_0 + \sum_{i=1}^{n} \alpha_i \times x_i) \mod 2$, where $\alpha_0, \ldots, \alpha_n$ are $\{0,1\}$-constants and $x_1, \ldots, x_n$ are $\{0,1\}$-variables. Then we extend the domain from $\{0,1\}$ to integers directly. Consider a formula F over integer domain. Let $\mathcal{I}_F$ denote the family of hash functions for the integer domain and each $I \in \mathcal{I}_F$ is of the form

$$I(\boldsymbol{x}) = (\alpha_0 + \sum_{i=1}^{n} \alpha_i \times x_i) \mod 2.$$

It is easy to see that $\Pr_{I \in \mathcal{I}_F}\big(I(\boldsymbol{\alpha}) = true\big) = \frac{1}{2}$ is hold for any assignment $\boldsymbol{\alpha}$ of F as well.

To extend our approach to the integer domain, it is necessary to adapt the new integer hash function to satisfiability query. However, in practice, the modulo operation is not supported in many satisfiability checkers, e.g., SMT(LIA) solvers. We introduce a new technique to encode the modulo operation via arithmetic operations. Given a simple constraint that contains a modulo operation,

$$f(\boldsymbol{x}) \mod m = n, \tag{2}$$

where $f(\boldsymbol{x})$ is a formula over variables $\boldsymbol{x}$, m and n are constants. We introduce a new integer variable y. Then consider the following constraint

$$f(\boldsymbol{x}) = m \times y + n. \tag{3}$$

Obviously, Equation 2 is satisfiable if and only if Equation 3 is satisfiable. If $f(\boldsymbol{x})$ is a linear arithmetic formula, Equation 3 is able to be solved by linear arithmetic solvers.

In conclusion, we transform the hash function constraint $(\alpha_0 + \sum_{i=1}^{n} \alpha_i \times x_i) \mod 2 = 1$ into a linear arithmetic constraint

$$(\alpha_0 + \sum_{i=1}^{n} \alpha_i \times x_i) = 2 \times y + 1.$$

It enables the extension of our approach to SMT(LIA) formulas. A prototype tool for SMT(LIA) formulas is also implemented.

6 Analysis

In this section, we assume Equation (1) holds. Based on this assumption, theoretical results on the error estimation of our approach are presented. For lack of space, we omit proofs in this section.

Recall that in Algorithm 2, $\#F$ is approximated by a value $\log_{1-2^{-d}} \frac{counter}{T}$. Let q_d denote the value of $(1-2^{-d})^{\#F}$. We obtain that $\Pr(F_d \ is \ unsat) = q_d$ for a randomly generated formula F_d. This is justified by Equations (1). Since the ratio $\frac{counter}{T}$ in Algorithm 2 is a proportion of successes in a Bernoulli trial process, which is used to estimate the value of q_d. Then $counter$ is a random variable following a binomial distribution $\mathbb{B}(T, q_d)$.

Theorem 1. *Let $z_{1-\delta}$ be the $1-\delta$ quantile of $\mathbb{N}(0,1)$ and*

$$T = max\left(\lceil(\frac{z_{1-\delta}}{2q_d(1-q_d^{\epsilon})})^2\rceil, \lceil(\frac{z_{1-\delta}}{2(q_d^{(1+\epsilon)^{-1}}-q_d)})^2)\rceil\right). \quad (4)$$

Then $\Pr[\frac{\#F}{1+\epsilon} \leq \log_{1-2^{-d}} \frac{counter}{T} \leq (1+\epsilon)\#F] \geq 1-\delta$.

Proof. By above discussions, the ratio $\frac{counter}{T}$ is the proportion of successes in a Bernoulli trial process which follows the distribution $\mathbb{B}(T, q_d)$. Then we use the approximate formula of a binomial proportion confidence interval $q_d \pm z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}}$, i.e., $\Pr[q_d - z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}} \leq \frac{counter}{T} \leq q_d + z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}}] \geq 1-\delta$. The log function is monotone, so we only have to consider the following two inequalities:

$$\log_{1-2^{-d}}(q_d - z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}}) \leq (1+\epsilon)\#F, \quad (5)$$

$$(1+\epsilon)^{-1}\#F \leq \log_{1-2^{-d}}(q_d + z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}}). \quad (6)$$

We first consider Equation (5). By substituting $\log_{1-2^{-d}} q_d$ for $\#F$, we have

$$\begin{aligned}
&\log_{1-2^{-d}}(q_d - z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}}) \leq (1+\epsilon)\log_{1-2^{-d}} q_d \\
\Leftrightarrow\ & q_d - z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}} \geq q_d^{(1+\epsilon)} \\
\Leftrightarrow\ & q_d(1-q_d^{\epsilon}) \geq z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}} \\
\Leftrightarrow\ & T \geq (\frac{z_{1-\delta}}{q_d(1-q_d^{\epsilon})})^2 q_d(1-q_d).
\end{aligned}$$

Since $0 \leq q_d \leq 1$, we have $\sqrt{q_d(1-q_d)} \leq \frac{1}{2}$. Therefore, $T = \lceil (\frac{z_{1-\delta}}{2q_d(1-q_d^{\epsilon})})^2 \rceil \geq (\frac{z_{1-\delta}}{q_d(1-q_d^{\epsilon})})^2 q_d(1-q_d)$.

We next consider Equation (6). Similarly, we have

$$\begin{aligned}
&\log_{1-2^{-d}} (q_d + z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}}) \geq (1+\epsilon)^{-1} \log_{1-2^{-d}} q_d \\
\Leftrightarrow\ & q_d + z_{1-\delta}\sqrt{\frac{q_d(1-q_d)}{T}} \leq q_d^{1/(1+\epsilon)} \\
\Leftrightarrow\ & T \geq (\frac{z_{1-\delta}}{q_d^{1/(1+\epsilon)} - q_d})^2 q_d(1-q_d).
\end{aligned}$$

So Equation (4) implies Equations (5) and (6).

Theorem 1 shows that the result of Algorithm 2 lies in the interval $[(1+\epsilon)^{-1}\#F, (1+\epsilon)\#F]$ with probability at least $1-\delta$ when T is set to a proper value. So we focus on the possible smallest value of T in subsequent analysis.

The next two lemmas are easy to show by derivations.

Lemma 1. $\frac{z_{1-\delta}}{2x(1-x^{\epsilon})}$ *is monotone increasing and monotone decreasing in* $[(1+\epsilon)^{-\frac{1}{\epsilon}}, 1]$ *and* $[0, (1+\epsilon)^{-\frac{1}{\epsilon}}]$ *respectively.*

Lemma 2. $\frac{z_{1-\delta}}{2(x^{1/(1+\epsilon)}-x)}$ *is monotone increasing and monotone decreasing in* $[(1+\epsilon)^{-\frac{1+\epsilon}{\epsilon}}, 1]$ *and* $[0, (1+\epsilon)^{-\frac{1+\epsilon}{\epsilon}}]$ *respectively.*

Theorem 2. *If* $\#F > 5$*, then there exists a proper integer value of* d *such that* $q_d \in [0.4, 0.65]$.

Proof. Let x denote the value of $q_d = (1-\frac{1}{2^d})^{\#F}$, then we have $(1-\frac{1}{2^{d+1}})^{\#F} = (\frac{1}{2}+\frac{x^{\frac{1}{\#F}}}{2})^{\#F}$. Consider the derivation

$$\tfrac{d}{d\#F}(\tfrac{1}{2}+\tfrac{x^{\frac{1}{\#F}}}{2})^{\#F} = (\frac{1}{2}+\frac{x^{\frac{1}{\#F}}}{2})^{\#F} \ln(\frac{1}{2}+\frac{x^{\frac{1}{\#F}}}{2})\frac{x^{\frac{1}{\#F}}}{2} \ln x \frac{d}{d\#F}(\#F^{-1}).$$

Note that $(\frac{1}{2}+\frac{x^{\frac{1}{\#F}}}{2})^{\#F}$ and $\frac{x^{\frac{1}{\#F}}}{2}$ are the positive terms and $\ln(\frac{1}{2}+\frac{x^{\frac{1}{\#F}}}{2})$, $\ln x$ and $\frac{d}{d\#F}(\#F^{-1})$ are the negative terms. Therefore, the derivation is negative, i.e., $(\frac{1}{2}+\frac{x^{\frac{1}{\#F}}}{2})^{\#F}$ is monotone decreasing with respect to $\#F$. In addition, $(\frac{1}{2}+\frac{x^{\frac{1}{5}}}{2})^5$ is the upper bound when $\#F \geq 5$.

Let $x = 0.4$, then $(1-\frac{1}{2^{d+1}})^{\#F} \leq (\frac{1}{2}+\frac{0.4^{\frac{1}{5}}}{2})^5 \approx 0.65$. Since $(1-\frac{1}{2^0})^{\#F} = 0$ and $\lim_{d\to+\infty}(1-\frac{1}{2^d})^{\#F} = 1$ and $(1-\frac{1}{2^d})^{\#F}$ is continuous with respect to d, we consider the circumstances close to the interval $[0.4, 0.65]$. Assume there exists an integer σ such that $(1-\frac{1}{2^{\sigma}})^{\#F} < 0.4$ and $(1-\frac{1}{2^{\sigma+1}})^{\#F} > 0.65$. According to the intermediate value theorem, we can find a value $e > 0$ such that $(1-\frac{1}{2^{\sigma+e}})^{\#F} = 0.4$. Obviously, we have $(1-\frac{1}{2^{\sigma+e+1}})^{\#F} \leq 0.65$ which is contrary with the monotone decreasing property.

From Theorem 2 and Lemma 1 and 2, it suffices to consider the results of Equation (4) when $q_d = 0.4$ and $q_d = 0.65$. For example, $T = 22$ for $\epsilon = 0.8$ and $\delta = 0.2$, $T = 998$ for $\epsilon = 0.1$ and $\delta = 0.1$, etc. We therefore pre-computed a table of the value of T. The proof of next theorem is omitted.

Theorem 3. *There exists an integer* d *such that* $q_d < 0.05$ *and* $q_{d+7} > 0.95$.

Let $depth$ denote the result of `GetDepth` in Algorithm 2. Then F_d is unsatisfiable only if $d \geq depth$. Theorem 3 shows that there exists an integer d such that $\Pr(depth < d) < 0.05$ and $\Pr(depth < d + 7) > 0.95$, i.e., $\Pr(d \leq depth \leq d + 7) > 0.9$. So in most cases, the value of $depth$ lies in an interval $[d, d + 7]$. Also, it is easy to see that $\log_2 \#F$ lies in this interval as well. The following theorem is obvious now.

Theorem 4. *Algorithm 2 runs in time linear in* $\log_2 \#F$ *relative to an NP-oracle.*

7 Evaluation

To evaluate the performance and effectiveness of our approach, two prototype implementations `STAC_CNF` and `STAC_BV` with dynamic stopping criterion for propositional logic formulas and SMT(BV) formulas are built respectively. We considered a wide range of benchmarks from different domains: grid networks, plan recognition, DQMR networks, Langford sequences, circuit synthesis, random 3-CNF, logistics problems and program synthesis [29, 24, 9, 7] [6]. For lack of space, we only list a part of results here. All our experiments were conducted on a single core of an Intel Xeon 2.40GHz (16 cores) machine with 32GB memory and CentOS6.5 operating system.

7.1 Quality of Approximation

Recall that our approach is based on Equation (1) which has not been proved. So we would like to see whether the approximation fits the bound. We experimented 100 times on each instance.

In Table 1, column 1 gives the instance name, column 2 the number of Boolean variables n, column 3 the exact counts $\#F$, and column 4 the interval $[1.8^{-1}\#F, 1.8\#F]$. The frequencies of approximations that lie in the interval $[1.8^{-1}\#F, 1.8\#F]$ in 100 times of experiments are presented in column 5. The average time consumptions, average number of iterations, and average number of SAT query invocations are presented in column 6, 7 and 8 respectively, which also indicate the advantages of our approach.

Under the dynamic stopping criterion, the counts returned by our approach should lie in an interval $[1.8^{-1}\#F, 1.8\#F]$ with probability 80% for $\epsilon = 0.8$ and $\delta = 0.2$. The statistical results in Table 1 show that the frequencies are around 80 for 100-times experiments which fit the 80% probability. The average number of iterations $\bar{T}$ listed in Table 1 is smaller than the loop termination criterion $T = 22$ which is obtained via Formula 4, indicating that the dynamic stopping technique significantly improves the

[6] Our tools `STAC_CNF` and `STAC_BV` and the suite of benchmarks are available at `https://github.com/bearben/STAC`

Table 1. Statistical results of 100-times experiments on `STAC_CNF` ($\epsilon = 0.8, \delta = 0.2$)

Instance	n	$\#F$	$[1.8^{-1}\#F, 1.8\#F]$	Freq.	$\bar{t}$ (s)	$\bar{T}$	$\bar{Q}$
special-1	20	1.0×10^6	$[5.8 \times 10^5, 1.9 \times 10^6]$	82	0.3	12.2	86.7
special-2	20	1	$[0.6, 1.8]$	86	0.6	12.6	37.6
special-3	25	3.4×10^7	$[1.9 \times 10^7, 6.0 \times 10^7]$	82	11.2	11.8	90.1
5step	177	8.1×10^4	$[4.5 \times 10^4, 1.5 \times 10^5]$	90	0.1	11.9	80.5
blockmap_05_01	1411	6.4×10^2	$[3.6 \times 10^2, 1.2 \times 10^3]$	84	1.1	12.0	73.8
blockmap_05_02	1738	9.4×10^6	$[5.2 \times 10^6, 1.7 \times 10^7]$	89	12.7	11.8	87.7
blockmap_10_01	11328	2.9×10^6	$[1.6 \times 10^6, 5.2 \times 10^6]$	83	80.3	12.0	85.0
fs-01	32	7.7×10^2	$[4.3 \times 10^2, 1.4 \times 10^3]$	80	0.02	12.6	76.2
or-50-10-10-UC-20	100	3.7×10^6	$[2.0 \times 10^6, 6.6 \times 10^6]$	77	7.7	12.0	86.1
or-60-10-10-UC-40	120	3.4×10^6	$[1.9 \times 10^6, 6.1 \times 10^6]$	91	3.5	12.1	86.0

efficiency. In addition, the values of $\bar{T}$ appear to be stable for different instances, hinting that there exists a constant upper bound on T which is irrelevant to instances.

Intuitively, our approach may start to fail on "loose" formulas, i.e., with an "infinitesimal" fraction of non-models. Instance *special-1* and *special-3* are such "loose" formulas where *special-1* has 2^{20} models with only 20 variables and *special-3* has $2^{25} - 1$ models with 25 variables. Instance *special-2* is another extreme case which only has one model. The results in Table 1 demonstrate that `STAC_CNF` also works fine on these extreme cases.

Table 2. Statistical results of 100-times experiments on `STAC_CNF` ($\epsilon = 0.2, \delta = 0.1$)

Instance	n	$\#F$	$[1.2^{-1}\#F, 1.2\#F]$	Freq.	$\bar{t}$ (s)	$\bar{T}$	$\bar{Q}$
special-1	20	1.0×10^6	$[8.7 \times 10^5, 1.3 \times 10^6]$	86	4.0	179	1023
special-2	20	1	$[0.8, 1.2]$	91	0.1	179	540
special-3	25	3.4×10^7	$[2.8 \times 10^7, 4.0 \times 10^7]$	91	138	178	1029
5step	177	8.1×10^4	$[6.8 \times 10^4, 9.8 \times 10^5]$	96	1.9	190	1078
blockmap_05_01	1411	6.4×10^2	$[5.3 \times 10^2, 7.7 \times 10^2]$	94	17.1	190	1069
blockmap_05_02	1738	9.4×10^6	$[7.9 \times 10^6, 1.1 \times 10^7]$	87	281	193	1088
blockmap_10_01	11328	2.9×10^6	$[2.4 \times 10^6, 3.5 \times 10^6]$	93	1371	180	1034
fs-01	32	7.7×10^2	$[6.4 \times 10^2, 9.2 \times 10^2]$	91	0.1	172	975
or-50-10-10-UC-20	100	3.7×10^6	$[3.1 \times 10^6, 4.4 \times 10^6]$	90	140	166	925
or-60-10-10-UC-40	120	3.4×10^6	$[2.8 \times 10^6, 4.1 \times 10^6]$	92	66	167	949

We considered another pair of parameters $\epsilon = 0.2, \delta = 0.1$. Then the interval should be $[1.2^{-1}\#F, 1.2\#F]$ and the probability should be 90%. Table 2 shows the results on such parameter setting. The frequencies that the approximation lies in interval $[1.2^{-1}\#F, 1.2\#F]$ are all around or over 90 which fits the 90% probability.

We also conducted 100-times experiments on SMT(BV) problems and the results show that `STAC_BV` is also promising. Table 3 similarly shows the results of 100-times experiments on `STAC_BV`. Its column 2 gives the sum of widths of all bit-vector variables (Boolean variable is counted as a bit-vector of width 1) instead. The statistical results demonstrate that the dynamic stopping criterion is also promising on SMT(BV) problems.

Table 3. Statistical results of 100-times experiments on STAC_BV ($\epsilon = 0.8, \delta = 0.2$)

Instance	TB.	$\#F$	$[1.8^{-1}\#F, 1.8\#F]$	Freq.	$\bar{t}$ (s)	$\bar{T}$	$\bar{Q}$
FINDpath1	32	4.1×10^6	$[2.3 \times 10^6, 7.3 \times 10^6]$	83	27.5	12.4	88.0
queue	16	8.4×10	$[4.7 \times 10, 1.5 \times 10^2]$	75	1.7	12.0	70.6
getopPath2	24	8.1×10^3	$[4.5 \times 10^3, 1.5 \times 10^4]$	88	2.7	12.2	79.5
coloring_4	32	1.8×10^9	$[1.0 \times 10^9, 3.3 \times 10^9]$	76	51.9	12.0	96.1
FISCHER2-7-fair	240	3.0×10^4	$[1.7 \times 10^4, 5.4 \times 10^4]$	79	149	11.8	79.8
case2	24	4.2×10^6	$[2.3 \times 10^6, 7.6 \times 10^6]$	79	16.5	12.4	89.3
case4	16	3.3×10^4	$[1.8 \times 10^4, 5.9 \times 10^4]$	87	2.2	12.5	85.2
case7	18	1.3×10^5	$[7.3 \times 10^4, 2.4 \times 10^5]$	83	2.9	12.4	84.1
case8	24	8.4×10^6	$[4.7 \times 10^6, 1.5 \times 10^7]$	82	14.4	12.1	91.1
case11	15	1.6×10^4	$[9.1 \times 10^3, 2.9 \times 10^4]$	76	2.1	12.0	81.2

7.2 Performance Comparison with (ϵ, δ)-counters

We compared our tools with ApproxMC2 [10] and SMTApproxMC [7] which are hashing-based (ϵ, δ)-counters. Both STAC_CNF and ApproxMC2 use CryptoMiniSAT [31], an efficient SAT solver designed for XOR clauses. STAC_BV and SMTApproxMC use the state-of-the-art SMT(BV) solver Boolector [5].

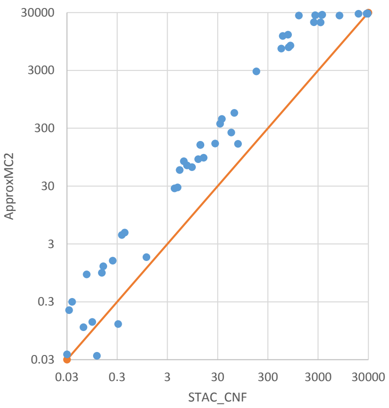

Fig. 1. Performance comparison between STAC_CNF and ApproxMC2

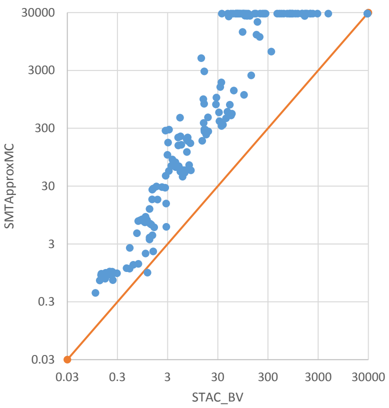

Fig. 2. Performance comparison between STAC_BV and SMTApproxMC

We first conducted experiments with $\epsilon = 0.8, \delta = 0.2$ and 8 hours timeout which are also used in evaluation in previous works [10, 7]. Figure 1 presents a comparison on performance between STAC_CNF and ApproxMC2. Each point represents an instance, whose x-coordinate and y-coordinate are the running times of STAC_CNF and ApproxMC2 on this instance, respectively. The figure is in logarithmic coordinates and demonstrates that STAC_CNF outperforms ApprxMC2 by about one order of magnitude. Figure 2 presents a similar comparison on performance between STAC_BV and SMTApproxMC, showing that STAC_BV outperforms SMTApproxMC by one or two orders of magnitude. Furthermore, the advantage enlarges as the scale grows.

Table 4. Performance comparison between STAC_CNF and ApproxMC2 with different pairs of (ϵ, δ) parameters

Instance / (ϵ, δ)		blockmap			fs-01	5step	ran5	ran6	ran7
		05_01	05_02	10_01					
$(0.8, 0.3)$	Time Ratio	1.11	3.99	1.22	3.00	3.83	6.53	8.24	5.57
	#Calls Ratio	22.60	39.02	17.91	19.12	23.11	22.53	21.28	23.68
$(0.8, 0.2)$	Time Ratio	1.84	6.16	2.44	2.80	6.05	9.61	15.41	7.37
	#Calls Ratio	26.70	34.68	25.16	33.46	27.24	33.35	38.22	30.94
$(0.8, 0.1)$	Time Ratio	2.27	7.36	3.72	5.25	12.62	9.60	9.54	8.19
	#Calls Ratio	44.88	48.26	40.01	49.40	43.03	46.12	44.84	52.63
$(0.4, 0.3)$	Time Ratio	0.75	1.37	0.42	3.00	5.04	1.97	2.31	2.74
	#Calls Ratio	17.75	36.20	14.69	16.40	27.63	21.07	27.34	21.63
$(0.4, 0.2)$	Time Ratio	0.77	1.44	0.86	4.50	7.70	2.82	1.77	3.02
	#Calls Ratio	20.91	26.35	29.16	26.72	40.66	26.49	27.82	28.94
$(0.4, 0.1)$	Time Ratio	1.08	2.57	1.29	4.90	7.09	3.84	3.43	3.11
	#Calls Ratio	37.16	46.28	39.40	31.99	39.36	41.02	35.88	34.11
$(0.2, 0.3)$	Time Ratio	0.42	0.47	0.23	5.08	3.79	1.26	1.14	1.81
	#Calls Ratio	13.75	20.82	24.35	13.37	19.74	25.20	19.19	20.06
$(0.2, 0.2)$	Time Ratio	0.57	0.92	0.26	8.42	3.37	2.07	1.50	2.45
	#Calls Ratio	21.80	29.62	25.60	21.83	21.59	25.88	22.72	22.98
$(0.2, 0.1)$	Time Ratio	0.87	0.92	0.44	16.69	3.17	3.61	2.27	2.60
	#Calls Ratio	27.86	29.91	33.36	34.17	31.58	40.81	29.01	29.90

Table 4 presents more experimental results with (ϵ, δ) parameters other than $(\epsilon = 0.8, \delta = 0.2)$. Nine pairs of parameters were experimented. "Time Ratio" represents the ratio of the running times of ApproxMC2 to STAC_CNF. "#Calls Ratio" represents the ratio of the number of SAT calls of ApproxMC2 to STAC_CNF. The results show that ApproxMC2 gains advantage as ϵ decreases and STAC_CNF gains advantage as δ decreases. On the whole, ApproxMC2 gains advantage when ϵ and δ both decrease. Note that the numbers of SAT calls represent the complexity of both algorithms. In Table 4, #Calls Ratio is more stable than Time Ratio among different pairs of parameters and also different instances. It indicates that the difficulty of NP-oracle is also an important factor of running time performance.

8 Conclusion

In this paper, we propose a new hashing-based approximate algorithm with dynamic stopping criterion. Our approach has two key strengths: it requires only one satisfiability query for each cut, and it terminates once meeting the criterion of accuracy. We implemented prototype tools for propositional logic formulas and SMT(BV) formulas. Extensive experiments demonstrate that our approach is efficient and promising. Despite that we are unable to prove the correctness of Equation (1), the experimental results fit it quite well. This phenomenon might be caused by some hidden properties of the hash functions. To fully understand these functions and their correlation with the model count of the hashed formula might be an interesting problem to the community. In addition, extending the idea in this paper to count solutions of other formulas is also a direction of future research.

References

1. D. Achlioptas and P. Theodoropoulos. Probabilistic model counting with short xors. In *Proc. of SAT*, pages 3–19, 2017.
2. M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of NP-witnesses using an NP-oracle. *Inf. Comput.*, 163(2):510–526, 2000.
3. V. Belle, G. V. Broeck, and A. Passerini. Hashing-based approximate probabilistic inference in hybrid domains. In *Proc. of UAI*, pages 141–150, 2015.
4. L. D. Brown, T. T. Cai, and A. Dasgupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001.
5. R. Brummayer and A. Biere. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Proc. of TACAS*, pages 174–177, 2009.
6. S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proc of AAAI*, pages 1722–1730, 2014.
7. S. Chakraborty, K. S. Meel, R. Mistry, and M. Y. Vardi. Approximate probabilistic inference via word-level counting. In *Proc. of AAAI*, pages 3218–3224, 2016.
8. S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable and nearly uniform generator of SAT witnesses. In *Proc. of CAV*, pages 608–623, 2013.
9. S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable approximate model counter. In *Proc. of CP*, pages 200–216, 2013.
10. S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. of IJCAI*, pages 3569–3576, 2016.
11. M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.
12. D. Chistikov, R. Dimitrova, and R. Majumdar. Approximate counting in SMT and value estimation for probabilistic programs. In *Proc. of TACAS*, pages 320–334, 2015.
13. C. Domshlak and J. Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *J. Artif. Intell. Res. (JAIR)*, 30:565–620, 2007.
14. S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Embed and project: Discrete sampling with universal hashing. In *Advances in Neural Information Processing Systems 26*, pages 2085–2093, 2013.
15. A. Filieri, C. S. Pasareanu, and W. Visser. Reliability analysis in symbolic pathfinder: A brief summary. In *Proc. of ICSE*, pages 39–40, 2014.

16. A. Filieri, C. S. Pasareanu, and G. Yang. Quantification of software changes through probabilistic symbolic execution (N). In *Proc. of ASE*, pages 703–708, 2015.
17. M. Fredrikson and S. Jha. Satisfiability modulo counting: a new approach for analyzing privacy properties. In *Proc. of CSL-LICS*, pages 42:1–42:10, 2014.
18. J. Geldenhuys, M. B. Dwyer, and W. Visser. Probabilistic symbolic execution. In *Proc. of ISSTA*, pages 166–176, 2012.
19. K. Gleissenthall, B. Köpf, and A. Rybalchenko. Symbolic polytopes for quantitative interpolation and verification. In *Proc. of CAV*, pages 178–194, 2015.
20. C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *Proc. of AAAI*, pages 54–61, 2006.
21. C. P. Gomes, A. Sabharwal, and B. Selman. Near-uniform sampling of combinatorial spaces using XOR constraints. In *Advances in Neural Information Processing Systems 19*, pages 481–488, 2006.
22. A. Ivrii, S. Malik, K. S. Meel, and M. Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, 21(1):41–58, 2016.
23. R. M. Karp, M. Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.
24. L. Kroc, A. Sabharwal, and B. Selman. Leveraging belief propagation, backtrack search, and statistics for model counting. *Annals of OR*, 184(1):209–231, 2011.
25. S. Liu and J. Zhang. Program analysis: from qualitative analysis to quantitative analysis. In *Proc. of ICSE*, pages 956–959, 2011.
26. K. S. Meel, M. Y. Vardi, S. Chakraborty, D. J. Fremont, S. A. Seshia, D. Fried, A. Ivrii, and S. Malik. Constrained sampling and counting: Universal hashing meets SAT solving. In *Proceedings of Workshop on Beyond NP(BNP)*, 2016.
27. Q. Phan, P. Malacaria, C. S. Pasareanu, and M. d'Amorim. Quantifying information leaks using reliability analysis. In *Proc. of SPIN*, pages 105–108, 2014.
28. D. Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.
29. T. Sang, P. Beame, and H. A. Kautz. Performing bayesian inference by weighted model counting. In *Proc. of AAAI*, pages 475–482, 2005.
30. M. Sipser. A complexity theoretic approach to randomness. In *Proc. of the 15th Annual ACM Symposium on Theory of Computing*, pages 330–335, 1983.
31. M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In *Proc. of SAT*, pages 244–257, 2009.
32. L. J. Stockmeyer. The complexity of approximate counting (preliminary version). In *Proc. of the 15th Annual ACM Symposium on Theory of Computing*, pages 118–126, 1983.
33. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
34. S. Wallis. Binomial confidence intervals and contingency tests: Mathematical fundamentals and the evaluation of alternative methods. *Journal of Quantitative Linguistics*, 20(3):178–208, 2013.
35. E. B. Wilson. Probable inference, the law of succession and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.