Model Counting meets F_0 Estimation*

A. Pavan ⓡ
pavan@cs.iastate.edu
Iowa State University

N. V. Vinodchandran ⓡ
vinod@cse.unl.edu
University of Nebraska, Lincoln

Arnab Bhattacharyya ⓡ
arnabb@nus.edu.sg
National University of Singapore

Kuldeep S. Meel
meel@comp.nus.edu.sg
National University of Singapore

ABSTRACT

Constraint satisfaction problems (CSP's) and data stream models are two powerful abstractions to capture a wide variety of problems arising in different domains of computer science. Developments in the two communities have mostly occurred independently and with little interaction between them. In this work, we seek to investigate whether bridging the seeming communication gap between the two communities may pave the way to richer fundamental insights. To this end, we focus on two foundational problems: model counting for CSP's and computation of zeroth frequency moments (F_0) for data streams.

Our investigations lead us to observe striking similarity in the core techniques employed in the algorithmic frameworks that have evolved separately for model counting and F_0 computation. We design a recipe for translation of algorithms developed for F_0 estimation to that of model counting, resulting in new algorithms for model counting. We then observe that algorithms in the context of distributed streaming can be transformed to distributed algorithms for model counting. We next turn our attention to viewing streaming from the lens of counting and show that framing F_0 estimation as a special case of #DNF counting allows us to obtain a general recipe for a rich class of streaming problems, which had been subjected to case-specific analysis in prior works. In particular, our view yields a state-of-the art algorithm for multidimensional range efficient F_0 estimation with a simpler analysis.

CCS CONCEPTS

• **Theory of computation** → **Streaming models**; **Sketching and sampling**.

KEYWORDS

Model Counting, Streaming Algorithms, F_0-computation, DNF Counting

ACM Reference Format:
A. Pavan ⓡ, N. V. Vinodchandran ⓡ, Arnab Bhattacharyya ⓡ, and Kuldeep S. Meel. 2021. Model Counting meets F_0 Estimation. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3452021.3458311

1 INTRODUCTION

Constraint Satisfaction Problems (CSP's) and the *data stream model* are two core themes in computer science with a diverse set of applications, ranging from probabilistic reasoning, networks, databases, verification, and the like. *Model counting* and computation of *zeroth frequency moment* (F_0) are fundamental problems for CSP's and the data stream model respectively. This paper is motivated by our observation that despite the usage of similar algorithmic techniques for the two problems, the developments in the two communities have, surprisingly, evolved separately, and rarely has a paper from one community been cited by the other.

Given a set of constraints φ over a set of variables in a finite domain $\mathcal{D}$, the problem of model counting is to estimate the number of solutions of φ. We are often interested when φ is restricted to a special class of representations such as Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF). A data stream over a domain $[N]$ is represented by $\mathbf{a} = a_1, a_2, \cdots a_m$ wherein each item $a_i \subseteq [N]$. The *zeroth frequency moment*, denoted as F_0, of $\mathbf{a}$ is the number of distinct elements appearing in $\mathbf{a}$, i.e., $|\cup_i a_i|$ (traditionally, a_is are singletons; we will also be interested in the case when a_is are sets). The fundamental nature of model counting and F_0 computation over data streams has led to intense interest from theoreticians and practitioners alike in the respective communities for the past few decades.

The starting point of this work is the confluence of two viewpoints. The first viewpoint contends that some of the algorithms for model counting can conceptually be thought of as operating on the stream of the solutions of the constraints. The second viewpoint contends that a stream can be viewed as a DNF formula, and the problem of F_0 estimation is similar to model counting. These viewpoints make it natural to believe that algorithms developed in the streaming setting can be directly applied to model counting, and vice versa. We explore this connection and indeed, design new algorithms for model counting inspired by algorithms for estimating F_0 in data streams. By exploring this connection further, we

*The authors decided to forgo the convention of alphabetical ordering of names 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 with confirmation code: XiQE7V3pKq_A. For citation of the work, authors request that the citation guidelines by AEA (available at https://www.aeaweb.org/journals/policies/random-author-order) for random author ordering be followed.

PODS '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8381-3/21/06…$15.00
https://doi.org/10.1145/3452021.3458311

design new algorithms to estimate F_0 for streaming sets that are succinctly represented by constraints. To put our contributions in context, we briefly survey the historical development of algorithmic frameworks in both model counting and F_0 estimation and point out the similarities.

Model Counting

The complexity-theoretic study of model counting was initiated by Valiant who showed that this problem, in general, is #P-complete [59]. This motivated researchers to investigate approximate model counting and in particular achieving (ε, δ)-approximation schemes. The complexity of approximate model counting depends on its representation. When the model φ is represented as a CNF formula φ, designing an efficient (ε, δ)-approximation is NP-hard [55]. In contrast, when it is represented as a DNF formula, model counting admits FPRAS (fully polynomial-time approximation scheme) [38, 39]. We will use #CNF to refer to the case when φ is a CNF formula while #DNF to refer to the case when φ is a DNF formula.

For #CNF, Stockmeyer [55] provided a hashing-based randomized procedure that can compute (ε, δ)-approximation within time polynomial in $|\varphi|, \varepsilon, \delta$, given access to an NP oracle. Building on Stockmeyer's approach and motivated by the unprecedented breakthroughs in the design of SAT solvers, researchers have proposed a series of algorithmic improvements that have allowed the hashing-based techniques for approximate model counting to scale to formulas involving hundreds of thousands of variables [2, 13, 14, 16, 23, 31, 35, 52, 53]. The practical implementations substitute NP oracle with SAT solvers. In the context of model counting, we are primarily interested in time complexity and therefore, the number of NP queries is of key importance. The emphasis on the number of NP calls also stems from practice as the practical implementation of model counting algorithms have shown to spend over 99% of their time in the underlying SAT calls [53].

Karp and Luby [38] proposed the first FPRAS scheme for #DNF, which was subsequently improved in the follow-up works [22, 39]. Chakraborty, Meel, and Vardi [14] demonstrated that the hashing-based framework can be extended to #DNF, hereby providing a unified framework for both #CNF and #DNF. Meel, Shrotri, and Vardi [44–46] subsequently improved the complexity of the hashing-based approach for #DNF and observed that hashing-based techniques achieve better scalability than that of Monte Carlo techniques.

Zeroth Frequency Moment Estimation

Estimating (ε, δ)-approximation of the k^{th} frequency moments (F_k) is a central problem in the data streaming model [3]. In particular, considerable work has been done in designing algorithms for estimating the 0^{th} frequency moment (F_0), the number of distinct elements in the stream. While designing streaming algorithms, the primary resource concerns are two-fold: space complexity and processing time per element. For an algorithm to be considered efficient, these should be poly$(\log N, 1/\epsilon)$ where N is the size of the universe [1].

The first algorithm for computing F_0 with a constant factor approximation was proposed by Flajolet and Martin, who assumed the existence of hash functions with ideal properties resulting in an algorithm with undesirable space complexity [29]. In their seminal work, Alon, Matias, and Szegedy designed an $O(\log N)$ space algorithm for F_0 with a constant approximation ratio that employs 2-universal hash functions [3]. Subsequent investigations into hashing-based schemes by Gibbons and Tirthapura [30] and Bar-Yossef, Kumar, and Sivakumar [8] provided (ε, δ)-approximation algorithms with space and time complexity $\log N \cdot \text{poly}(\frac{1}{\varepsilon})$. Subsequently, Bar-Yossef et al. proposed *three algorithms* with improved space and time complexity [7]. While the three algorithms employ hash functions, they differ conceptually in the usage of relevant random variables for the estimation of F_0. This line of work resulted in the development of an algorithm with optimal space complexity $O(\log N + \frac{1}{\varepsilon^2})$ and $O(\log N)$ update time [37].

The above-mentioned works are in the setting where each data item a_i is an element of the universe. Subsequently, there has been a series of results of estimating F_0 in rich scenarios with particular focus to handle the cases $a_i \subseteq [N]$ such as a list or a multidimensional range [8, 47, 56, 58].

The Road to a Unifying Framework

As mentioned above, the algorithmic developments for model counting and F_0 estimation have largely relied on the usage of hashing-based techniques and yet these developments have, surprisingly, been separate, and rarely has a work from one community been cited by the other. In this context, we wonder whether it is possible to bridge this gap and if such an exercise would contribute to new algorithms for model counting as well as for F_0 estimation? The main conceptual contribution of this work is an affirmative answer to the above question. First, we point out that the two well-known algorithms; Stockmeyer's #CNF algorithm [55] that is further refined by Chakraborty et. al. [14] and Gibbons and Tirthapura's F_0 estimation algorithm [30], are essentially the same.

The core idea of the hashing-based technique of Stockmeyer's and Chakraborty et al's scheme is to use pairwise independent hash functions to partition the solution space (satisfying assignments of a CNF formula) into *roughly equal and small* cells, wherein a cell is *small* if the number of solutions is less than a pre-computed threshold, denoted by Thresh. Then a good estimate for the number of solutions is the *number of solutions in an arbitrary cell* $\times$ *number of cells*. To partition the solution space, pairwise independent hash functions are used. To determine the appropriate number of cells, the solution space is iteratively partitioned as follows. At the m^{th} iteration, a hash function with range $\{0,1\}^m$ is considered resulting in cells $h^{-1}(y)$ for each $y \in \{0,1\}^m$. An NP oracle can be employed to check whether a particular cell (for example $h^{-1}(0^m)$) is small by enumerating solutions one by one until we have either obtained Thresh+1 number of solutions or we have exhaustively enumerated all the solutions. If the the cell $h^{-1}(0^m)$ is small, then the algorithm outputs $t \times 2^m$ as an estimate where t is the number of solutions in the cell $h^{-1}(0^m)$. If the cell $h^{-1}(0^m)$ is not small, then the algorithm moves on to the next iteration where a hash function with range $\{0,1\}^{m+1}$ is considered.

We now describe Gibbons and Tirthapura's algorithm for F_0 estimation which we call the Bucketing algorithm. We will assume the universe $[N] = \{0,1\}^n$. The algorithm maintains a bucket of size

[1] We ignore $O(\log \frac{1}{\delta})$ factor in this discussion

Thresh and starts by picking a hash function $h : \{0,1\}^n \to \{0,1\}^n$. It iterates over sampling levels. At level m, when a data item x comes, if $h(x)$ starts with 0^m, then x is added to the bucket. If the bucket overflows, then the sampling level is increased to $m+1$ and all elements x in the bucket other than the ones with $h(x) = 0^{m+1}$ are deleted. At the end of the stream, the value $t \times 2^m$ is output as the estimate where t is the number of elements in the bucket and m is the sampling level.

These two algorithms are conceptually the same. In the Bucketing algorithm, at the sampling level m, it looks at only the first m bits of the hashed value; this is equivalent to considering a hash function with range $\{0,1\}^m$. Thus the bucket is nothing but all the elements in the stream that belong to the cell $h^{-1}(0^m)$. The final estimate is the number of elements in the bucket times the number of cells, identical to Chakraborty et. al's algorithm. In both algorithms, to obtain an (ε, δ) approximation, the Thresh value is chosen as $O(\frac{1}{\varepsilon^2})$ and the median of $O(\log \frac{1}{\delta})$ independent estimations is output.

Our Contributions

Motivated by the conceptual identity between the two algorithms, we further explore the connections between algorithms for model counting and F_0 estimation.

(1) We formalize a recipe to transform streaming algorithms for F_0 estimation to those for model counting. Such a transformation yields new (ε, δ)-approximate algorithms for model counting, which are different from currently known algorithms. Recent studies in the fields of automated reasoning have highlighted the need for diverse approaches [63], and similar studies in the context of #DNF provided strong evidence to the power of diversity of approaches [45]. In this context, these newly obtained algorithms open up several new interesting directions of research ranging from the development of MaxSAT solvers with native XOR support to open problems in designing FPRAS schemes.
(2) Given the central importance of #DNF (and its weighted variant) due to a recent surge of interest in scalable techniques for provenance in probabilistic databases [49, 50], a natural question is whether one can design efficient techniques in the distributed setting. In this work, we initiate the study of distributed #DNF. We then show that the transformation recipe from F_0 estimation to model counting allows us to view the problem of the design of distributed #DNF algorithms through the lens of *distributed functional monitoring* that is well studied in the data streaming literature.
(3) Building upon the connection between model counting and F_0 estimation, we design new algorithms to estimate F_0 over *structured set streams* where each element of the stream is a (succinct representation of a) subset of the universe. Thus, the stream is $S_1, S_2, \cdots$ where each $S_i \subseteq [N]$ and the goal is to estimate the F_0 of the stream, i.e. size of $\cup_i S_i$. In this scenario, a traditional F_0 streaming algorithm that processes each element of the set incurs high per-item processing time-complexity and is inefficient. Thus the goal is to design algorithms whose per-item time (time to process each S_i) is polylogarithmic in the size of the universe. Structured set streams that are considered in the literature include 1-dimensional and multidimensional ranges [47, 58]. Several interesting problems such as max-dominance norm [19], counting triangles in graphs [8], and distinct summation problem [17] can be reduced to computing F_0 over such ranges.

    We observe that several structured sets can be represented as small DNF formulae and thus F_0 counting over these structured set data streams can be viewed as a special case of #DNF. Using the hashing-based techniques for #DNF, we obtain a general recipe for a rich class of structured sets that include multidimensional ranges, multidimensional arithmetic progressions, and affine spaces. Prior work on single and multidimensional ranges [2] had to rely on involved analysis for each of the specific instances, while our work provides a general recipe for both analysis and implementation.

Organization

We present notations and preliminaries in Section 2. We then present the transformation of F_0 estimation to model counting in Section 3. We then focus on distributed #DNF in Section 4. We then present the transformation of model counting algorithms to structured set streaming algorithms in Section 5. We conclude in Section 6 with a discussion of future research directions.

We would like to emphasize that the primary objective of this work is to provide a unifying framework for F_0 estimation and model counting. Therefore, when designing new algorithms based on the transformation recipes, we intentionally focus on conceptually cleaner algorithms and leave potential improvements in time and space complexity for future work.

2 NOTATION

We will use assume the universe $[N] = \{0,1\}^n$. We write $\Pr[\mathcal{Z} : \Omega]$ to denote the probability of outcome $\mathcal{Z}$ when sampling from a probability space Ω. For brevity, we omit Ω when it is clear from the context.

F_0 Estimation. A data stream $\mathbf{a}$ over domain $[N]$ can be represented as $\mathbf{a} = a_1, a_2, \ldots a_m$ wherein each item $a_i \in [N]$. Let $\mathbf{a}_u = \cup_i \{a_i\}$. F_0 of the stream $\mathbf{a}$ is $|\mathbf{a}_u|$. We are often interested in a a *probably approximately correct* scheme that returns an (ε, δ)-*estimate* c, i.e.,

$$\Pr\left[\frac{|\mathbf{a}_u|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathbf{a}_u|\right] \geq 1-\delta$$

Model Counting. Let $\{x_1, x_2, \ldots x_n\}$ be a set of Boolean variables. For a Boolean formula φ, let $\mathsf{Vars}(\varphi)$ denote the set of variables appearing in φ. Throughout the paper, unless otherwise stated, we will assume that the relationship $n = |\mathsf{Vars}(\varphi)|$ holds. We denote the set of all satisfying assignments of φ by $\mathsf{Sol}(\varphi)$.

The *propositional model counting problem* is to compute $|\mathsf{Sol}(\varphi)|$ for a given formula φ. A *probably approximately correct* (or PAC) counter is a probabilistic algorithm $\mathsf{ApproxCount}(\cdot,\cdot,\cdot)$ that takes as inputs a formula φ, a tolerance $\varepsilon > 0$, and a confidence $\delta \in (0,1]$, and returns a (ε, δ)-estimate c, i.e.,

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

[2] Please refer to Remark 1 in Section 5 for a discussion on the earlier work on multidimensional ranges [58].

PAC guarantees are also sometimes referred to as (ε, δ)-guarantees. We use #CNF (resp. #DNF) to refer to the model counting problem when φ is represented as CNF (resp. DNF).

k-wise Independent hash functions. Let $n, m \in \mathbb{N}$ and $\mathcal{H}(n,m) \triangleq \{h : \{0,1\}^n \to \{0,1\}^m\}$ be a family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$. We use $h \xleftarrow{R} \mathcal{H}(n,m)$ to denote the probability space obtained by choosing a function h uniformly at random from $\mathcal{H}(n,m)$.

Definition 1. A family of hash functions $\mathcal{H}(n,m)$ is $k-$wise independent if $\forall \alpha_1, \alpha_2, \ldots \alpha_k \in \{0,1\}^m$, distinct $x_1, x_2, \ldots x_k \in \{0,1\}^n, h \xleftarrow{R} \mathcal{H}(n,m)$,

$$\Pr[(h(x_1) = \alpha_1) \wedge (h(x_2) = \alpha_2) \ldots (h(x_k) = \alpha_k)] = \frac{1}{2^{km}} \quad (1)$$

We will use $\mathcal{H}_{\mathsf{k-wise}}(n,m)$ to refer to a $k-$wise independent family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$.

Explicit families. In this work, one hash family of particular interest is $\mathcal{H}_{\mathsf{Toeplitz}}(n,m)$, which is known to be 2-wise independent [10]. The family is defined as follows: $\mathcal{H}_{\mathsf{Toeplitz}}(n,m) \triangleq \{h : \{0,1\}^n \to \{0,1\}^m\}$ is the family of functions of the form $h(x) = Ax + b$ with $A \in \mathbb{F}_2^{m \times n}$ and $b \in \mathbb{F}_2^{m \times 1}$ where A is a uniformly randomly chosen Toeplitz matrix of size $m \times n$ while b is uniformly randomly matrix of size $m \times 1$. Another related hash family of interest is $\mathcal{H}_{\mathsf{xor}}(n,m)$ wherein $h(X)$ is again of the form $Ax + b$ where A and b are uniformly randomly chosen matrices of sizes $m \times n$ and $m \times 1$ respectively. Both $\mathcal{H}_{\mathsf{Toeplitz}}$ and $\mathcal{H}_{\mathsf{xor}}$ are 2-wise independent but it is worth noticing that $\mathcal{H}_{\mathsf{Toeplitz}}$ can be represented with $\Theta(n)$-bits while $\mathcal{H}_{\mathsf{xor}}$ requires $\Theta(n^2)$ bits of representation.

For every $m \in \{1, \ldots n\}$, the m^{th} prefix-slice of h, denoted h_m, is a map from $\{0,1\}^n$ to $\{0,1\}^m$, where $h_m(y)$ is the first m bits of $h(y)$. Observe that when $h(x) = Ax + b$, $h_m(x) = A_m x + b_m$, where A_m denotes the submatrix formed by the first m rows of A and b_m is the first m entries of the vector b.

3 FROM STREAMING TO COUNTING

As a first step, we present a unified view of the three hashing-based algorithms proposed in Bar-Yossef et al [7]. The first algorithm is the Bucketing algorithm discussed above with the observation that instead of keeping the elements in the bucket, it suffices to keep their hashed values. Since in the context of model counting, our primary concern is with time complexity, we will focus on Gibbons and Tirthapura's Bucketing algorithm in [30] rather than Bar-Yossef et al.'s modification. The second algorithm, which we call Minimum, is based on the idea that if we hash all the items of the stream, then $O(1/\varepsilon^2)$-th minimum of the hash valued can be used compute a good estimate of F_0. The third algorithm, which we call Estimation, chooses a set of k functions, $\{h_1, h_2, \ldots\}$, such that each h_j is picked randomly from an $O(\log(1/\varepsilon))$-independent hash family. For each hash function h_j, we say that h_j is not *lonely* if there exists $a_i \in \mathbf{a}$ such that $h_j(a_i) = 0$. One can then estimate F_0 of $\mathbf{a}$ by estimating the number of hash functions that are not lonely.

Algorithm 1, called ComputeF0, presents the overarching architecture of the three proposed algorithms. Each of these algorithms first picks an appropriate set of hash functions H and initializes the sketch $\mathcal{S}$. The architecture of ComputeF0 is fairly simple: it chooses a collection of hash functions using ChooseHashFunctions, calls the subroutine ProcessUpdate for every incoming element of the stream, and invokes ComputeEst at the end of the stream to return the F_0 approximation.

ChooseHashFunctions. As shown in Algorithm 2, the hash functions depend on the strategy being implemented. The subroutine PickHashFunctions($\mathcal{H}, t$) returns a collection of t independent hash functions from the family $\mathcal{H}$. We use H to denote the collection of hash functions returned, this collection viewed as either 1-dimensional array or as a 2-dimensional array. When H is 1-dimensional array, $H[i]$ to denote the ith hash function of the collection and when H is a 2-dimensional array $H[i][]j$ is the $[i,j]$th hash functions.

Sketch Properties. For each of the three algorithms, their corresponding sketches can be viewed as arrays of size of $35\log(1/\delta)$. The parameter Thresh is set to $96/\varepsilon^2$.

Bucketing The element $\mathcal{S}[i]$ is a tuple $\langle \ell_i, m_i \rangle$ where ℓ_i is a list of size at most Thresh, where $\ell_i = \{x \in \mathbf{a} \mid H[i]_{m_i}(x) = 0^{m_i}\}$. We use $\mathcal{S}[i](0)$ to denote ℓ_i and $\mathcal{S}[i](1)$ to denote m_i.

Minimum The element $\mathcal{S}[i]$ holds the lexicographically distinct Thresh many smallest elements of $\{H[i](x) \mid x \in \mathbf{a}\}$.

Estimation The element $\mathcal{S}[i]$ holds a tuple of size Thresh. The j'th entry of this tuple is the largest number of trailing zeros in any element of $H[i,j](\mathbf{a})$.

ProcessUpdate. For a new item x, the update of $\mathcal{S}$, as shown in Algorithm 3 is as follows:

Bucketing For a new item x, if $H[i]_{m_i}(x) = 0^{m_i}$, then we add it to $\mathcal{S}[i]$ if x is not already present in $\mathcal{S}[i]$. If the size of $\mathcal{S}[i]$ is greater than Thresh (which is set to be $O(1/\varepsilon^2)$), then we increment the m_i as in line 8.

Minimum For a new item x, if $H[i](x)$ is smaller than the max $\mathcal{S}[i]$, then we replace max $\mathcal{S}[i]$ with $H[i](x)$.

Estimation For a new item x, compute $z = \mathsf{TrailZero}(H[i,j](x))$, i.e, the number of trailing zeros in $H[i,j](x)$, and replace $\mathcal{S}[i,j]$ with z if z is larger than $\mathcal{S}[i,j]$.

ComputeEst. Finally, for each of the algorithms, we estimate F_0 based on the sketch $\mathcal{S}$ as described in the subroutine ComputeEst presented as Algorithm 4. It is crucial to note that the estimation of F_0 is performed solely using the sketch $\mathcal{S}$ for the Bucketing and Minimum algorithms. The Estimation algorithm requires an additional parameter r that depends on a loose estimate of F_0; we defer details to Section 3.4.

Algorithm 1 ComputeF0(n, ε, δ)

```
1: Thresh ← 96/ε²
2: t ← 35 log(1/δ)
3: H ← ChooseHashFunctions(n, Thresh, t)
4: S ← {}
5: while true do
6:     if EndStream then exit;
7:     x ← input()
8:     ProcessUpdate(S, H, x, Thresh)
9: Est ← ComputeEst(S, Thresh)
10: return Est
```

Algorithm 2 ChooseHashFunctions(n, Thresh, t)

switch AlgorithmType **do**
  case AlgorithmType==Bucketing
    $H \leftarrow$ PickHashFunctions($\mathcal{H}_{\text{Toeplitz}}(n, n), t$)
  case AlgorithmType==Minimum
    $H \leftarrow$ PickHashFunctions($\mathcal{H}_{\text{Toeplitz}}(n, 3n), t$)
  case AlgorithmType==Estimation
    $s \leftarrow 10\log(1/\varepsilon)$
    $H \leftarrow$ PickHashFunctions($\mathcal{H}_{s-\text{wise}}(n, n), t \times$ Thresh)
return H

Algorithm 3 ProcessUpdate($\mathcal{S}, H, x$, Thresh)

for $i \in [1, |H|]$ **do**
  switch AlgorithmType **do**
    case Bucketing
      $m_i = \mathcal{S}[i](0)$
      if $H[i]_{m_i}(x) == 0^{m_i}$ **then**
        $\mathcal{S}[i](0) \leftarrow \mathcal{S}[i](0) \cup \{x\}$
        if size($\mathcal{S}[i](0)$) > Thresh **then**
          $\mathcal{S}[i](1) \leftarrow \mathcal{S}[i](1) + 1$
          for $y \in \mathcal{S}$ **do**
            if $H[i]_{m_i+1}(y) \neq 0^{m_i+1}$ **then**
              Remove($\mathcal{S}[i](0), y$)
    case Minimum
      if size($\mathcal{S}[i]$) < Thresh **then**
        $\mathcal{S}[i]$.Append($H[i](x)$)
      else
        $j \leftarrow \arg\max(S[i])$
        if $\mathcal{S}[i](j) > H[i](x)$ **then**
          $\mathcal{S}[i](j) \leftarrow H[i](x)$
    case Estimation
      for $j \in [1,$ Thresh$]$ **do**
        $S[i, j] \leftarrow \max(S[i, j],$ TrailZero($H[i, j](x)$))
return $\mathcal{S}$

Algorithm 4 ComputeEst($\mathcal{S}$, Thresh)

switch AlgorithmType **do**
  case Bucketing
    return Median $\left(\left\{\text{size}(\mathcal{S}[i](0)) \times 2^{\mathcal{S}[i](1)}\right\}_i\right)$
  case Minimum
    return Median $\left(\left\{\frac{\text{Thresh} \times 2^m}{\max\{\mathcal{S}[i]\}}\right\}_i\right)$
  case Estimation(r)
    return Median $\left(\left\{\frac{\ln\left(1-\frac{1}{\text{Thresh}}\sum_{j=1}^{\text{Thresh}} \mathbb{1}\{\mathcal{S}[i,j] \geq r\}\right)}{\ln(1-2^{-r})}\right\}_i\right)$

3.1 A Recipe For Transformation

Observe that for each of the algorithms, the final computation of F_0 estimation depends on the sketch $\mathcal{S}$. Therefore, as long as for two streams $\mathbf{a}$ and $\hat{\mathbf{a}}$, if their corresponding sketches, say $\mathcal{S}$ and $\hat{\mathcal{S}}$ respectively, are equivalent, the three schemes presented above would return the same estimates. The recipe for a transformation of streaming algorithms to model counting algorithms is based on the following insight:

(1) Capture the relationship $\mathcal{P}(\mathcal{S}, H, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, set of hash functions H, and set $\mathbf{a}_u$ at the end of stream. Recall that $\mathbf{a}_u$ is the set of all distinct elements of the stream $\mathbf{a}$.
(2) The formula φ is viewed as symbolic representation of the unique set $\mathbf{a}_u$ represented by the stream $\mathbf{a}$ such that $\text{Sol}(\varphi) = \mathbf{a}_u$.
(3) Given a formula φ and set of hash functions H, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, H, \text{Sol}(\varphi))$ holds. And now, we can estimate $|\text{Sol}(\varphi)|$ from $\mathcal{S}$.

In the rest of this section, we will apply the above recipe to the three types of F_0 estimation algorithms, and derive corresponding model counting algorithms. In particular, we show how applying the above recipe to the Bucketing algorithm leads us to reproduce the state of the art hashing-based model counting algorithm, ApproxMC, proposed by Chakraborty et al [14]. Applying the above recipe to Minimum and Estimation allows us to obtain fundamentally different schemes. In particular, we observe while model counting algorithms based on Bucketing and Minimum provide FPRAS's when φ is DNF, such is not the case for the algorithm derived based on Estimation.

3.2 Bucketing-based Algorithm

The Bucketing algorithm chooses a set H of pairwise independent hash functions and maintains a sketch $\mathcal{S}$ that we will describe. Here we use $\mathcal{H}_{\text{Toeplitz}}$ as our choice of pairwise independent hash functions. The sketch $\mathcal{S}$ is an array where, each $\mathcal{S}[i]$ of the form $\langle c_i, m_i \rangle$. We say that the relation $\mathcal{P}_1(\mathcal{S}, H, \mathbf{a}_u)$ holds if

(1) $|\mathbf{a}_u \cap \{x \mid H[i]_{m_i-1}(x) = 0^{m_i-1}\}| \geq \frac{96}{\varepsilon^2}$
(2) $c_i = |\mathbf{a}_u \cap \{x \mid H[i]_{m_i}(x) = 0^{m_i}\}| < \frac{96}{\varepsilon^2}$

The following lemma due to Bar-Yossef *et al.* [7] and Gibbons and Tirthapura [30] captures the relationship among the sketch $\mathcal{S}$, the relation $\mathcal{P}_1$ and the number of distinct elements of a multiset.

Lemma 1. *[7, 30] Let $\mathbf{a} \subseteq \{0,1\}^n$ be a multiset and $H \subseteq \mathcal{H}_{\text{Toeplitz}}(n, n)$ where and each $H[i]$s are independently drawn and $|H| = O(\log 1/\delta)$ and let $\mathcal{S}$ be such that the $\mathcal{P}_1(\mathcal{S}, H, a_u)$ holds. Let $c = \text{Median}\ \{c_i \times 2^{m_i}\}_i$. Then*

$$\Pr\left[\frac{|\mathbf{a}_u|}{(1+\varepsilon)} \leq c \leq (1+\varepsilon)|\mathbf{a}_u|\right] \geq 1 - \delta.$$

To design an algorithm for model counting, based on the bucketing strategy, we turn to the subroutine introduced by Chakraborty, Meel, and Vardi: BoundedSAT, whose properties are formalized as follows:

Proposition 1. *[13, 14] There is an algorithm* BoundedSAT *that gets φ over n variables, a hash function $h \in \mathcal{H}_{\text{Toeplitz}}(n, m)$, and a number p as inputs, returns* $\min(p, |\text{Sol}(\varphi \wedge h(x) = 0^m)|)$. *If φ is a CNF formula, then* BoundedSAT *makes $O(p)$ calls to a NP oracle. If φ is a DNF formula with k terms, then* BoundedSAT *takes $O(n^3 \cdot k \cdot p)$ time.*

Algorithm 5 ApproxMC($\varphi, \varepsilon, \delta$)

1: $t \leftarrow 35 \log(\frac{1}{\delta})$
2: $H \leftarrow$ PickHashFunctions($\mathcal{H}_{\mathsf{Toeplitz}}(n, n), t$)
3: $\mathcal{S} \leftarrow \{\}$;
4: Thresh $\leftarrow \frac{96}{\varepsilon^2}$
5: **for** $i \in [1, t]$ **do**
6: $\quad m_i \leftarrow 0$
7: $\quad c_i \leftarrow$ BoundedSAT($\varphi, H[i]|_{m_i}$, Thresh)
8: $\quad$ **while** $c_i \geq$ Thresh **do**
9: $\quad\quad m_i \leftarrow m_i + 1$
10: $\quad\quad c_i \leftarrow$ BoundedSAT($\varphi, H[i]|_{m_i}(x)$, Thresh)
11: $\quad \mathcal{S}[i] \leftarrow (c_i, m_i)$
12: $Est \leftarrow Median(\{\mathcal{S}[i](0) \times 2^{\mathcal{S}[i](1)}\}_i)$
13: **return** Est

Equipped with Proposition 1, we now turn to designing an algorithm for model counting based on the Bucketing strategy. The algorithm follows in similar fashion to its streaming counterpart where m_i is iteratively incremented until the number of solutions of the formula $(\varphi \wedge H[i]_{m_i}(x) = 0^{m_i})$ is less than Thresh. Interestingly, an approximate model counting algorithm, called ApproxMC, based on bucketing strategy was discovered independently by Chakraborty et. al. [13] in 2013. We reproduce an adaptation ApproxMC in Algorithm 5 to showcase how ApproxMC can be viewed as transformation of the Bucketing algorithm. In the spirit of Bucketing, ApproxMC seeks to construct a sketch $\mathcal{S}$ of size $t \in O(\log(1/\delta))$. To this end, for every iteration of the loop, we continue to increment the value of the loop until the conditions specified by the relation $\mathcal{P}_1(\mathcal{S}, H, \mathrm{Sol}(\varphi))$ are met. For every iteration i, the estimate of the model count is $c_i \times 2^{m_i}$. Finally, the estimate of the model count is simply the median of the estimation of all the iterations. Since in the context of model counting, we are concerned with time complexity, wherein both $\mathcal{H}_{\mathsf{Toeplitz}}$ and $\mathcal{H}_{\mathsf{xor}}$ lead to same time complexity. Furthermore, Chakraborty et al. [12] observed no difference in empirical runtime behavior due to $\mathcal{H}_{\mathsf{Toeplitz}}$ and $\mathcal{H}_{\mathsf{xor}}$.

The following theorem establishes the correctness of ApproxMC, and the proof follows from Lemma 1 and Proposition 1.

Theorem 2. *Given a formula φ, ε, and δ,* ApproxMC *returns Est such that* $\Pr[\frac{|\mathrm{Sol}(\varphi)|}{1+\varepsilon} \leq Est \leq (1+\varepsilon)|\mathrm{Sol}(\varphi)|] \geq 1 - \delta$. *If φ is a CNF formula, then this algorithm makes $O(n \cdot \frac{1}{\varepsilon^2} \log(1/\delta))$ calls to NP oracle. If φ is a DNF formula then* ApproxMC *is FPRAS. In particular for a DNF formula with k terms,* ApproxMC *takes $O(n^4 \cdot k \cdot \frac{1}{\varepsilon^2} \cdot \log(1/\delta))$ time.*

Further Optimizations. We now discuss how the setting of model counting allows for further optimizations. Observe that for all i, $\mathrm{Sol}(\varphi \wedge (H[i]_{m_i-1})(x) = 0^{m_i-1}) \subseteq \mathrm{Sol}(\varphi \wedge (H[i]_{m_i})(x) = 0^{m_i})$. Note that we are interested in finding the value of m_i such that $|\mathrm{Sol}(\varphi \wedge (H[i]_{m_i-1})(x) = 0^{m_i-1})| \geq \frac{96}{\varepsilon^2}$ and $|\mathrm{Sol}(\varphi \wedge (H[i]_{m_i})(x) = 0^{m_i})| < \frac{96}{\varepsilon^2}$, therefore, we can perform a binary search for m_i instead of a linear search performed in the loop 8– 10. Indeed, this observation was at the core of Chakraborty et al's followup work [14], which proposed the ApproxMC2, thereby reducing the number of calls to NP oracle from $O(n \cdot \frac{1}{\varepsilon^2} \log(1/\delta))$ to $O(\log n \cdot \frac{1}{\varepsilon^2} \log(1/\delta))$. Furthermore, the reduction in NP oracle calls led to significant runtime improvement in practice. It is worth commenting that the usage of ApproxMC2 as FPRAS for DNF is shown to achieve runtime efficiency over the alternatives based on Monte Carlo methods [44–46].

3.3 Minimum-based Algorithm

For a given multiset $\mathbf{a}$ (eg: a data stream or solutions to a model), we now specify the property $\mathcal{P}_2(\mathcal{S}, H, \mathbf{a}_u)$. The sketch $\mathcal{S}$ is an array of sets indexed by members of H that holds lexicographically p minimum elements of $H[i](\mathbf{a}_u)$ where p is $\min(\frac{96}{\varepsilon^2}, |\mathbf{a}_u|)$. $\mathcal{P}_2$ is the property that specifies this relationship. More formally, the relationship $\mathcal{P}_2$ holds, if the following conditions are met.

(1) $\forall i, |\mathcal{S}[i]| = \min(\frac{96}{\varepsilon^2}, |\mathbf{a}_u|)$
(2) $\forall i, \forall y \notin \mathcal{S}[i], \forall y' \in \mathcal{S}[i]$ it holds that $H[i](y') \preceq H[i](y)$

The following lemma due to Bar-Yossef *et al.* [7] establishes the relationship between the property $\mathcal{P}_2$ and the number of distinct elements of a multiset. Let $\max(S_i)$ denote the largest element of the set S_i.

Lemma 2. *[7] Let $\mathbf{a} \subseteq \{0,1\}^n$ be a multiset and $H \subseteq \mathcal{H}_{\mathsf{Toeplitz}}(n, m)$ where $m = 3n$ and each $H[i]$s are independently drawn and $|H| = O(\log 1/\delta)$ and let $\mathcal{S}$ be such that the $\mathcal{P}_2(\mathcal{S}, H, a_u)$ holds. Let $c =$* Median $\{\frac{p \cdot 2^m}{\max(S[i])}\}_i$. *Then*

$$\Pr\left[\frac{|\mathbf{a}_u|}{(1+\varepsilon)} \leq c \leq (1+\varepsilon)|\mathbf{a}_u|\right] \geq 1 - \delta.$$

Therefore, we can transform the Minimum algorithm for F_0 estimation to that of model counting given access to a subroutine that can compute $\mathcal{S}$ such that $\mathcal{P}_2(\mathcal{S}, H, \mathrm{Sol}(\varphi))$ holds true. The following proposition establishes the existence and complexity of such a subroutine, called FindMin:

Proposition 2. *There is an algorithm* FindMin *that, given φ over n variables, $h \in \mathcal{H}_{\mathsf{Toeplitz}}(n, m)$, and p as input, returns a set, $\mathcal{B} \subseteq h(\mathrm{Sol}(\varphi))$ so that if $|h(\mathrm{Sol}(\varphi))| \leq p$, then $\mathcal{B} = h(\mathrm{Sol}(\varphi))$, otherwise $\mathcal{B}$ is the p lexicographically minimum elements of $h(\mathrm{Sol}(\varphi))$. Moreover, if φ is a CNF formula, then* FindMin *makes $O(p \cdot m)$ calls to an NP oracle, and if φ is a DNF formula with k terms, then* FindMin *takes $O(m^3 \cdot n \cdot k \cdot p)$ time.*

Equipped with Proposition 2, we are now ready to present the algorithm, called ApproxModelCountMin, for model counting. Since the complexity of FindMin is PTIME when φ is in DNF, we have ApproxModelCountMin as a FPRAS for DNF formulas.

Theorem 3. *Given φ, ε,δ,* ApproxModelCountMin *returns c such that*

$$\Pr\left(\frac{|\mathrm{Sol}(\varphi)}{1+\varepsilon} \leq Est \leq (1+\varepsilon)|\mathrm{Sol}(\varphi)|\right) \geq 1 - \delta.$$

If φ is a CNF formula, then ApproxModelCountMin *is a polynomial-time algorithm that makes $O(\frac{1}{\varepsilon^2} n \log(\frac{1}{\delta}))$ calls to NP oracle. If φ is a DNF formula, then* ApproxModelCountMin *is an FPRAS.*

Implementing the Min-based Algorithm. We now give a proof of Proposition 2 by giving an implementation of FindMin subroutine.

Algorithm 6 ApproxModelCountMin($\varphi, \varepsilon, \delta$)

$t \leftarrow 35 \log(1/\delta)$
$H \leftarrow$ PickHashFunctions($\mathcal{H}_{\text{Toeplitz}}(n, 3n), t$)
$S \leftarrow \{\}$
Thresh $\leftarrow \frac{96}{\varepsilon^2}$
for $i \in [1, t]$ **do**
    $S[i] \leftarrow$ FindMin($\varphi, H[i]$, Thresh)
$Est \leftarrow$ Median $\left(\left\{\frac{\text{Thresh} \times 2^{3n}}{\max\{S[i]\}}\right\}_i\right)$
return Est

Proof. We first present the algorithm when the formula φ is a DNF formula. Adapting the algorithm for the case of CNF can be done by suing similar ideas.

Let $\phi = T_1 \vee T_2 \vee \cdots \vee T_k$ be a DNF formula over n variables where T_i is a term. Let $h : \{0,1\}^n \to \{0,1\}^m$ be a linear hash function in $\mathcal{H}_{xor}(n, m)$ defined by a $m \times n$ binary matrix A. Let C be the set of hashed values of the satisfying assignments for φ: $C = \{h(x) \mid x \models \varphi\} \subseteq \{0,1\}^m$. Let C_p be the first p elements of C in the lexicographic order. Our goal is to compute C_p.

We will give an algorithm with running time $O(m^3 np)$ to compute C_p when the formula is just a term T. Using this algorithm we can compute C_p for a formula with k terms by iteratively merging C_p for each term. The time complexity increases by a factor of k, resulting in an $O(m^3 nkp)$ time algorithm.

Let T be a term with width w (number of literals) and $C = \{Ax \mid x \models T\}$. By fixing the variables in T we get a vector b_T and an $N \times (n - w)$ matrix A_T so that $C = \{A_T x + b_T \mid x \in \{0,1\}^{(n-w)}\}$. Both A_T and b_T can be computed from A and T in linear time. Let $h_T(x)$ be the transformation $A_T x + b_T$.

We will compute C_p (p lexicographically minimum elements in C) iteratively as follows: assuming we have computed $(q-1)^{th}$ minimum of C, we will compute q^{th} minimum using a prefix-searching strategy. We will use a subroutine to solve the following basic prefix-searching primitive: Given any l bit string $y_1 \ldots y_l$, is there an $x \in \{0,1\}^{n-w}$ so that $y_1 \ldots y_l$ is a prefix for some string in $\{h_T(x)\}$? This task can be performed using Gaussian elimination over an $(l+1) \times (n-w)$ binary matrix and can be implemented in time $O(l^2(n-w))$.

Let $y = y_1 \ldots y_m$ be the $(q-1)^{th}$ minimum in C. Let r_1 be the rightmost 0 of y. Then using the above mentioned procedure we can find the lexicographically smallest string in the range of h_T that extends $y_1 \ldots y_{(r-1)}1$ if it exists. If no such string exists in C, find the index of the next 0 in y and repeat the procedure. In this manner the q^{th} minimum can be computed using $O(m)$ calls to the prefix-searching primitive resulting in an $O(m^3 n)$ time algorithm. Invoking the above procedure p times results in an algorithm to compute C_p in $O(m^3 np)$ time.

If φ is a CNF formula, we can employ the same prefix searching strategy. Consider the following NP oracle: $O = \{\langle \varphi, h, y, y' \rangle \mid \exists x, \exists y'',$ so that $x \models \varphi, y'y'' > y, h(x) = y'y''\}$. With m calls to O, we can compute string in C that is lexicographically greater than y. So with $p \cdot m$, calls to O, we can compute C_p.

□

Further Optimizations. As mentioned in Section 1, the problem of model counting has witnessed a significant interest from practitioners owing to its practical usages and the recent developments have been fueled by the breakthrough progress in the SAT solving wherein calls to NP oracles are replaced by invocations of SAT solver in practice. Motivated by the progress in SAT solving, there has been significant interest in design of efficient algorithmic frameworks for related problems such as MaxSAT and its variants. The state of the art MaxSAT based on sophisticated strategies such as implicit hitting sets and are shown to significant outperform algorithms based on merely invoking a SAT solver iteratively. Of particular interest to us is the recent progress in the design of MaxSAT solvers to handle lexicographic objective functions. In this context, it is worth remarking that we expect practical implementation of FindMin would invoke a MaxSAT solver $O(p)$ times.

3.4 Estimation-based Algorithm

We now adapt the Estimation algorithm to model counting. For a given stream $\mathbf{a}$ and chosen hash functions H, the sketch $\mathcal{S}$ corresponding to the estimation-based algorithm satisfies the following relation $\mathcal{P}_3(\mathcal{S}, H, \mathbf{a}_u)$:

$$\mathcal{P}_3(\mathcal{S}, H, \mathbf{a}_u) := \left(S[i,j] = \max_{x \in \mathbf{a}_u} \text{TrailZero}(H[i,j])(x)\right) \tag{2}$$

where the procedure TrailZero(z) is the length of the longest all-zero suffix of z. Bar-Yossef *et al.* [7] show the following relationship between the property $\mathcal{P}_3$ and F_0.

Lemma 3. *[7] Let $\mathbf{a} \subseteq \{0,1\}^n$ be a multiset. For $i \in [T]$ and $j \in [M]$, suppose $H[i,j]$ is drawn independently from $\mathcal{H}_{s-\text{wise}}(n,n)$ where $s = O(\log(1/\varepsilon))$, $T = O(\log(1/\delta))$, and $M = O(1/\varepsilon^2)$. Let H denote the collection of these hash functions. Suppose $\mathcal{S}$ satisfies $\mathcal{P}_3(\mathcal{S}, H, \mathbf{a}_u)$. For any integer r, define:*

$$c_r = \text{Median}\left\{\frac{\ln\left(1 - \frac{1}{M}\sum_{j=1}^{M} \mathbb{1}\{\mathcal{S}[i,j] \geq r\}\right)}{\ln(1 - 2^{-r})}\right\}_i$$

Then, if $2F_0 \leq 2^r \leq 50F_0$:

$$\Pr\left[(1-\varepsilon)F_0 \leq c_r \leq (1+\varepsilon)F_0\right] \geq 1 - \delta$$

Following the recipe outlined above, we can transform a F_0 streaming algorithm to a model counting algorithm by designing a subroutine that can compute the sketch for the set of all solutions described by φ and a subroutine to find r. The following proposition achieves the first objective for CNF formulas using a small number of calls to an NP oracle:

Proposition 3. *There is an algorithm* FindMaxRange *that given φ over n variables and hash function $h \in \mathcal{H}_{s-\text{wise}}(n,n)$, returns t such that*

(1) $\exists z, z \models \varphi$ and $h(z)$ has t least significant bits equal to zero.
(2) $\forall (z \models \varphi) \implies h(z)$ has $\leq t$ least significant bits equal to zero.

If φ is a CNF formula, then FindMaxRange *makes $O(\log n)$ calls to an NP oracle.*

Proof. Consider an NP oracle $O = \{\langle \varphi, h, t \rangle \mid \exists x, \exists y, x \models \varphi, h(x) = y0^t\rangle\}$. Note that h can be implemented as a degree-s

Algorithm 7 ApproxModelCountEst($\varphi, \varepsilon, \delta, r$)

Thresh $\leftarrow 96/\varepsilon^2$
$t \leftarrow 35\log(1/\delta)$
$H \leftarrow$ PickHashFunctions($\mathcal{H}_{s-\text{wise}}(n,n), t \times$ Thresh)
$S \leftarrow \{\}$
for $i \in [1,t]$ **do**
  for $j \in [1,$ Thresh$]$ **do**
    $S[i,j] \leftarrow$ FindMaxRange(φ, TrailZero($H[i,j]$))
$Est \leftarrow$ Median $\left\{ \frac{\ln\left(1-\frac{1}{\text{Thresh}}\sum_{j=1}^{\text{Thresh}} \mathbb{1}\{S[i,j]\geq r\}\right)}{\ln(1-2^{-r})} \right\}_i$
return Est

polynomial $h : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$, so that $h(x)$ can be evaluated in polynomial time. A binary search, requiring $O(\log n)$ calls to O, suffices to find the largest value of t for which $\langle \varphi, h, t\rangle$ belongs to O. □

We note that unlike Propositions 1 and 2, we do not know whether FindMaxRange can be implemented efficiently when φ is a DNF formula. For a degree-s polynomial $h : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$, we can efficiently test whether h has a root by computing $\gcd(h(x), x^{2^n} - x)$, but it is not clear how to simultaneously constrain some variables according to a DNF term.

Equipped with Proposition 3, we obtain ApproxModelCountEst that takes in a formula φ and a suitable value of r and returns $|\text{Sol}(\varphi)|$. The key idea of ApproxModelCountEst is to repeatedly invoke FindMaxRange for each of the chosen hash functions and compute the estimate based on the sketch $\mathcal{S}$ and the value of r. The following lemma summarizes the time complexity and guarantees of ApproxModelCountEst for CNF formulas.

Theorem 4. *Given a CNF formula φ, parameters ε and δ, and r such that $2F_0 \leq 2^r \leq 50F_0$, the algorithm* ApproxModelCountEst *returns c satisfying*

$$\Pr\left[\frac{|\text{Sol}(\varphi)}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\text{Sol}(\varphi)|\right] \geq 1-\delta.$$

ApproxModelCountEst *makes $O(\frac{1}{\varepsilon^2}\log n \log(\frac{1}{\delta}))$ calls to an NP oracle.*

In order to obtain r, we run in parallel another counting algorithm based on the simple F_0-estimation algorithm [3, 29] which we call FlajoletMartin. Given a stream $\mathbf{a}$, the FlajoletMartin algorithm chooses a random pairwise-independent hash function $h \in H_{xor}(n,n)$, computes the largest r so that for some $x \in \mathbf{a}_u$, the r least significant bits of $h(x)$ are zero, and outputs r. Alon, Matias and Szegedy [3] showed that 2^r is a 5-factor approximation of F_0 with probability 3/5. Using our recipe, we can convert FlajoletMartin into an algorithm that approximates the number of solutions to a CNF formula φ within a factor of 5 with probability 3/5. It is easy to check that using the same idea as in Proposition 3, this algorithm requires $O(\log n)$ calls to an NP oracle.

3.5 The Opportunities Ahead

As noted in Section 3.2, the algorithms based on Bucketing was already known and have witnessed a detailed technical development from both applied and algorithmic perspectives. The model counting algorithms based on Minimum and Estimation are new. We discuss some potential implications of these new algorithms to SAT solvers and other aspects.

MaxSAT solvers with native support for XOR constraints. When the input formula φ is represented as CNF, then ApproxMC, the model counting algorithm based on Bucketing strategy, invokes NP oracle over CNF-XOR formulas, i.e., formulas expressed as conjunction of CNF and XOR constraints. The significant improvement in runtime performance of ApproxMC owes to the design of SAT solvers with native support for CNF-XOR formulas [52–54]. Such solvers have now found applications in other domains such as cryptanalysis. It is perhaps worth emphasizing that the proposal of ApproxMC was crucial to renewed interest in the design of SAT solvers with native support for CNF-XOR formulas. As observed in Section 3.3, the algorithm based on Minimum strategy would ideally invoke a MaxSAT solver that can handle XOR constraints natively. We believe that Minimum-based algorithm will ignite interest in the design of MaxSAT solver with native support for XOR constraints.

FPRAS for DNF based on Estimation. In Section 3.4, we were unable to show that the model counting algorithm obtained based on Estimation is FPRAS when φ is represented as DNF. The algorithms based on Estimation have been shown to achieve optimal space efficiency in the context of F_0 estimation. In this context, an open problem is to investigate whether Estimation-based strategy lends itself to FPRAS for DNF counting.

Empirical Study of FPRAS for DNF Based on Minimum. Meel et al. [45, 46] observed that FPRAS for DNF based on Bucketing has superior performance, in terms of the number of instances solved, to that of FPRAS schemes based on Monte Carlo framework. In this context, a natural direction of future work would be to conduct an empirical study to understand behavior of FPRAS scheme based on Minimum strategy.

4 DISTRIBUTED DNF COUNTING

Consider the problem of *distributed DNF counting*. In this setting, there are k sites that can each communicate with a central coordinator. The input DNF formula φ is partitioned into k DNF subformulas $\varphi_1, \ldots, \varphi_k$, where each φ_i is a subset of the terms of the original φ, with the j'th site receiving only φ_j. The goal is for the coordinator to obtain an (ϵ, δ)-approximation of the number of solutions to φ, while minimizing the total number of bits communicated between the sites and the coordinator. Distributed algorithms for sampling and counting solutions to CSP's have been studied recently in other models of distributed computation [25–28]. From a practical perspective, given the centrality of #DNF in the context of probabilistic databases [48, 49], a distributed DNF counting would entail applications in distributed probabilistic databases.

From our perspective, distributed DNF counting falls within the *distributed functional monitoring* framework formalized by Cormode et al. [20]. Here, the input is a stream $\mathbf{a}$ which is partitioned arbitrarily into sub-streams $\mathbf{a}_1, \ldots, \mathbf{a}_k$ that arrive at each of k sites. Each site can communicate with the central coordinator, and the goal is for the coordinator to compute a function of the joint stream

$\Box$ while minimizing the total communication. This general framework has several direct applications and has been studied extensively [4, 6, 18, 21, 33, 40–42, 51, 61, 62, 64]. In distributed DNF counting, each sub-stream $\mathbf{a}_i$ corresponds to the set of satisfying assignments to each subformula φ_i, while the function to be computed is F_0.

The model counting algorithms discussed in Section 3 can be extended to the distributed setting. We briefly indicate the distributed implementations for each of the three algorithms. As above, we set the parameters Thresh to $O(1/\varepsilon^2)$. Correctness follows from Bar-Yossef *et al.* [7] and the earlier discussion.

We consider an adaptation of BoundedSAT that takes in φ over n variables, a hash function $h \in \mathcal{H}_{\mathsf{Toeplitz}}(n, m)$, and a threshold t as inputs, returns a set U of solutions such that $|U| = \min(t, |\mathsf{Sol}(\varphi \wedge h(x) = 0^m)|)$.

Bucketing. Setting $m = O(\log(k/\delta\varepsilon^2))$, the coordinator chooses $H[1], \ldots, H[T]$ from $\mathcal{H}_{\mathsf{Toeplitz}}(n, n)$ and G from $\mathcal{H}_{\mathsf{xor}}(n, m)$. It then sends them to the k sites. Let $m_{i,j}$ be the smallest m such that the size of the set $\mathsf{BoundedSAT}(\varphi_j, H[i]_m, \mathsf{thresh})$ is smaller than thresh. The j'th site sends the coordinator the following tuples: $\langle G(x), \mathsf{TrailZero}(H[i](x))\rangle$ for each $i \in [t]$ and for each x in $\mathsf{BoundedSAT}(\varphi_j, H[i]_{m_{i,j}}, \mathsf{thresh})$. Note that each site only sends tuples for at most $O(1/\delta\varepsilon^2)$ choices of x, so that G hashes these x to distinct values with probability $1 - \delta/2$. It is easy to verify that the coordinator can then execute the rest of the algorithm ApproxMC. The communication cost is $\tilde{O}(k(n + 1/\varepsilon^2) \cdot \log(1/\delta))$, and the time complexity for each site is polynomial in n, ε^{-1}, and $\log(\delta^{-1})$.

Minimum. The coordinator chooses hash functions $H[1], \ldots, H[t]$ from $\mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$ and sends it to the k sites. Each site runs the FindMin algorithm for each hash function and sends the outputs to the coordinator. So, the coordinator receives sets $S[i, j]$, consisting of the Thresh lexicographically smallest hash values of the solutions to φ_j. The coordinator then extracts $S[i]$, the Thresh lexicographically smallest elements of $S[i, 1] \cup \cdots \cup S[i, k]$ and proceeds with the rest of algorithm ApproxModelCountMin. The communication cost is $O(kn/\varepsilon^2 \cdot \log(1/\delta))$ to account for the k sites sending the outputs of their FindMin invocations. The time complexity for each site is polynomial in n, ε^{-1}, and $\log(\delta^{-1})$.

Estimation. For each $i \in [t]$, the coordinator chooses Thresh hash functions $H[i, 1], \ldots, H[i, \mathsf{Thresh}]$, drawn pairwise independently from $\mathcal{H}_{s-\mathsf{wise}}(n, n)$ (for $s = O(\log(1/\varepsilon))$) and sends it to the k sites. Each site runs the FindMaxRange algorithm for each hash function and sends the output to the coordinator. Suppose the coordinator receives $S[i, j, \ell] \in [n]$ for each $i \in [t], j \in [\mathsf{Thresh}]$ and $\ell \in [k]$. It computes $S[i, j] = \max_\ell S[i, j, \ell]$. The rest of ApproxModelCountEst is then executed by the coordinator. The communication cost is $\tilde{O}(k(n + 1/\varepsilon^2)\log(1/\delta))$. However, as earlier, we do not know a polynomial time algorithm to implement the FindMaxRange algorithm for DNF terms.

Lower Bound

The communication cost for the Bucketing- and Estimation-based algorithms is nearly optimal in their dependence on k and ε. Woodruff and Zhang [61] showed that the randomized communication complexity of estimating F_0 up to a $1 + \varepsilon$ factor in the distributed functional monitoring setting is $\Omega(k/\varepsilon^2)$. We can reduce F_0 estimation problem to distributed DNF counting. Namely, if for the F_0 estimation problem, the j'th site receives items $a_1, \ldots, a_m \in [N]$, then for the distributed DNF counting problem, φ_j is a DNF formula on $\lceil \log_2 N \rceil$ variables whose solutions are exactly $a_1, \ldots, a_m$ in their binary encoding. Thus, we immediately get an $\Omega(k/\varepsilon^2)$ lower bound for the distributed DNF counting problem. Finding the optimal dependence on N for $k > 1$ remains an interesting open question[3].

5 FROM COUNTING TO STREAMING: STRUCTURED SET STREAMING

In this section we consider *structured set streaming model* where each item S_i of the stream is a succinct representation of a set over the universe $U = \{0, 1\}^n$. Our goal is to design efficient algorithms (both in terms of memory and processing time per item) for computing $|\cup_i S_i|$ - number of distinct elements in the union of all the sets in the stream. We call this problem F_0 computation over structured set streams.

DNF Sets

A particular representation we are interested in is where each set is presented as the set of satisfying assignments to a DNF formula. Let φ is a DNF formula over n variables. Then the *DNF Set* corresponding to φ is the set of satisfying assignments of φ. The *size* of this representation is the number of terms in the formula φ.

A stream over DNF sets is a stream of DNF formulas $\varphi_1, \varphi_2, \ldots$. Given such a DNF stream, the goal is to estimate $|\bigcup_i S_i|$ where S_i the DNF set represented by φ_i. This quantity is same as the number of satisfying assignments of the formula $\vee_i \varphi_i$. We show that the algorithms described in the previous section carry over to obtain (ϵ, δ) estimation algorithms for this problem with space and per-item time $\mathrm{poly}(1/\epsilon, n, k, \log(1/\delta))$ where k is the size of the formula.

Notice that this model generalizes the traditional streaming model where each item of the stream is an element $x \in U$ as it can be represented as single term DNF formula ϕ_x whose only satisfying assignment is x. This model also subsumes certain other models considered in the streaming literature that we discuss later.

Theorem 5. *There is a streaming algorithm to compute an (ϵ, δ) approximation of F_0 over DNF sets. This algorithm takes space $O(\frac{n}{\varepsilon^2} \cdot \log \frac{1}{\delta})$ and processing time $O(n^4 \cdot k \cdot \frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta})$ per item where k is the size (number of terms) of the corresponding DNF formula.*

Proof. We show how to adapt Minimum-value based algorithm from Section 3.3 to this setting. The algorithm picks a hash function $h \in \mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$ maintains the set $\mathcal{B}$ consisting of t lexicographically minimum elements of the set $\{h(\mathsf{Sol}(\varphi_1, \vee \ldots \vee, \varphi_{i-1}))\}$ after processing $i - 1$ items. When φ_i arrives, it computes the set $\mathcal{B}'$ consisting of the t lexicographically minimum values of the set $\{h(\mathsf{Sol}(\varphi_i))\}$ and subsequently updates $\mathcal{B}$ by computing the t lexicographically smallest elements from $\mathcal{B} \cup \mathcal{B}'$. By Proposition 2,

[3] Note that if $k = 1$, then $\log(n/\varepsilon)$ bits suffices, as the site can solve the problem on its own and send to the coordinator the binary encoding of a $(1 + \varepsilon)$-approximation of F_0.

computation of $\mathcal{B}'$ can be done in time $O(n^4 \cdot k \cdot t)$ where k is the number of terms in φ_i. Updating $\mathcal{B}$ can be done in $O(t \cdot n)$ time. Thus update time for the item φ_i is $O(n^4 \cdot k \cdot t)$. For obtaining an (ε, δ) approximations we set $t = O(\frac{1}{\varepsilon^2})$ and repeat the procedure $O(\log \frac{1}{\delta})$ times and take the median value. Thus the update time for item φ is $O(n^4 \cdot k \cdot \frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta})$. For analyzing sapce, each hash function uses $O(n)$ bits and to store $O(\frac{1}{\epsilon^2})$ minimums, we require $O(\frac{n}{\epsilon^2})$ space resulting in overall space usage of $O(\frac{n}{\varepsilon^2} \cdot \log \frac{1}{\delta})$. The proof of correctness follows from Lemma 2. □

Instead of using Minimum-value based algorithm, we could adapt Bucketing-based algorithm to obtain an algorithm with similar space and time complexities. As noted earlier, some of the set streaming models considered in the literature can be reduced the DNF set streaming. We discuss them next.

Multidimensional Ranges

A d dimensional range over an universe U is defined as $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$. Such a range represents the set of tuples $\{(x_1, \ldots, x_d)$ where $a_i \leq x_i \leq b_i$ and x_i is an integer. Note that every d-dimensional range can be succinctly by the tuple $\langle a_1, b_1, \cdots a_d, b_d \rangle$. A multi-dimensional stream is a stream where each item is a d-dimensional range. The goal is to compute F_0 of the union of the d-dimensional ranges effeiciently. We will show that F_0 computation over multi-dimensional ranges can reduced to F_0 computation over DNF sets. Using this reduction we arrive at a simple algorithm to compute F_0 over multi-dimensional ranges.

Lemma 4. *Any d-dimensional range R over U can be represented as a DNF formula φ_R over nd variables whose size is at most $(2n)^d$. There is algorithm that takes R as input and outputs the i^{th} term of φ_R using $O(nd)$ space, for $1 \leq i \leq (2n)^d$.*

Proof. Let $R = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ be a d-dimensional range over U^d. We will first describe the formula to represent the multi-dimensional range as a conjunction of d DNF formulae $\phi_1, \cdots, \phi_d$ each with at most $2n$ terms, where ϕ_i represents $[a_i, b_i]$, the range in the i^{th} dimension. Converting this into a DNF formula will result in the formula ϕ_R with $(2n)^d$ terms.

For any ℓ bit number c, $1 \leq c \leq 2^n$, it is straightforward to write a DNF formula $\varphi_{\leq c}$, of size at most ℓ, that represents the range $[0, c]$ (or equivalently the set $\{x \mid 0 \leq x \leq c\}$). Similarly we can write a DNF formula $\varphi_{\geq c}$, of size at most ℓ for the range $[c, 2^{\ell-1}]$. Now we construct a formula to represent the range $[a, b]$ over U as follows. Let $a_1 a_2 \cdots a_n$ and $b_1 b_2 \cdots b_n$ be the binary representations of a and b respectively. Let ℓ be the largest integer such that $a_1 a_2 \cdots a_l = b_1 b_2 \cdots b_l$. Hence $a_{\ell+1} = 0$ and $b_{\ell+1} = 1$. Let a' and b' denote the integers represented by $a_{l+2} \cdots a_n$ and $b_{l+2} \cdots b_n$. Also, let ψ denote the formula (a single term) that represents the string $a_1 \cdots a_\ell$. Then the formula representing $[a, b]$ is $\psi \wedge (\overline{x_{\ell+1}} \varphi_{\geq a'} \vee x_{\ell+1} \varphi_{\leq b'})$. This can be written as a DNF formula by distributing ψ and the number of terms in the resulting formula is at most $2n$, and has n variables. Note that each φ_i can be constructed using $O(n)$ space. To obtain the final DNF representing the range R, we need to convert $\varphi_1 \wedge \cdots \varphi_d$ into a DNF formula. It is easy to see that for any i, then ith term of this DNF can be computed using space $O(nd)$. Note that this formula has nd variables, n variables per each dimension.

□

Using the above reduction and Theorem 5, we obtain an an algorithm for estimating F_0 over multidimensional ranges in a range-efficient manner.

Theorem 6. *There is a streaming algorithm to compute an (ϵ, δ) approximation of F_0 over d-dimensional ranges that takes space $O(\frac{nd}{\varepsilon^2} \cdot \log(1/\delta))$ and processing time $O((nd)^4 \cdot n^d \cdot \frac{1}{\varepsilon^2}) \log(1/\delta))$ per item.*

Remark 1. *Tirthapura and Woodruff [58] studied the problem of range efficient estimation of F_k (k^{th} frequency moments) over d-dimensional ranges. They claimed an algorithm to estimate F_0 with space and per-item time complexity* $\text{poly}(n, d, 1/\epsilon, \log 1/\delta)$. *However they have retracted their claim [60]. Their method only yields* $\text{poly}(n^d, 1/\epsilon, \log 1/\delta)$ *time per item. Their proof appears to be involved that require a range efficient implementations of* count sketch *algorithm [15] and recursive sketches [9, 34]. We obtain the same complexity bounds with much simpler analysis and a practically efficient algorithm that can use off the shelf available implementations [45].*

Remark 2. *Subsequent to the present work, an improved algorithm for F_0 over structured sets is presented in [57] (to appear in PODS 2021). In particular, the paper presents an F_0 estimation algorithm, called* APS-Estimator, *for streams over* Delphic sets. *A set $S \subseteq \{0,1\}^n$ belongs to Delphic family if the following queries can be done in $O(n)$ time: (1) know the size of the set S, (2) draw a uniform random sample from S, and (3) given any x check if $x \in S$. The authors design a streaming algorithm that given a stream $\mathcal{S} = \langle S_1, S_2 \cdots, S_M \rangle$ wherein each $S_i \subseteq \{0,1\}^n$ belongs to Delphic family, computes an (ε, δ)-approximation of $|\bigcup_{i=1}^{M} S_i|$ with worst case space complexity $O(n \cdot \log(M/\delta) \cdot \varepsilon^{-2})$ and per-item time is $\widetilde{O}(n \cdot \log(M/\delta) \cdot \varepsilon^{-2})$. The algorithm* APS-Estimator, *when applied to d-dimensional ranges, gives per-item time and space complexity bounds that are* $\text{poly}(n, d, \log M, 1/\varepsilon, \log 1/\delta)$. *While* APS-Estimator *brings down the dependency on d from exponential to polynomial, it works under the assumption that the length of the stream M is known. The general setup presented in [57], however, can be applied to other structured sets considered in this paper including multidimensional arithmetic progressions.*

Representing Multidimensional Ranges as CNF Formulas. Since the algorithm, APS-Estimator, presented in [57], employs a sampling-based technique, a natural question is whether there exists a hashing-based technique that achieves per-item time polynomial in n and d. We note that the above approach of representing a multi-dimensional range as DNF formula does not yield such an algorithm. This is because there exist d-dimensional ranges whose DNF representation requires $\Omega(n^d)$ size.

Observation 1. *There exist d-dimensional ranges whose DNF representation has size $\geq n^d$.*

Proof. The observation follows by considering the range $R = [1, 2^n - 1]^d$ (only 0 is missing from the interval in each dimension). We will argue that any DNF formula φ for this range has size (number of terms) $\geq n^d$. For any $1 \leq j \leq d$, we use the set of variables

$X^j = \{x_1^j, x_2^j, \ldots, x_n^j\}$ for representing the j^{th} coordinate of R. Then R can be represented as the formula $\varphi_R = \vee_{(i_1,i_2,\ldots,i_d)} x_{i_1}^1 x_{i_2}^2 \ldots x_{i_d}^d$, where $1 \leq i_j \leq n$. This formula has n^d terms. Let φ be any other DNF formula representing R. The main observation is that any term T of φ is completely contained (in terms of the set of solutions) in one of the terms of φ_R. This implies that φ should have n^d terms. Now we argue that T is contained in one of the terms of φ_R. T should have at least one variable as positive literal from each of X^j. Suppose T does not have any variable from X^j for some j. Then T contains a solution with all the variable in X^j set to 0 and hence not in R. Now let $x_{i_j}^j$ be a variable from X^j that is in T. Then clearly T is in the term $x_{i_1}^1 x_{i_2}^2 \ldots x_{i_d}^d$ of R. □

This leads to the question of whether we can obtain a super-polynomial lower bound on the time per item. We observe that such a lower bound would imply P ≠ NP. For this, we note the following.

Observation 2. *Any d-dimensional range R can be represented as a CNF formula of size $O(nd)$ over nd variables.*

This is because a single dimensional range $[a, b]$ can also be represented as a CNF formula of size $O(n)$ [11] and thus the CNF formula for R is a conjunction of formulas along each dimension. Thus the problem of computing F_0 over d-dimensional ranges reduces to computing F_0 over a stream where each item of the stream is a CNF formula. As in the proof of Theorem 5, we can adapt Minimum-value based algorithm for CNF streams. When a CNF formula φ_i arrive, we need to compute the t lexicographically smallest elements of $h(\mathrm{Sol}(\varphi_i))$ where $h \in \mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$. By Proposition 2, this can be done in polynomial-time by making $O(tnd)$ calls to an NP oracle since ϕ_i is a CNF formula over nd variables. Thus if P equals NP, then the time taken per range is polynomial in n, d, and $1/\varepsilon^2$. Thus a super polynomial time lower bound on time per item implies that P differs from NP.

From Weighted #DNF to d-Dimensional Ranges. Designing a streaming algorithm with a per item of polynomial in n and d is a very interesting open problem with implications on weighted DNF counting. Consider a formula φ defined on the set of variables $x = \{x_1, x_2, \ldots x_n\}$. Let a weight function $\rho : x \mapsto (0, 1)$ be such that weight of an assignment σ can be defined as follows:

$$W(\sigma) = \prod_{x_i:\sigma(x_i)=1} \rho(x_i) \prod_{x_i:\sigma(x_i)=0} (1 - \rho(x_i))$$

Furthermore, we define the weight of a formula φ as

$$W(\varphi) = \sum_{\sigma \models \varphi} W(\sigma)$$

Given φ and ρ, the problem of weighted counting is to compute $W(\varphi)$. We consider the case where for each x_i, $\rho(x_i)$ is represented using m_i bits in binary representation, i.e., $\rho(x_i) = \frac{k_i}{2^{m_i}}$. Inspired by the key idea of weighted to unweighted reduction due to Chakraborty et al. [11], we show how the problem of weighted DNF counting can be reduced to that of estimation of F_0 estimation of n-dimensional ranges. The reduction is as follows: we transform every term of φ into a product of multi-dimension ranges where every variable x_i is replaced with interval $[1, k_i]$ while $\neg x_i$ is replaced with $[k_i+1, 2^{m_i}]$ and every $\wedge$ is replaced with $\times$. For example, a term $(x_1 \wedge \neg x_2 \wedge \neg x_3)$ is replaced with $[1, k_1] \times [k_2+1, 2^{m_2}] \times [k_3+1, 2^{m_3}]$. Given F_0 of the resulting stream, we can compute the weight of φ simply as $W(\varphi) = \frac{F_0}{2^{\sum_i m_i}}$. Thus a hashing based streaming algorithm that has $poly(n, d)$ time per item, yields a hashing based FPRAS for weighted DNF counting, and open problem from [1].

Multidimensional Arithmetic Progressions. We will now generalize Theorem 6 to handle arithmetic progressions instead of ranges. Let $[a, b, c]$ represent the arithmetic progression with common difference c in the range $[a, b]$, i.e., $a, a + c, a + 2c, a + id$, where i is the largest integer such that $a+id \leq b$. Here, we consider d-dimensional arithmetic progressions $R = [a_1, b_1, c_1] \times \cdots \times [a_d, b_d, c_d]$ where each c_i is a power two. We first observe that the set represented by $[a, b, 2^\ell]$ can be expressed as a DNF formula as follows: Let ϕ be the DNF formula representing the range $[a, b]$ and let $a_1, \cdots, a_\ell$ are the least significant bits of a, Let ψ be the term that represents the bit sequence $a_1 \cdots a_\ell$. Now the formula to represent the arithmetic progression $[a, b, 2^\ell]$ is $\phi \wedge \psi$ which can be converted to a DNF formula of size $O(2n)$. Thus the multi-dimensional arithmetic progression R can be represented as a DNF formula of size $(2n)^d$. Note that time and space required to convert R into a DNF formula are as before, i.e, $O(n^d)$ time and $O(nd)$ space. This leads us to the following corollary.

Corollary 1. *There is a streaming algorithm to compute an (ϵ, δ) approximation of F_0 over d-dimensional arithmetic progressions, whose common differences are powers of two, that takes space $O(nd/\varepsilon^2 \cdot \log 1/\delta)$ and processing time $O((nd)^4 \cdot n^d \cdot \frac{1}{\varepsilon^2}) \log(1/\delta))$ per item.*

Affine Spaces

Another example of structured stream is where each item of the stream is an affine space represented by $Ax = B$ where A is a boolean matrix and B is a zero-one vector. Without loss of generality, we may assume that where A is a $n \times n$ matrix. Thus an affine stream consists of $\langle A_1, B_\rangle, \langle A_2, B_2\rangle \cdots$, where each $\langle A_i, B_i\rangle$ is succinctly represents a set $\{x \in \{0, 1\}^n \mid A_i x = B_i\}$.

For a $n \times n$ Boolean matrix A and a zero-one vector B, let $\mathrm{Sol}(\langle A, B\rangle)$ denote the set of all x that satisfy $Ax = B$.

Proposition 4. *Given (A, B), $h \in \mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$, and t as input, there is an algorithm,* AffineFindMin, *that returns a set, $\mathcal{B} \subseteq h(\mathrm{Sol}(\langle A, B\rangle))$ so that if $|h(\mathrm{Sol}(\langle A, B\rangle))| \leq t$, then $\mathcal{B} = h(\mathrm{Sol}(\langle A, B\rangle))$, otherwise $\mathcal{B}$ is the t lexicographically minimum elements of $h(\mathrm{Sol}(\langle A, B\rangle))$. Time taken by this algorithm is $O(n^4 t)$ and the space taken the algorithm is $O(tn)$.*

Proof. Let D be the matrix that specifies the hash function h. Let $C = \{Dx \mid Ax = B\}$, and the goal is to compute the t smallest element of C. Note that if $y \in C$, then it must be the case that $D|Ax = y|B$ where $D|A$ is the matrix obtained by appending rows of A to the rows of D (at the end), and $y|B$ is the vector obtained by appending B to y. Note that $D|A$ is a matrix with $4n$ rows. Now the proof is very similar to the proof of Proposition 2. We can do a prefix search as before and this involves doing Gaussian elimination using sub matrices of $D|A$. □

Theorem 7. *There is a streaming algorithms computes (ϵ, δ) approximation of F_0 over affine spaces. This algorithm takes space $O(\frac{n}{\epsilon^2} \cdot \log(1/\delta))$ and processing time of $O(n^4 \frac{1}{\epsilon^2} \log(1/\delta))$ per item.*

6 CONCLUSION AND FUTURE OUTLOOK

To summarize, our investigation led to a diverse set of results that unify over two decades of work in model counting and F_0 estimation. We believe that the viewpoint presented in this work has potential to spur several new interesting research directions. We sketch some of these directions below:

Sampling The problem of counting and sampling are closely related. In particular, the seminal work of Jerrum, Valiant, and Vazirani [36] showed that the problem of approximate counting and almost-uniform sampling are inter-reducible for self-reducible NP problems. Concurrent to developments in approximate model counting, there has been a significant interest in the design of efficient sampling algorithms. A natural direction would be to launch a similar investigation.

Higher Moments There has been a long line of work on estimation of higher moments, i.e. F_k in streaming context. A natural direction of future research is to adapt the notion of F_k in the context of CSP. For example, in the context of DNF, one can view F_1 be simply a sum of the size of clauses but it remains to be seen to understand the relevance and potential applications of higher moments such as F_2 in the context of CSP. Given the similarity of the core algorithmic frameworks for higher moments, we expect extension of the framework and recipe presented in the paper to derive algorithms for higher moments in the context of CSP.

Sparse XORs In the context of model counting, the performance of underlying SAT solvers strongly depends on the size of XORs. The standard construction of $\mathcal{H}_{\mathsf{Toeplitz}}$ and $\mathcal{H}_{\mathsf{xor}}$ lead to XORs of size $\Theta(n/2)$ and interesting line of research has focused on the design of sparse XOR-based hash functions [2, 5, 24, 32, 35] culminating in showing that one can use hash functions of form where $h(x) = Ax + b$ wherein each entry of m-th row of A is 1 with probability $O(\frac{\log m}{m})$ [43]. Such XORs were shown to improve the runtime efficiency. In this context, a natural direction would be to explore the usage of sparse XORs in the context of F_0 estimation.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of PODS 21 for valuable comments. Bhattacharyya was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0002] and an Amazon Research Award. Meel was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme[NRF-NRFFAI1-2019-0004] and AI Singapore Programme [AISG-RP-2018-005], and NUS ODPRT Grant [R-252-000-685-13]. Vinod was supported in part by NSF CCF-184908 and NSF HDR:TRIPODS-1934884 awards. Pavan was supported in part by NSF CCF-1849053 and NSF HDR:TRIPODS-1934884 awards.

REFERENCES

[1] Ralph Abboud, Ismail Ilkan Ceylan, and Thomas Lukasiewicz. 2019. Learning to Reason: Leveraging Neural Networks for Approximate DNF Counting. *arXiv preprint arXiv:1904.02688* (2019).
[2] Dimitris Achlioptas and Panos Theodoropoulos. 2017. Probabilistic model counting with short XORs. In *Proc. of SAT*. Springer, 3–19.
[3] Noga Alon, Yossi Matias, and Mario Szegedy. 1999. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.* 58, 1 (1999), 137–147.
[4] Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. 2009. Functional monitoring without monotonicity. In *International Colloquium on Automata, Languages, and Programming*. Springer, 95–106.
[5] Megasthenis Asteris and Alexandros G Dimakis. 2016. *LDPC codes for discrete integration*. Technical Report. Technical report, UT Austin.
[6] Brian Babcock and Chris Olston. 2003. Distributed top-k monitoring. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 28–39.
[7] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. [n. d.]. Counting Distinct Elements in a Data Stream. In *Proc. of RANDOM*, Vol. 2483. 1–10.
[8] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. of SODA*. ACM/SIAM, 623–632.
[9] Vladimir Braverman and Rafail Ostrovsky. 2010. Recursive Sketching For Frequency Moments. *CoRR* abs/1011.2571 (2010).
[10] J Lawrence Carter and Mark N Wegman. 1977. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*. ACM, 106–112.
[11] Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. 2015. From weighted to unweighted model counting. In *Proceedings of AAAI*. 689–695.
[12] S. Chakraborty, K. S. Meel, and M. Y. Vardi. 2013. A Scalable and Nearly Uniform Generator of SAT Witnesses. In *Proc. of CAV*. 608–623.
[13] S. Chakraborty, K. S. Meel, and M. Y. Vardi. 2013. A Scalable Approximate Model Counter. In *Proc. of CP*. 200–216.
[14] S. Chakraborty, K. S. Meel, and M. Y. Vardi. 2016. Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls. In *Proc. of IJCAI*.
[15] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. 2004. Finding frequent items in data streams. *Theor. Comput. Sci.* 312, 1 (2004), 3–15.
[16] Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. 2015. Approximate counting in SMT and value estimation for probabilistic programs. In *Proc. of TACAS*. Springer, 320–334.
[17] Jeffrey Considine, Feifei Li, George Kollios, and John W. Byers. 2004. Approximate Aggregation Techniques for Sensor Databases. In *Proc. of ICDE*, Z. Meral Özsoyoglu and Stanley B. Zdonik (Eds.). IEEE Computer Society, 449–460.
[18] Graham Cormode, Minos Garofalakis, Shanmugavelayutham Muthukrishnan, and Rajeev Rastogi. 2005. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proc. of SIGMOD*. 25–36.
[19] Graham Cormode and S. Muthukrishnan. 2003. Estimating Dominance Norms of Multiple Data Streams. In *Proc. of ESA (Lecture Notes in Computer Science)*, Giuseppe Di Battista and Uri Zwick (Eds.), Vol. 2832. Springer, 148–160.
[20] Graham Cormode, Shanmugavelayutham Muthukrishnan, and Ke Yi. 2011. Algorithms for distributed functional monitoring. *ACM Transactions on Algorithms (TALG)* 7, 2 (2011), 1–20.
[21] Graham Cormode, Shanmugavelayutham Muthukrishnan, Ke Yi, and Qin Zhang. 2012. Continuous sampling from distributed streams. *Journal of the ACM (JACM)* 59, 2 (2012), 1–25.
[22] P. Dagum, R. Karp, M. Luby, and S. Ross. 2000. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on computing* 29, 5 (2000), 1484–1496.
[23] Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2013. Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization. In *Proc. of ICML*. 334–342.
[24] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. 2014. Low-density Parity Constraints for Hashing-Based Discrete Integration. In *Proc. of ICML*. 271–279.
[25] Weiming Feng, Thomas P Hayes, and Yitong Yin. 2018. Distributed symmetry breaking in sampling (optimal distributed randomly coloring with fewer colors). *arXiv preprint arXiv:1802.06953* (2018).
[26] Weiming Feng, Yuxin Sun, and Yitong Yin. 2018. What can be sampled locally? *Distributed Computing* (2018), 1–27.
[27] Weiming Feng and Yitong Yin. 2018. On local distributed sampling and counting. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. 189–198.
[28] Manuela Fischer and Mohsen Ghaffari. 2018. A Simple Parallel and Distributed Sampling Technique: Local Glauber Dynamics. In *32nd International Symposium on Distributed Computing*.
[29] Philippe Flajolet and G. Nigel Martin. 1985. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.* 31, 2 (1985), 182–209.
[30] Phillip B. Gibbons and Srikanta Tirthapura. 2001. Estimating simple functions on the union of data streams. In *Proc. of SPAA*, Arnold L. Rosenberg (Ed.). ACM, 281–291.

[31] C.P. Gomes, A. Sabharwal, and B. Selman. 2007. Near-Uniform sampling of combinatorial spaces using XOR constraints. In *Proc. of NIPS*. 670–676.

[32] Carla P Gomes, Joerg Hoffmann, Ashish Sabharwal, and Bart Selman. 2007. From Sampling to Model Counting.. In *Proc. of IJCAI*. 2293–2299.

[33] Zengfeng Huang, Ke Yi, and Qin Zhang. 2012. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *Proc. of PODS*. 295–306.

[34] Piotr Indyk and David P. Woodruff. 2005. Optimal approximations of the frequency moments of data streams. In *Proc. of STOC*. ACM, 202–208.

[35] Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. 2015. On computing minimal independent support and its applications to sampling and counting. *Constraints* (2015), 1–18. https://doi.org/10.1007/s10601-015-9204-z

[36] M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. 1986. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science* 43, 2-3 (1986), 169–188.

[37] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. 2010. An optimal algorithm for the distinct elements problem. In *Proc. of PODS*. ACM, 41–52.

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

[39] Richard M Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms* 10, 3 (1989), 429 – 448. https://doi.org/10.1016/0196-6774(89)90038-2

[40] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. 2006. Communication-efficient distributed monitoring of thresholded counts. In *Proc. of SIGMOD*. 289–300.

[41] Daniel Keren, Izchak Sharfman, Assaf Schuster, and Avishay Livne. 2011. Shape sensitive geometric monitoring. *IEEE Transactions on Knowledge and Data Engineering* 24, 8 (2011), 1520–1535.

[42] Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston. 2005. Finding (recently) frequent items in distributed data streams. In *Proc. of ICDE*. IEEE, 767–778.

[43] Kuldeep S. Meel Ⓡ S. Akshay. 2020. Sparse Hashing for Scalable Approximate Model Counting: Theory and Practice. In *Proc. of LICS*.

[44] Kuldeep S Meel, Aditya A Shrotri, and Moshe Y Vardi. 2017. On Hashing-Based Approaches to Approximate DNF-Counting. In *In Proc. of FSTTCS*.

[45] Kuldeep S. Meel, Aditya A. Shrotri, and Moshe Y. Vardi. 2018. Not All FPRASs are Equal: Demystifying FPRASs for DNF-Counting. (12 2018).

[46] Kuldeep S. Meel, Aditya A. Shrotri, and Moshe Y. Vardi. 2019. Not All FPRASs are Equal: Demystifying FPRASs for DNF-Counting (Extended Abstract). In *Proc. of IJCAI*.

[47] A. Pavan and Srikanta Tirthapura. 2007. Range-Efficient Counting of Distinct Elements in a Massive Data Stream. *SIAM J. Comput.* 37, 2 (2007), 359–379.

[48] Christopher Ré and Dan Suciu. 2008. Approximate lineage for probabilistic databases. *Proceedings of the VLDB Endowment* 1, 1 (2008), 797–808.

[49] Pierre Senellart. 2018. Provenance and probabilities in relational databases. *ACM SIGMOD Record* 46, 4 (2018), 5–15.

[50] Pierre Senellart. 2019. Provenance in Databases: Principles and Applications. In *Reasoning Web. Explainable Artificial Intelligence*. Springer, 104–109.

[51] Izchak Sharfman, Assaf Schuster, and Daniel Keren. 2010. A geometric approach to monitoring threshold functions over distributed data streams. In *Ubiquitous knowledge discovery*. Springer, 163–186.

[52] Mate Soos, Stephan Gocht, and Kuldeep S. Meel. 2020. Tinted, Detached, and Lazy CNF-XOR solving and its Applications to Counting and Sampling. In *Proceedings of International Conference on Computer-Aided Verification (CAV)*.

[53] Mate Soos and Kuldeep S Meel. 2019. BIRD: Engineering an Efficient CNF-XOR SAT Solver and its Applications to Approximate Model Counting. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)(1 2019)*.

[54] Mate Soos, Karsten Nohl, and Claude Castelluccia. 2009. Extending SAT Solvers to Cryptographic Problems. In *Proc. of SAT*. 244–257.

[55] L. Stockmeyer. 1983. The complexity of approximate counting. In *Proc. of STOC*. 118–126.

[56] He Sun and Chung Keung Poon. 2009. Two improved range-efficient algorithms for F_0 estimation. *Theor. Comput. Sci.* 410, 11 (2009), 1073–1080.

[57] Kuldeep S. Meel Ⓡ N. V. Vinodchandran Ⓡ Sourav Chakraborty. 2021. Estimating Size of Union of Sets in Streaming Model. In *Proc. of PODS 2021*.

[58] Srikanta Tirthapura and David P. Woodruff. 2012. Rectangle-efficient aggregation in spatial data streams. In *Proc. of PODS*. ACM, 283–294.

[59] L.G. Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.

[60] David Woodruff. [n. d.]. personal communication.

[61] David P Woodruff and Qin Zhang. 2012. Tight bounds for distributed functional monitoring. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. 941–960.

[62] David P Woodruff and Qin Zhang. 2017. When distributed computation is communication expensive. *Distributed Computing* 30, 5 (2017), 309–323.

[63] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of artificial intelligence research* 32 (2008), 565–606.

[64] Ke Yi and Qin Zhang. 2013. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica* 65, 1 (2013), 206–223.